The Adaptable IO System (ADIOS)

David Pugmire[1], Norbert Podhorszki[1], Scott Klasky[1], Matthew Wolf [1], James Kress[1], Mark Kim[1], Nicholas Thompson[1], Jeremy Logan[1], Ruonan Wang[1], Kshitij Mehta[1], Eric Suchyta[1], William Godoy[1], Jong Choi[1], George Ostrouchov[1], Lipeng Wan[1], Jieyang Chen[1], Berk Geveci[2] Chuck Atkins[2], Caitlin Ross[2], Greg Eisenhauer[3], Junmin Gu[4], John Wu[4], Axel Huebl[4], Seiji Tsutsumi[5]

Abstract The Adaptable I/O System (ADIOS) provides a publish/subscribe abstraction for data access and storage. The framework provides various engines for producing and consuming data through different mediums (storage, memory, network) for various application scenarios. ADIOS engines exist to write/read files on a storage system, to couple independent simulations together or to stream data from a simulation to analysis and visualization tools via the computer's network infrastructure, and to stream experimental/observational data from the producer to data processors via the wide-area-network. Both lossy and lossless compression are supported by ADIOS to provide for seamless exchange of data between producer and consumer. In this work we provide a description for the ADIOS framework and the abstractions provided. We demonstrate the capabilities of the ADIOS framework using a number of examples, including strong coupling of simulation codes, in situ visualization running on a separate computing cluster, and streaming of experimental data between Asia and the United States.

1 Introduction

The Adaptable I/O System (ADIOS) was designed with the observation that applications almost universally read and write files from storage, and that this can be used as an abstraction for access to data [13]. ADIOS is a middleware layer that sits between the application and the computing system to manage the movement of data. This middleware layer makes it possible for an application to write data to a target that is determined at runtime. One possible target is traditional file storage. Other targets are able to support in situ processing methods, and include a memory buffer on the nodes where the application is running, or over the network to a memory buffer on another set of resources. Applications (e.g., visualization and analysis codes) can

[1]Oak Ridge National Laboratory · [2]Lawrence Berkeley National Laboratory · [3]Kitware, Inc. · [4]Georgia Institute of Technology · [5]Japan Aerospace Exploration Agency

read data from any of the file or in situ targets. A schematic showing examples of these use cases in given in Figure 1. The advantage of this design is that the sharing and movement of data is decoupled from the producer and the consumer, and can be modified at runtime as needed. This advantage addresses several of the challenges described in Section **??** of Chapter **??**. Workflow execution is made easier when data producers and consumers can be connected together without modifying the source codes. Software complexity is reduced because the issues related to data access across evolving systems are provided by the middleware layer. Better resilience is possible because the producer and consumer need not run together in the same memory space.

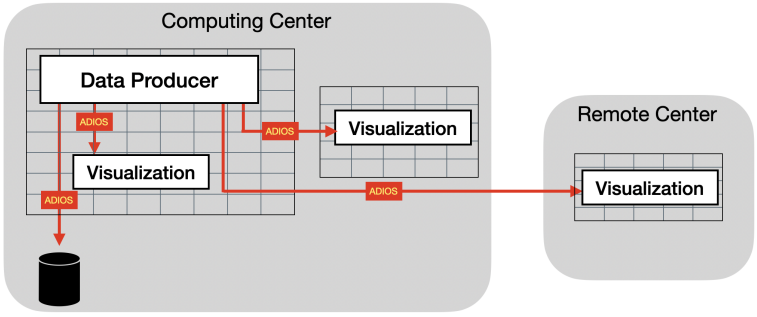

Fig. 1 Examples of ADIOS usage. In this example, the producer can write data to disk, to a visualization process sharing the same resource as the simulation, to a separate set of nodes, or across the network to a remote computing center.

In terms of the taxonomy described in Section **??** of Chapter **??**, ADIOS can be classified in the following ways. *Integration type*: Apart from the usage of the ADIOS API for I/O, no additional instrumentation is needed by the application. *Proximity*: The visualization can run on the same or different computing resources. *Access*: The visualization can directly access memory in the simulation, or it can be copied to a memory buffer on the same or on different computing resources. *Division of Execution*: Synchronous or asynchronous are both possible, providing support for both time and space division. *Operation Controls*: Human in the loop is possible using both synchronous (blocking) or asynchronous (non-blocking) modes.

The ADIOS framework design was based on the goal to provide an I/O abstraction for parallel and distributed applications that expresses *what* data is produced for output and *when* that data is ready for output, or what data an application wants to read and when[22, 23, 16]. This is achieved in ADIOS through the use of different types of I/O *engines*. When an application writes a set of data, it uses the appropriate engine (e.g., File engine, In situ engine, etc) to do the actual data movement. Similarly for a reader, it uses the appropriate Engine to request the data that it needs. For flexibility, the types of engine used by producer and consumer can be specified in a configuration file for runtime selection. In this way, applications that either read

or write data need only select the appropriate output engine, and do not need to be concerned with the implementation details needed to achieve scalable performance.

The separation of concerns, namely that an application only need to be concerned about the data production and consumption but not how the data should be delivered, allows for creating optimized ADIOS engines that all can work with the same application code. Using the ADIOS interface makes application I/O *scalable*, a primary goal of the ADIOS framework, which is designed to work well on the largest supercomputers. ADIOS regularly runs on the largest supercomputers in the world for applications that consume and produce multiple petabytes of data during the course of a simulation run.

In this chapter we describe details of the ADIOS framework and how it can be used for in situ visualization. In Section 2 we describe the I/O abstractions used by ADIOS and how applications and visualization codes can use them. Section 2.1 provides a description of the engines provided by ADIOS and how they handle the movement of data. Advanced features in ADIOS are described in Section 2.2, followed by a discussion on the relative strengths of each engine and coding examples in Sections 2.3 and 2.4. In Section 3 we describe the use of ADIOS for in situ visualization with application partners, followed by some concluding remarks in Section 4.

2 ADIOS I/O Abstraction

A parallel application that produces data uses ADIOS to define *variables* (*n*-dimensional distributed arrays of a particular type) and *attributes* (labels associated with individual variables or the entire output data set). It also specifies when the data is available for output. The output is organized around output *Steps*, instead of individual variables. A Step includes all the variables and attributes that are to be sent to the target at once. There is nothing in the ADIOS interface that prescribes how to handle the data (e.g. data aggregation among the processes, targeting a single file or one file per process, handling multiple readers, and handling the disappearance of a potential reader). These belong to the *IO strategy* and are implemented in various ways by different Engines. The user can control the behavior of the application by choosing a specific engine, and parameterizing it with available options.

Similarly, a reading application only declares what data it wants to retrieve from a source, each process of the parallel application declaring what it needs, and when it expects the data to be in its memory. The input is also organized around Steps, not individual variables. The semantics of the ADIOS API ensure that a reader never gets into an inconsistent state where portion of the data of some variables belong to a certain step, and other portion to another step.

Analysis and visualization are typically data-hungry operations [7]. This makes scalable access to data key. In an in situ environment, the ability to maintain clear boundaries between simulation and analysis tasks can promote fault tolerance, interoperability, programmability and maintainability.

The flexibility of the data movement abstractions provided by ADIOS makes it easy to integrate with analysis and visualization applications. The "file-like" API provided by ADIOS allows seamless reads from disk, from memory, or streaming over the network. Likewise, on the write side, outputs produced by analysis and visualization applications can be written to disk, or shared with other applications through memory or streamed over the network. The abstraction used by ADIOS makes it easy to move data as it does something that the application is already doing, namely, reading and writing from files.

The abstraction provides the same access to data regardless of where the data are located, be it disk, memory or streaming. As examples, the VisIt [6] and ParaView [1] visualization tools have support for reading data from ADIOS in this manner. Both tools provide access to ADIOS data using a data reader plugin. The plugin reads the ADIOS data and creates mesh-based data that can be visualized by the VisIt and ParaView tools. An example of VisIt visualizing streaming data is described in Section 3.3.

When visualization is used in an in situ environment, the ability to have clear boundaries between the simulation and analysis and visualization, and rely on a middleware layer for the sharing and exchange of data is valuable in a number of ways. (1) It makes it much easier to reconfigure components in a workflow based on the needs of the scientific campaign. (2) Fault tolerance is increased because the simulation can be separated from the visualization. (3) The mechanism for sharing data between producer and consumer can more easily be modified, changed or replaced.

In the remainder of this section we describe ADIOS in more detail. In Section 2.1 and 2.2 we describe the ADIOS engines used to move data and some advanced topics on reduction and data interpretation. In Section 2.3 we discuss the characteristics of these engines and how they relate to visualization and analysis needs and costs. Finally, in Section 2.4 we show some code examples of how ADIOS is used for both data producers and data consumers.

2.1 ADIOS Engines

The mechanism in ADIOS for moving data is called an *engine*. An engine is tasked with executing the I/O heavy operations associated with the movement of data. Each engine supports a unified interface that allows data producers to *put* data, and data consumers to *get* data. The details of moving the data between source and destination are left to particular implementation details of each engine. ADIOS provides a number of engines, which are described below.

2.1.1 File-based Engines

ADIOS provides two types of engines for performing parallel IO of data to disk storage.

BPFile Engine

The BPFile engine is the default engine for storage. The output file target is a directory, which contains both metadata and data files. The number of files is tailored to the capability of the file system, not to the number of writers or readers, which ensures scalable I/O performance. The steps stored in a single file target, can be read by other applications step-by-step simultaneously. Therefore, this engine can be used for in situ processing through the file system.

HDF5 Engine

This engine can be used to write and read HDF5 formatted files. It uses the Parallel HDF5 library, so it provides only a compatibility layer to process HDF5 files in an ADIOS application workflow and is only as scalable as the HDF5 library itself. Streaming access to data is not currently available, but will be available once it is supported by HDF5.

2.1.2 Data Staging Engines

Data staging is a generic concept in ADIOS for providing concurrent access to data to one or more consumers through memory or streamed over the network. It can map onto both time and space division of the taxonomy in Chapter **??**. Data staging engines are typically used for doing in situ analysis and visualization and code coupling. With staging engines, the data producer will write the data using the ADIOS API. The data are then available to be read by the consumers using the ADIOS API. Each staging engine has the ability to move the data in different ways, which are described below.

Scalable Staging Transport (SST)

The most versatile and flexible staging engine uses either RDMA, TCP, UDP, or shared memory to move data from a producer (parallel application) to other consumers (multiple independent parallel applications). Consumers can come and go dynamically without affecting the producer. The output step is buffered in the producer's memory, and readers pull out portions of the buffered data with RDMA operations or communicate with a thread in the producer to receive it via TCP. The

requirement of all engines to always provide a consistent view of a step to a reader may result in blocking the producer from progressing if the consumer is slower than the producer. The SST writer engine aggregates metadata for every step, and shares it with all readers. Readers then issue remote reading operations based on the I/O pattern in metadata. This allows the I/O pattern to vary over time. SST also allows readers to disconnect and reconnect while writers keep writing. To address different application requirements, the SST buffering policy can be configured at run-time. This includes keeping only the most recent step, buffering a fixed window of consecutive steps, or blocking until the step is consumed. In cases where strong coupling is required between applications, the buffer limit can be set to 1, which ensures that every step is consumed by the reader before the producer moves to the next data step. For use cases like interactive visualization, buffering only the latest step is useful since the user typically does not want to block the simulation while a particular time step is explored. While SST aims to provide the flexibility for addressing various application requirements, the fact that it manages metadata for every single step may be overkill for uses cases where the metadata does not change frequently, or at all.

Insitu-MPI

This engine focuses on the speed of data movement for use cases where the metadata is constant across the workflow and a single metadata aggregation at the first data step will suffice. After this first step, each writer and reader knows the exact I/O pattern and direct communication is performed using asynchronous send and receive operations using an MPI communicator. Since it uses MPI, the producer and consumer applications must be launched within a single mpiexec command using the Multiple Program Multiple Data (MPMD) mode. The engine directly sends the application data to the consumer, hence, the producer is synchronized to the consumer at every step to avoid modifying the data before it is received. For very large applications with constant I/O patterns, the Insitu-MPI engine can provide CPU savings for metadata management. However, since it must be launched in MPMD mode under MPI, the flexibility of readers dynamically join or leave is not supported at run-time.

Staging for Strong Coupling (SSC)

The SSC engine is also designed for applications that have constant metadata over time. Similar to the Insitu-MPI engine, the SSC engine aggregates metadata once on the first time step. The main differences between SSC and Insitu-MPI are that SSC uses one-sided MPI communication and that the producer output is buffered. The one sided MPI paradigm does not require the send and receive calls to be paired. Instead, it allows direct access to remote memory of another process. The buffering of application data, on the other hand, enables the producer to continue with the computation while the data is transferred to the consumer. In very large scale coupling use cases this approach saves the overhead of one side waiting for the other

side to complete the send and receive pairs, and makes it possible for applications to very quickly, and frequently exchange data.

DataMan

This engine focuses on providing good bandwidth over wide-area-networks (WAN. It uses the publish and subscribe communication mechanism of the ZeroMQ library and has been optimized specifically for long-distance low-latency data movement. Unlike other staging engines, such as SSC described above, DataMan does not guarantee that every data step is transferred. Instead, the subscriber is designed to read only the latest data steps, while ignoring the previous steps. This saves the two-way communications for checking step completion, which usually means several hundred milliseconds in inter-continental data transfers. Because of this, the data transfer latency is greatly reduced and can support near-real-time analysis better than other engines over the WAN.

2.2 Advanced Data Management Services

ADIOS has a number of internal and external supports for advanced management of data. These include data compression, and schemas for providing additional information about ADIOS data to help downstream processing applications, such as analysis and visualization to properly interpret the raw data.

2.2.1 Data Compression

ADIOS supports operators as a mechanism for performing calculations on the data before it is written by an engine. A general purpose operator, called a Callback provides the user with the ability to perform arbitrary calculations and manipulations to the data inside the engine. Data compression is provided in ADIOS using this mechanism. It provides support for a number of different lossless and lossy compression methods, which are described below.

In the classical workflow for high-performance scientific simulations, the entire data set is written to storage after generation. This will no longer be viable at the exascale, simply because the amount of data will swamp the filesystem. To accelerate scientific discovery, we must prioritize information over data. It will be vital to take advantage of a priori user information to prioritize the most useful data so that I/O can be completed under standard HPC time constraints. (For example, on Summit, jobs are limited to 24 hours.) One solution is data compression. ADIOS supports storing or transporting data in compressed form to reduce the I/O cost while preserving key information, which in turns speed up simulations or in situ data analysis and visualization [5]. Enabling compression requires minimal devel-

opment effort from users. Simply specifying an operator for each variable enables ADIOS to automatically compress or decompress at the point of data publication or subscription. Lossless compressors such as BZip2 [25] preserve every bit of the data, but compression ratios observed in practice are minimal. Lossy compressors such as MGARD [2, 4, 3], SZ [10, 20, 26], and ZFP [21] provide much higher compression ratios (usually more than an order of magnitude than lossless), but information is lost. However, most lossy compressors allow control of the loss through parameters, which can be easily set in ADIOS. Also, as derived quantities in data are particular important for scientific discovery, one of the compressors supported by ADIOS, MGARD, can consider one or more relevant quantities of interest and reduce the data so as to preserve these quantities. Furthermore, ADIOS supports the meta-compressor Blosc which provides further lossless compressors (Zstd, Snappy, BloscLZ, LZ4HC) as well as shuffle pre-conditioners. In GPU-centric applications, using ADIOS with Blosc's threaded-chunked compressor variants regularly trades unutilized CPU-cycles for I/O speedup [15].

2.2.2 Schemas

Schemas provide the ability to annotate the semantics of the array-based layout of data in ADIOS. These provide the meaning of each data array, and the relationship between groups of arrays in an ADIOS file or stream. This capability makes it easier for tools using ADIOS to be used together in, for example, a complex scientific workflow. Two examples of such schemas are described below.

ADIS Visualization Schema

The Adaptable Data Interface for Services (ADIS) is a schema for describing mesh-based data that are used by visualisation tools. ADIS uses a JavaScript Object Notation (JSON) formatted strings to describe the content of ADIOS data. For example, for ADIOS data arrays representing field data on a uniform grid, the ADIS schema will specify that there is a uniform mesh of a given size, and the names of the arrays in the ADIOS stream for each field and the association on the mesh (e.g., zone centered, point centered, etc). For more complex mesh types, like unstructured grids, ADIS specifies the names of the arrays for specifying the relevant mesh structures (e.g., point coordinate values, cell information, etc).

ADIS also supports the creation of data sets from ADIOS in the VTK-m [24] format. Given a schema, and an ADIOS file/stream, ADIS will read data from ADIOS and construct the appropriate VTK-m data object.

openPMD Schema

The Open Standard for Particle-Mesh Data Files (openPMD) is a schema for describing mesh- and particle-based data. Its primary focus is the exchange, archival and replicability of scientific data via a minimal set of conventions and meta information. The schema is defined in the so-called "base standard" and "extensions". The former is agnostic of the data's scientific domain and can be automatically verified/parsed, visualized, scaled and dimensionally analyzed (describing units and quantities). The base standard also provides means to document authorship, hardware and software environments towards reproducible research. Based on this, standardized meta-information in openPMD schema "extensions" add further meaning for domain-scientists, e.g. by documenting algorithms and methods that generated a data set.

Contrary to visualization-focused and domain-specific schemas, openPMD is a notion for *scientifically self-describing* data in general, providing a unified description for data in scientific workflows from source, over processing, analysis and visualization to archival in (open) data repositories. openPMD is widely adopted in plasma-physics, particle-accelerator physics, photon-science, among others.[1]

The schema can be added to data described via hierarchical, self-describing (portable) data formats. Open implementations are available in C++, Python and Fortran and currently range from MPI-parallel ADIOS1, ADIOS2 and HDF5 library backends to serial JSON files. The openPMD schema is versioned, citable and developed on GitHub. Its release is version 1.1.0 and data files using the schema are forward-updatable via lightweight meta-data transformations [14].

2.3 Discussion

The number of engines available in ADIOS provides a large amount of flexibility when selecting a configuration. Further, multiple executables can be connected using the read/write API of an ADIOS engine to support a range of different types of workflows. The example workflow in Figure 2 shows ADIOS (indicated with red arrows) being used as the data movement mechanism for a number of tasks. It is used to couple two simulations, in situ visualization on the HPC resource, in transit visualization on a cluster in the HPC center, and transfer data over the WAN to remote site for analysis. Additionally, data from a sensor/experiment is streamed over the WAN for analysis that uses simulation results.

When designing a visualization workflow, the choice of engine for each component is dependent on a number of factors. Broadly speaking these classes of visualization are post hoc, in situ (time division), and in transit (space division). Post hoc visualization is the traditional mode of visualization where the data are read from disk. As discussed in Chapter **??**, in situ visualization, while a broadly

[1] Curated list available at https://github.com/openPMD/openPMD-projects

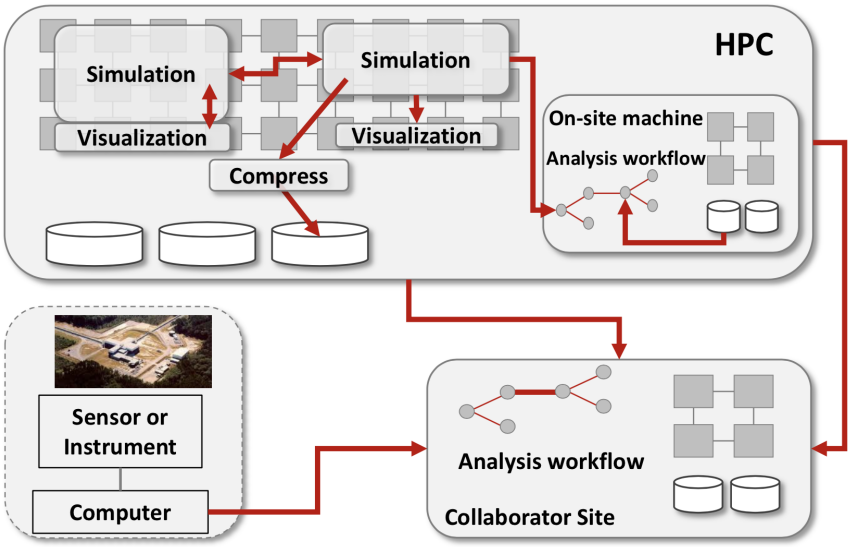

Fig. 2 Example workflow using ADIOS for simulation coupling, in situ visualization, in transit visualization, and streaming of both simulation and experimental data over the WAN to a remote site for analysis.

defined term, is for simplicity, the case where the visualization and simulation use the same set of resources. In transit visualization uses two distinct sets of resources, one dedicated to the simulation and the other dedicated to the visualization. The network is used to transfer data between the two sets of resources. Below, we discuss these three modes from the ADIOS and visualization perspectives and the impact of choices made have on visualization functionality and performance.

2.3.1 ADIOS Perspective

From an ADIOS perspective, the following three characteristics are important: (1) data access and movement, (2) fault tolerance, and (3) programmability.

Data Access and Movement

Data access is defined as how much of the total spatio-temporal data are available, as the temporal range of data that are available. Data movement is the amount of data that must be moved from producer to consumer.

- **Post hoc:** Has access to all the spatio-temporal data that have been saved. However, the data movement cost is highest, and may restrict the amount of data available.

- **In situ / time division:** Has the highest access and lowest movement costs for spatio-temporal data as resources are shared with the producer. Access to multiple temporal steps requires additional on-node resources.
- **In transit / space difision:** Data access is configurable based on needs. The dedicated resource can be sized to control the amount of spatio-temporal access, as well as temporal range. Since data movement occurs over the internal network, it is much faster than I/O.

Fault Tolerance

Fault tolerance describes the relative robustness of the system with respect to faults occurring in either the producer or the consumer.

- **Post hoc:** The consumer is independent from the data producer, so fault tolerance is very high.
- **In situ / time division:** Because resources are shared, the producer and consumer can impact each other. This includes faults, memory corruption, memory usage, etc.
- **In transit / space division:** Like post hoc, the consumer is independent from the data producer. Faults occurring on the dedicated nodes will not impact the producer.

Programmability

Programmability describes the relative ease and flexibility of connecting a simulation with visualization. This includes composing a workflow, connecting components in a workflow, and modifying the underlying data movement mechanism. Since all three classes of visualization use the same abstraction, the programmability is improved by simply changing the engine used.

Table 1 provides a visual representation of the relative strengths of each ADIOS engine with respect the characteristics described above. A score for each engine is assigned based on how well the engine performs with respect to each characteristic described above. A " + " signifies a favorable evaluation, " − " a less favorable evaluation, and "0" for in between.

2.3.2 Visualization Perspective

From a visualization perspective, a different set of characteristics are important (see for example, [17]). We discuss the following characteristics below: (1) scalability and resource requirements, (2) interactivity, (3) fault tolerance, and (4) programmability.

Scalability and Resource Requirements

Scalability is defined as how efficiently the visualization task can use the allocated resources. Resource requirements is defined as the need for additional resources beyond that of the simulation.

- **Post hoc:** Has the flexibility to allocate resources suitable for the required tasks, however I/O can slow for large data.
- **In situ / time division:** Since the visualization must run at the scale of the visualization, the performance will depend on the operation. Communication heavy algorithms could suffer poor performance at larger scales.
- **In transit / space division:** Has the flexibility to allocate resources suitable for the required tasks. Since I/O is avoided, access to data can be much faster.

Interactivity

Interactivity is defined as the ability for a user to interact with the data, select regions of interest, and plot the data to extract understanding.

- **Post hoc:** Because visualization is independent from the simulation, full interactivity is possible with all data available.
- **In situ / time division:** Visualization has full access to all of the data that are available on the simulation resources. Due to limited available resources, the temporal range of data could be limited. If the data are shared, the simulation could be blocked while visualization occurs.
- **In transit / space division:** Visualization has full access to all of the spatio-temporal data that are moved to the dedicated resources. Because the data are not shared with the simulation, blocking can be avoided.

Table 1 Characterization of each ADIOS engine

		Engine Type					
	ADIOS Characteristics	**BPFile**	**HDF5**	**SST**	**Insitu-MPI**	**SSC**	**DataMan**
Data Access & Data Movement	*Spatial Access*	+	+	0	0	0	0
	Temporal Fidelity	-	-	+	+	+	0
	Temporal Range	+	+	0	0	0	0
	Movement	-	-	0	0	+	0
	Fault Tolerance	+	+	+	+	0	+
	Programability	+	0	+	+	+	+

Fault Tolerance

As above, fault tolerance refers to the robustness of the visualization to avoid impacting the simulation.

- **Post hoc:** Visualization is independent from the simulation, so fault tolerance is very high.
- **In situ / time division:** Because resources are shared, it is possible for the visualization task to negatively impact the simulation.
- **In transit / space division:** Like post hoc, the visualization is independent from the simulation. Errors occurring on the dedicated nodes will not impact the simulation.

Programmability

As above, programmability describes the ease of using visualization tools with simulation data in a variety of configurations. This includes performing visualization tasks within a workflow, connecting analysis and visualization tasks together, and the ability to access data from different sources. Since all three classes of visualization use the same abstraction, the programmability is improved by simply changing the engine used.

Table 2 provides a visual representation of the relative strengths of each ADIOS engine with respect the important visualization characteristics described above. As above, a " + " signifies a favorable evaluation, " – " a less favorable evaluation, and "0" for in between.

Table 2 Characterization of each ADIOS engine for visualization

		Engine Type					
	Visualization Characteristics	**BPFile**	**HDF5**	**SST**	**Insitu-MPI**	**SSC**	**DataMan**
Scalability	*Data*	-	-	+	+	+	-
	Communication	+	+	+	-	-	+
	Resource	+	+	-	+	+	-
Interactivity	*Spatial*	-	-	0	+	+	0
	Temporal	-	-	0	+	+	0
	Temporal Range	+	+	0	-	-	0
	Block Simulation	+	+	+	-	-	+
	Fault Tolerance	+	+	+	+	0	+
	Programability	+	0	+	+	+	+

2.3.3 In Situ Data Placement and the Associated Performance Implications

Placement (in-line, in-transit, hybrid methods) is an important aspect to consider when planning for the use of in situ techniques. Performance can vary drastically depending on what analysis operations are used, how often they are performed, and at what scale they are performed. This performance difference is due primarily to the scaling characteristics of the analysis algorithms in relation to that of the underlying simulation, and can have a large effect on the overall cost of a simulation plus its visualization and analysis components.

In a work by Kress et al. [18] they look specifically at the cost of performing isocontours and ray tracing with parallel compositing both in-line and in-transit, and observe large cost variations based on placement as the simulation was scaled. Their work found that as the simulation was scaled to 16K cores, that visualization algorithms suffered large slowdowns in-line. However, if the data was transferred from the simulation over the network to a set of dedicated visualization nodes that the visualization routines completed much faster. They bring up a couple of general guidelines in that work: (1) if fastest time to solution is your goal at scale, moving the data and performing the visualization in-transit is the best solution; (2) if the lowest total combined cost of the simulation and visualization routines are the overall goal, the solution becomes more complicated. In general though, as the simulation scales if the visualization routines do not scale as well, moving the visualization routine to a smaller in-transit allocation is the best choice. However, careful consideration has to be given to how large of an in-transit allocation to reserve, and how often the visualization should be performed. A follow on work [19] develops a cost model to evaluate the use of in situ methods at scale.

2.4 Code Examples

This section contains examples using ADIOS to read and write data. The first example, shown in Listing 1, illustrates how a simulation code would write output data to disk using the BPFile engine. The engine type is specified in line 11.

Listing 1 Example of simulation writing outputs to a file.

```
1  adios2::ADIOS adios(MPI_COMM_WORLD);
2  // Declare named IO process
3  adios2::IO io = adios.DeclareIO("output");
4
5  // Declare output type and size.
6  adios2::Variable<double> var =
7    io.DefineVariable<double>("var", globalDims, offset, localDims);
8
9
10 // Set engine output to BPFile
11 io.SetEngine("BPFile");
12 adios2::Engine engine = io.Open("output.bp", adios2::Mode::Write);
13
14 // Run Simulation
15 for(...)
16 {
17    double *data = Simulation();
18
19    engine.BeginStep();
20    engine.Put(var, data);
21    engine.EndStep();
22 }
23 engine.Close();
```

Listing 2 shows a program that reads data from the output file and performs visualization on the data.

Listing 2 Example of a visualization program reading data from a file

```
1  adios2::ADIOS adios(MPI_COMM_WORLD);
2  adios2::IO io = adios.DeclareIO("input"); //Declare named IO process
3  io.SetEngine("BPFile");
4  adios2::Engine reader = io.Open("output.bp", adios2::Mode::Read);
5
6  std::vector<double> data;
7
8  while (reader.BeginStep(adios2::StepMode::Read) == adios2::StepStatus::OK)
9  {
10   adios2::Variable<double> var = reader.InquireVariable<double>("var");
11   if(var)
12     reader.Get<double>(var, data);
13   reader.EndStep();
14
15   Visualize(data);
16 }
17 engine.Close();
```

To change the simulation output mode from file based to the SST in situ mode, the only change required in Listings 1 and 2, is to change lines 11 and 3, respectively, from

```
io.SetEngine("BPFile");
```

to

```
io.SetEngine("SST");
```

The visualization program will now read the outputs produced by the simulation from the ADIOS stream named "output.bp", which in this case, will be coming from the SST engine in the simulation writer process. All of the engine types support by ADIOS can be changed in this way.

An alternative to specifying engine type in the source code is to use a configuration file, which is parsed at runtime and specifies the engines type and IO processes to be used. Both XML and YAML are supported as configuration file formats. the only change required in Listings 1 and 2, is to change line 1 from

```
adios2::ADIOS adios(MPI_COMM_WORLD);
```

to

```
adios2::ADIOS adios("config.xml", MPI_COMM_WORLD);
```

This allows the underlying data movement mechanism to be changed without re-compiling anything.

Listing 3 Configuration file, "config.xml" for examples shown in Listing 1 and 2

```
1
2  <!-- adios2 config file in XML format -->
3  <?xml version="1.0"?>
4  <adios-config>
5      <io name="output">
6          <!-- engine type can be set at runtime: BPFile, SST, etc. -->
7          <engine type="BPFile">
8          </engine>
9      </io>
10     <io name="input">
11         <engine type="BPFile" />
12     </io>
13 </adios-config>
```

Listing 4 Alternative configuration file, "config.yaml" for examples shown in Listing 1 and 2

```
1  ---
2  # adios2 config file in YAML format
3
4  - IO: "output"
5    Engine:
6      # engine type can be set at runtime: BPFile, SST, etc.
7      Type: "BPFile"
8
9  - IO: "input"
10   Engine:
11     Type: "BPFile"
```

Basic XML and YAML configuration files for ADIOS are shown in Listings 3 and 4 respectively. Changing the "type" field on line 7 from "BPFile" to "SST" will configure ADIOS to use the SST engine when the executables are run.

Listing 5 illustrates how to read ADIOS data using the Python high-level API. ADIOS provides a “pythonic” interface of an iterable container of steps using a generic “read” function that always return a numpy array for easy integration with the Python data analysis ecosystem. Similarly, switching between Engines is done through a parameter in the open function or using a config file as described above.

Listing 5 Python High-Level API Read Example

```
import adios2

with adios2.open("euler.bp", "r", engine_type="BPFile") as fh:

    for fstep in fh:

        # retrieve current step
        step = fstep.current_step()

        # inspect variables dictionary in current step
        step_vars = fstep.available_variables()
        for name, info in step_vars.items():
            print("variable_name: " + name)
            for key, value in info.items():
                print("\t" + key + ": " + value)
            print("\n")

        if( step == 0 ):
            size_in = fh_step.read("size")

        # read variables in current step
        # returning a numpy array for easy integration
        # with data science frameworks (e.g. pandas, scipy)
        T = fstep.read("T")
```

3 Example Use Cases

In this section we demonstrate how the I/O abstraction and engines described above in Section 2 can be used with different applications. These examples show how ADIOS engines can be used in different ways to accomplish the in situ processing required by a scientific campaign. The examples show how in situ can be used in both shared and separate resource configurations, and also include an example where data was streamed across the wide area network (WAN). In each example, we motivate the purpose of each scientific example and how ADIOS was used to provide the solution.

3.1 Strong Coupling in a Fusion Simulation

High-Fidelity Whole Device Modeling (WDM) of Magnetically Confined Fusion Plasmas is among the most computationally demanding and scientifically challenging simulation projects. The ten-year goal is to have a complete and comprehensive application that will include all the important physics components required to simulate a full toroidal discharge in a tokamak fusion reactor. The main driver is based on the strong coupling of two advanced and highly scalable gyrokinetic codes, XGC and GENE, where the former is a particle-in-cell code optimized for the treating the edge plasma while the other is a continuum code optimized for the core. Applications

for additional physics are intended to be coupled in the future, e.g. ones for material wall interactions or for high energy particles.

In the WDM workflow, the ADIOS BPFile engine is used to save checkpoint/restart files, offloads variables for in situ analysis and visualization [8]. For in-memory data exchange, the SST and Insitu-MPI engines are used for coupling of the core and edge simulations [11]. To date, three-dimensional field information has been shared between XGC and GENE, but a five-dimensional distribution function-based coupling is now under development. Published results [8, 11] have all relied on synchronous exchange, but asynchronicity will need to be explored in order to mitigate blocking while data are not available. The ADIOS SSC engine is designed to support the asynchronous WDM coupling workflow. ADIOS affords both performant scalability as data sizes grow with increased dimensionality, as well as APIs that support asynchronous operation.

3.2 Streaming Experimental Data

Fusion experiments provide critical information to validate and refine simulations that model complex physical processes in the fusion reactor as well as to test and validate hypotheses. Recent advances in sensors and imaging systems, such as sub-microsecond data acquisition capabilities and extremely fast 2D/3D imaging, allow researchers to capture very large volumes of data at high rates for monitoring and diagnostic purposes as well as post-experiment analyses. For example, JET, the world's largest magnetic confinement plasma physics experiment in the UK, is producing about 60 GB of diagnostic data per pulse [12]. A 2-D spatial imaging system, called Electron Cyclotron Emission Imaging (ECEI), in KSTAR, Korea, alone can capture 10 GB of image data per 10 second shot [28].

A system using ADIOS has been developed for KSTAR to support various data challenges by executing remote experimental data processing workflows in fusion

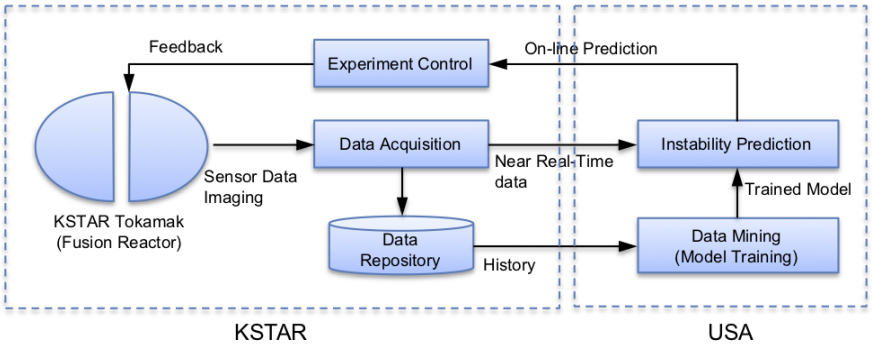

Fig. 3 Fusion instability monitoring and mitigation workflow.

science. It is one of the drivers for the development of the DataMan engine to support science workflows execution over the wide-area network (WAN) for near-real-time (NRT) streaming of experiment data to and from an experiment site and remote computing resource facilities.

An example of KSTAR workflow is shown in Figure 3. This workflow is a multi-level workflow in that each box consists of one or more sub-workflows, each of which can be composed with ADIOS engines. One of the main goals is to stream online fusion experiment data from KSTAR in Korea to a computing facility in USA in order to perform various computational intensive analyses, such as instability prediction and disruption simulation. While our previous effort [9] focused on building remote workflows with data indexing, we are currently working on composing the KSTAR workflow with DataMan. In this workflow, we use ADIOS' DataMan engine to move raw observational data as streams from Korea to the USA. Once data streams arrived in a USA computing facility, we launch a set of analysis and visualization workflows to perform denoising, segmentation, feature detection, and selection for detecting any instabilities. Visualization results can be delivered back to Korea for designing the next upcoming shots. In short, ADIOS engines enable researchers to compose and execute workflows spanning local resources and remote large-scale high performance computing facilities for NRT analysis and decision-making.

3.3 Interactive In Transit Visualization

The Japan Aerospace Exploration Agency (JAXA) has implemented various ways for visualizing one of their CFD simulations, upacs-mc-LES. The visualization of CFD data consists of both batch and interactive visualization. Batch visualization is performed to create preset view images of the flowfield. Interactive visualization is conducted by interactively using Visit to understand the physics of the flowfield. While interactive visualization is not performed all the time during simulation, it is essential to have the capability to launch and attach the visualization process to the simulation when necessary, then to seamlessly detach when finished.

The agency has a heterogeneous HPC system, the Supercomputer System Generation 2 (JSS2). The main computer is a Fujitsu supercomputer with FX100 CPUs specialized for vector computations. Another cluster with x86 processors and GPUs is available for visualization and GPU-based analysis. There is a shared Lustre file system, which can be used for post-processing. An Ethernet and InfiniBand network connects the two machines, but only a portion of the nodes can communicate between the two machines. Most of the nodes can only communicate with other nodes on the internal network.

Batch visualization in post-processing is an easy way to produce movies of preset 3D visualizations on the GPU cluster, but it is stressing the file system and cannot support the largest simulations due to the I/O overhead. In situ visualization based on LibSim [27] is another approach, where the main computer is used to produce the images within the simulation code. In situ not only allows for producing

a movie without dumping all data to disk but it also allows for interactive data exploration. ADIOS makes another approach feasible: in transit visualization where the simulation data is streamed from the main computer to another application, which in turn uses LibSim to create the visualizations. The visualization can be performed either on the main computer or on the GPU cluster (see Fig. 4). In all cases, Visit is used as the GUI for attaching to the visualization in case the user wants to interactively explore the data set.

The main drawback of in situ visualization with LibSim, for interactive exploration, is that the simulation process stops during interactive visualization. JAXA users want the simulation to progress with the computation while they are looking at a snapshot in time. In transit visualization using the ADIOS SST engine solves that problem and is as easy to use as in situ visualization when launching them as two separate applications together on the main computer in a single job.

Another advantage of in transit visualization (both for batch and interactive visualization) is that the simulation is not affected by the visualization process in terms of computing performance, nor by abnormal termination of the visualization process. The simulation progresses independently from the visualization and therefore the cost of visualization is amortized. On the other hand, data movement also has a cost and this offsets some of the advantages. As discussed in Section 2.3.3, there are trade-offs between the in situ and in transit approaches, and it depends on the simulation size, data size and visualization cost in order to determine which approach works best. Therefore, JAXA wants to maintain and provide all approaches to visualization for its users.

In transit visualization also provides the capability to use the GPU cluster for the visualization. The main difficulty with using a separate machine is that two jobs need to be submitted to two different machines and run at the same time. Current job scheduling policies only support batch processing. Therefore, the only way to

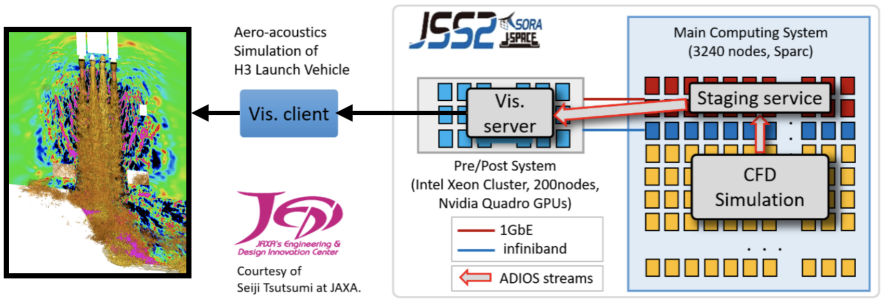

Fig. 4 Two steps of staging of data necessary on the JAXA heterogeneous system for interactive visualization. Simulation data is staged to a concurrent staging service on nodes that have network connections to the GPU cluster. The data is further staged to the visualization server running on the GPU cluster. The visualization client then visualizes the data. The visualization on the left shows acoustic waves on the cross section and exhaust jet are visualized by normalized pressure fluctuation and iso-surface of temperature, respectively.

do interactive visualization on the GPU cluster is to submit an interactive job once the simulation is running. This is fine for interactive visualization where the user is present. Although ADIOS makes it possible to run the visualization application immediately and let it wait for the connection to the simulation indefinitely, for a batch visualization of an overnight job, this is still a waste of resources (on the GPU cluster).

Lastly, note that using ADIOS in the simulation to output the data, the target for the data can be a concurrent application for batch visualization on the main computer, or an application on the GPU cluster for interactive/batch visualization, or it can be the Lustre file system for storing data for post-processing. The visualization application is also the same for all the three cases. It is only a matter of the runtime setup and the choice of the ADIOS Engine to run any of these cases.

4 Conclusion

ADIOS was designed from the observation that the API describing traditional I/O to the file system could be abstracted to describe more complicated data movement. Since applications almost always read and/or write data to storage it becomes straightforward to replace the traditional I/O mechanism with an abstraction layer that supports much more complex movement of data with minimal changes to the flow of execution.

In this chapter we have described the high level design of the ADIOS library as well as a description of the currently available engines. We also provided a comparative discussion on each engine and discussed their strengths, weaknesses, and where each is most suitable. To provide some concrete examples of how ADIOS has been used in practice, we described a number of experimental and simulation examples that use ADIOS in their workflow for in situ processing and visualization.

5 Acknowledgements

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

References

1. J. Ahrens, B. Geveci, and C. Law. Visualization in the paraview framework. In C. Hansen and C. Johnson, editors, *The Visualization Handbook*, pages 162–170, 2005.

2. M. Ainsworth, O. Tugluk, B. Whitney, and S. Klasky. Multilevel techniques for compression and reduction of scientific data—the univariate case. *Computing and Visualization in Science*, 19(5-6):65–76, 2018.
3. M. Ainsworth, O. Tugluk, B. Whitney, and S. Klasky. Multilevel techniques for compression and reduction of scientific data—the multivariate case. *SIAM Journal on Scientific Computing*, 41(2):A1278–A1303, 2019.
4. M. Ainsworth, O. Tugluk, B. Whitney, and S. Klasky. Multilevel techniques for compression and reduction of scientific data-quantitative control of accuracy in derived quantities. *SIAM Journal on Scientific Computing*, 41(4):A2146–A2171, 2019.
5. J. Chen, D. Pugmire, M. Wolf, N. Thompson, J. Logan, K. Mehta, L. Wan, J. Y. Choi, B. Whitney, and S. Klasky. Understanding performance-quality trade-offs in scientific visualization workflows with lossy compression.
6. H. Childs, E. Brugger, B. Whitlock, J. Meredith, S. Ahern, D. Pugmire, K. Biagas, M. Miller, C. Harrison, G. H. Weber, H. Krishnan, T. Fogal, A. Sanderson, C. Garth, E. W. Bethel, D. Camp, O. Rübel, M. Durant, J. M. Favre, and P. Navrátil. VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data. In *High Performance Visualization–Enabling Extreme-Scale Scientific Insight*, pages 357–372. Oct 2012.
7. H. Childs, D. Pugmire, S. Ahern, B. Whitlock, M. Howison, Prabhat, G. H. Weber, and E. W. Bethel. Extreme scaling of production visualization software on diverse architectures. *IEEE Computer Graphics and Applications*, 30(3):22–31, 2010.
8. J. Y. Choi, C. Chang, J. Dominski, S. Klasky, G. Merlo, E. Suchyta, M. Ainsworth, B. Allen, F. Cappello, M. Churchill, P. Davis, S. Di, G. Eisenhauer, S. Ethier, I. Foster, B. Geveci, H. Guo, K. Huck, F. Jenko, M. Kim, J. Kress, S. Ku, Q. Liu, J. Logan, A. Malony, K. Mehta, K. Moreland, T. Munson, M. Parashar, T. Peterka, N. Podhorszki, D. Pugmire, O. Tugluk, R. Wang, B. Whitney, M. Wolf, and C. Wood. Coupling exascale multiphysics applications: Methods and lessons learned. In *2018 IEEE 14th International Conference on e-Science (e-Science)*, pages 442–452, Oct 2018.
9. J. Y. Choi, K. Wu, J. C. Wu, A. Sim, Q. G. Liu, M. Wolf, C. Chang, and S. Klasky. Icee: Wide-area in transit data processing framework for near real-time scientific applications. In *4th SC Workshop on Petascale (Big) Data Analytics: Challenges and Opportunities in conjunction with SC13*, volume 11, 2013.
10. S. Di and F. Cappello. Fast error-bounded lossy hpc data compression with sz. In *2016 ieee international parallel and distributed processing symposium (ipdps)*, pages 730–739. IEEE, 2016.
11. J. Dominski, S. Ku, C.-S. Chang, J. Choi, E. Suchyta, S. Parker, S. Klasky, and A. Bhattacharjee. A tight-coupling scheme sharing minimum information across a spatial interface between gyrokinetic turbulence codes. *Physics of Plasmas*, 25(7):072308, 2018.
12. J. Farthing, T. Budd, A. Capel, N. Cook, A. Edwards, R. Felton, F. Griph, and E. Jones. Data management at jet with a look forward to iter. In *International Conference on Accelerator and Large Experimental Physics Control Systems*, 2006.
13. W. F. Godoy, N. Podhorszki, R. Wang, C. Atkins, G. Eisenhauer, J. Gu, P. Davis, J. Choi, K. Germaschewski, K. Huck, A. Huebl, M. Kim, J. Kress, T. Kurc, Q. Liu, J. Logan, K. Mehta, G. Ostrouchov, M. Parashar, F. Poeschel, D. Pugmire, E. Suchyta, K. Takahashi, N. Thompson, S. Tsutsumi, L. Wan, M. Wolf, K. Wu, and S. Klasky. ADIOS 2: The adaptable input output system. a framework for high-performance data management. *SoftwareX*, 12:100561, July 2020.
14. A. Huebl, R. Lehe, J.-L. Vay, D. P. Grote, I. F. Sbalzarini, S. Kuschel, D. Sagan, F. Pérez, F. Koller, and M. Bussmann. openPMD: A meta data standard for particle and mesh based data.
15. A. Huebl, R. Widera, F. Schmitt, A. Matthes, N. Podhorszki, J. Y. Choi, S. Klasky, and M. Bussmann. On the scalability of data reduction techniques in current and upcoming HPC systems from an application perspective. *Lect. Notes Comput. Sci.*, 10524(4):15–29, 2017.
16. S. A. Klasky, M. D. Wolf, M. Ainsworth, C. Atkins, J. Y. Choi, G. Eisenhauer, B. Geveci, W. F. Godoy, M. B. Kim, J. M. Kress, T. M. Kurc, Q. G. Liu, J. S. Logan, A. B. Maccabe,

K. V. Mehta, G. Ostrouchov, M. Parashar, N. Podhorszki, D. Pugmire, E. D. Suchyta, L. Wan, and R. Wang. A view from ornl: Scientific data research opportunities in the big data age.
17. J. Kress et al. Loosely coupled in situ visualization: A perspective on why it's here to stay. In *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*, ISAV2015, pages 1–6, New York, NY, USA, 2015. ACM.
18. J. Kress, M. Larsen, J. Choi, M. Kim, M. Wolf, N. Podhorszki, S. Klasky, H. Childs, and D. Pugmire. Comparing the efficiency of in situ visualization paradigms at scale. In *International Conference on High Performance Computing*, pages 99–117. Springer, 2019.
19. J. Kress, M. Larsen, J. Choi, M. Kim, M. Wolf, N. Podhorszki, S. Klasky, H. Childs, and D. Pugmire. Opportunities for cost savings with in-transit visualization. In *ISC High Performance 2020*. ISC, 2020.
20. X. Liang, S. Di, D. Tao, Z. Chen, and F. Cappello. An efficient transformation scheme for lossy data compression with point-wise relative error bound. In *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 179–189. IEEE, 2018.
21. P. Lindstrom. Fixed-rate compressed floating-point arrays. *IEEE transactions on visualization and computer graphics*, 20(12):2674–2683, 2014.
22. Q. Liu, J. Logan, Y. Tian, H. Abbasi, N. Podhorszki, J. Y. Choi, S. Klasky, R. Tchoua, J. Lofstead, R. Oldfield, et al. Hello ADIOS: the challenges and lessons of developing leadership class I/O frameworks. *Concurrency and Computation: Practice and Experience*, 26(7):1453–1473, 2014.
23. J. Logan, M. Ainsworth, C. Atkins, J. Chen, J. Y. Choi, J. Gu, J. M. Kress, G. Eisenhauer, B. Geveci, W. Godoy, M. B. Kim, T. Kurc, Q. Liu, K. V. Mehta, G. Ostrouchov, N. Podhorszki, D. Pugmire, E. D. Suchyta, N. Thompson, O. Tugluk, L. Wan, R. Wang, B. Whitney, M. D. Wolf, K. Wu, and S. A. Klasky. Extending the publish/subscribe abstraction for high-performance i/o and data management at extreme scale. *Bulletin of the IEEE Technical Committee on Data Engineering*, 43(1), 3 2020.
24. K. Moreland, C. Sewell, W. Usher, L.-T. Lo, J. Meredith, D. Pugmire, J. Kress, H. Schroots, K.-L. Ma, H. Childs, M. Larsen, C.-M. Chen, R. Maynard, and B. Maynard. Vtk-m: Accelerating the visualization toolkit for massively threaded architectures. *IEEE Computer Graphics and Applications*, 36:48–58, 05 2016.
25. J. Seward. bzip2 and libbzip2. *avaliable at http://www. bzip. org*, 1996.
26. D. Tao, S. Di, Z. Chen, and F. Cappello. Significantly improving lossy compression for scientific data sets based on multidimensional prediction and error-controlled quantization. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1129–1139. IEEE, 2017.
27. B. Whitlock, J. M. Favre, and J. S. Meredith. Parallel In Situ Coupling of Simulation with a Fully Featured Visualization System. In T. Kuhlen, R. Pajarola, and K. Zhou, editors, *Eurographics Symposium on Parallel Graphics and Visualization*. The Eurographics Association, 2011.
28. G. Yun, W. Lee, M. Choi, J. Kim, H. Park, C. Domier, B. Tobias, T. Liang, X. Kong, N. Luhmann Jr, et al. Development of kstar ece imaging system for measurement of temperature fluctuations and edge density fluctuations. *Review of Scientific Instruments*, 81(10):10D930, 2010.