2018 IEEE 14th International Conference on e-Science

Coupling Exascale Multiphysics Applications: Methods and Lessons Learned

Jong Youl Choi[∥], Choong-Seock Chang[**], Julien Dominski[**], Scott Klasky[∥], Gabriele Merlo[xiii], Eric Suchyta[∥],
Mark Ainsworth[†], Bryce Allen[*], Franck Cappello[*], Michael Churchill[**],
Philip Davis[††], Sheng Di[*], Greg Eisenhauer[‡], Stephane Ethier[**],
Ian Foster[*][xiv] Berk Geveci[§], Hanqi Guo[*], Kevin Huck[xi], Frank Jenko[xiii], Mark Kim[∥],
James Kress[∥], Seung-Hoe Ku[**], Qing Liu[¶], Jeremy Logan[∥][xii], Allen Malony[xi],
Kshitij Mehta[∥], Kenneth Moreland[x], Todd Munson[*], Manish Parashar[††], Tom Peterka[*],
Norbert Podhorszki[∥], Dave Pugmire[∥], Ozan Tugluk[†], Ruonan Wang[∥], Ben Whitney[†], Matthew Wolf[∥], Chad Wood[xi]

[*]Argonne National Lab., Lemont, IL 60439, USA
[†]Brown University, Providence, RI 02912, USA
[‡]Georgia Institute of Technology, Atlanta, GA 30332, USA
[§]Kitware, Clifton Park, NY 21065, USA
[¶]New Jersey Institute of Technology, Newark, NJ 07102, USA
[∥]Oak Ridge National Lab., Oak Ridge, TN 37831, USA
[**]Princeton Plasma Physics Lab., Princeton, NJ 08536, USA
[††]Rutgers University, New Brunswick, NJ 08901, USA
[x]Sandia National Lab., Albuquerque, NM 87185, USA
[xiv]Univ. of Chicago, Chicago, IL 60637, USA
[xi]Univ. of Oregon, Eugene, OR 97403, USA
[xii]Univ. of Tennessee, Knoxville, TN 37996, USA
[xiii]Univ. of Texas, Austin, TX 78712, USA

Abstract—**With the growing computational complexity of science and the complexity of new and emerging hardware, it is time to re-evaluate the traditional monolithic design of computational codes. One new paradigm is constructing larger scientific computational experiments from the coupling of multiple individual scientific applications, each targeting their own physics, characteristic lengths, and/or scales. We present a framework constructed by leveraging capabilities such as in-memory communications, workflow scheduling on HPC resources, and continuous performance monitoring. This code coupling capability is demonstrated by a fusion science scenario, where differences between the plasma at the edges and at the core of a device have different physical descriptions. This infrastructure not only enables the coupling of the physics components, but it also connects in situ or online analysis, compression, and visualization that accelerate the time between a run and the analysis of the science content. Results from runs on Titan and Cori are presented as a demonstration.**

I. Introduction

Code coupling is widely used in multi-scale and multi-physics studies. Code coupling, also known as coupled simulation or co-simulation, is a technique in the area of scientific computing in which multiple models (or applications) are run concurrently, working together in an orchestrated fashion to create a unified result. Typically, the applications exchange data with one another multiple times as the run progresses. The complexity of coupled applications can vary. Each application can be as simple as a one-off application (e.g., plotting, diagnosis, etc.) or as complex as a large-scale multi-process parallel simulation running on a supercomputer in which data needs to be exchanged concurrently with many different applications. Organizing the logistics of such complex data flows between multiple parallel applications has been a challenging task for users.

In supporting these coupled workflows, our focus is on running code coupling in large-scale high performance computing environments, such as DOE leadership computing facilities (LCFs) like Titan in ORNL, Cori in NERSC, and Theta in ANL. Running coupled simulation is non-trivial on those platforms.

Going forward, the complexities of dealing with the novel hardware and deep memory and storage hierarchies for exascale and post-Moore's-Law computing mean that it is even more important to think about ways to deal with the complexity of the software development process. If the code coupling paradigm supports flexible and high-performance connections between independently developed executables, then it allows for a larger team of specialists to collaborate without everyone working within a single mammoth code base. This goal, if achieved, would have a broad impact across in situ analysis and reduction, online analysis, and the construction of large coupled-physics science codes.

The ability to run multiple applications cooperatively can be accomplished in a variety of ways. MPI is the de facto standard in high performance computing for running multi-processes/multi-programs. However, enabling communications between different applications outside of the MPI global communicator (`MPI_COMM_WORLD`) needs extra steps; users need to modify existing code to avoid the use of global communicators (e.g., by using `MPI_Comm_split`). If users want to run an application out of the box which depends on the use of a global communicator, MPI will be unusable

978-1-5386-9156-4/18/$31.00 ©2018 IEEE
DOI 10.1109/eScience.2018.00133

for code coupling. Extra services, libraries, or extensions like MPI_COMM_spawn may be required to manage such intra-communications, such as workflow systems, database services, or pub-sub systems.

> Programming coupled systems is hard enough at current scales. With increases in hardware and code complexity, the software development cycle gets even harder. The community needs tools and practical examples that provide a separation of concerns between accuracy, correctness, and performance of the coupling framework.

Our framework provides a state-of-the-art experience for developing and executing large-scale coupled simulations so as to directly address this requirement. One highlight of the framework is relative ease of use. After users have achieved correctness with basic file-based I/O to couple, we aim at requiring virtually zero effort for users to convert their programs to the performant, low-latency, in-memory coupling workflow.

Besides the changes in data flows, running coupled workflows requires some extra services. For example, users need to monitor progress and capture performance information to identify bottlenecks. Data compression may be needed to help move data around the system efficiently while managing accuracy. On-line data reduction and visualization mechanisms are required to extract contents of data while the simulation is running. It is critical to provide these services without significantly increasing users' management burden.

Our framework provides an easy-to-use integrated environment with extra services users can use: performance monitoring, status updates, and online data analysis. Specifically, we build our system upon a set of existing technologies developed to address the complexities of achieving high performance for modern HPC environments. Leveraging the Adaptable I/O System (ADIOS), the Savanna/Cheetah runtime from the Exascale Computing Project (ECP), the MGARD and SZ compression libraries from ECP, the VTK-m visualization toolkit from the DOE Advanced Scientific Computing Research program and ECP, and the TAU performance monitoring system, this state-of-the-art code coupling system and its demonstration with the Whole Device Modeling fusion application (WDM) showcases a way forward to address a much wider set of future scientific code development opportunities. By focusing on this particular science application that is already a target for scaling to the future exascale systems, we hope to demonstrate the programmability and reusability of this approach. We have highlighted in each section below the capabilities and practical aspects of the coupling system's components that can be used more broadly. This demonstration system has allowed us to address the software engineering challenges of working with a diverse set of collaborators and specialists, using well-defined interfaces and efficient code coupling technologies to piece together a larger and more comprehensive computational experiment.

Many scientific workflow systems, such as Parsl [1], Kepler [2], Pegasus [3], Swift [4], and Taverna [5], include direct support for launching multiple applications. However, in general, these workflow systems do not facilitate communication between those applications, which is critical in code coupling to share and synchronize data as the applications progress. Another available mechanism for coupling multiple applications is to use a messaging system, such as pub-sub systems, ActiveMQ, ZeroMQ, etc. These systems are tuned for increasing network performance. Our solution allows both file and network I/O, enabling the seamless transition between disk-based I/O and memory-/network-supported I/O for the best software engineering experience. Users can start development with file I/O for debugging and verification and switch to a code coupling workflow instantaneously when they need to perform large-scale multi-scale/multi-physics studies. In addition, our framework provides an integrated environment to optimally utilize available resources in HPC environments and to perform other data related operations such as compression and analysis.

In this paper, we present techniques for supporting science use cases involving coupled simulations, and share our experiences in the development of needed tools and the integration of those tools for use for scientists. In summary, the paper makes the following contributions:

- Methods for the easy coupling of different executables (services) along with plug-ins that provide state-of-the-art I/O techniques. Especially, supporting transparent switching of data flows from file-based I/O, including NVMe and burst buffers, to memory-based data exchange, such as RDMA.
- The ability to allow for both large-scale, data-intensive computations running at scale and large-scale collaborative experiments/observations.
- The ability to compose and execute end-to-end, large-scale analysis for near-real-time feedback using ADIOS services.
- The capability to transparently reduce the amount of data output from a simulation and check its accuracy.
- The ability to compose multiple services on a HPC resource and place the right processing at the right place.
- The ability to provide performance feedback, so we can understand the cost of the LCF simulations.

The technical achievements described are driven by both the computational fusion science requirements and the changing nature of the HPC software development process. A preview of how these technologies interoperate can be seen in the desktop shot in Fig. 1. In order to fully describe the set of innovations, we start by discussing the motivating challenges of simulating a complete fusion Tokamak device in § II. Then we present more technical depth on the components of the code coupling infrastructure in § III, including discussions of lessons learned in § IV. Finally, we conclude in § V with a look toward the next steps for the fusion application and for broader impact on computational science at scale.

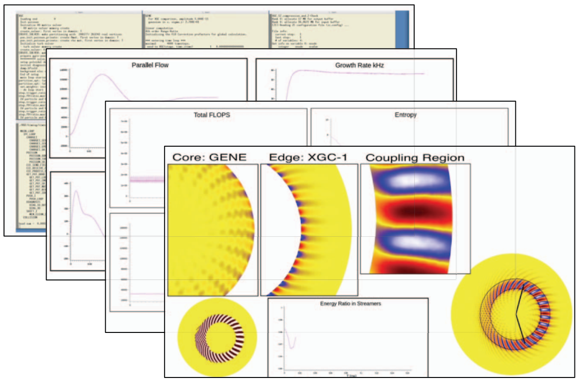


Fig. 1. A dashboard style visualization of the online information produced during a representative run, including the raw outputs from applications, physics diagnostics, performance and data compression monitoring, and physics data visualization.

II. Building a Modern Fusion Simulation Environment

In this section, we consider code coupling applied to cutting-edge fusion science simulations. Examples of this kind are the primary motivation for the work presented in this paper. Such simulations have diverse requirements on both the domain science and computer science sides. In § II-A, we contextualize the fusion science application, then provide further details about the physics applications in § II-B. Finally, in § II-C, we present the concrete set of coupled fusion simulations we have worked to support.

A. *Using Code Coupling to Support Whole Device Modeling*

For decades, scientists and engineers have been striving to use tokamak reactors to harness fusion energy. Tokamaks use strong magnetic fields to confine plasmas, creating conditions where temperature and density are high enough for nuclei to fuse and release energy. ITER[1] will be the largest fusion device ever built, with the goal of demonstrating net power output from a fusion reactor for the first time. However, for ITER or other future tokamaks to be successful, there is a growing urgency for a framework to predict with high fidelity the fusion performance of such machines. These simulations guide how tokamaks should be designed and operated to maintain energy confinement and maximize the life of the machine. Unfortunately, computational *whole device modeling* is very challenging. The interplay of several different length and time scales makes simulating the entire tokamak computationally demanding, even in high-performance environments.

Code coupling is an appealing strategy to address these challenges. Coupling separate codes that have been specially optimized for specific spatial regions or for specific types of physical phenomena could provide a more performant tool than a single code simulating the physics of the whole device. The *High-Fidelity Whole Device Modeling* program within the Exascale Computing Project (ECP) is pursuing an approach of this kind. The first physics coupling under investigation by the team is a spatial core-edge coupling, where two independent applications are run and tightly coupled. Though both are gyrokinetic applications for simulating tokamak microturbulence, one has been optimized for the central core region of the device and the other has been optimized for the outer edge of the tokamak. As the WDM project matures and the physics of core-edge coupling is better understood, other multi-physics codes will be coupled-in to complete the whole-device model, such as components for modeling high-energy particles, radio frequency heating, magneto hydrodynamics, etc.

There are numerous technical mathematical issues in whole device modeling that will need to be tested and verified for stability. One is interpolation between different grid types. The core region of a tokamak is well-suited to be modeled on a regular grid, but this is not true at the edge. Similarly, certain Fourier-space representations are valid in the core, but not appropriate for the edge. Rather than needing to adopt a single large code base to support the various modules, it is often more convenient to use a service-focused environment, with well-defined interfaces used to couple the components, so each individual service can be developed and debugged more or less independently.

Coupled modules need not be limited to physics-motivated ones. A coupling-based framework is also useful for enabling supporting technologies that help scientists understand their applications. For example, alongside the simulations, one can deploy visualization services or in situ analysis to study how the coupling is progressing or services to monitor whether good performance is being achieved. These are especially useful in debugging phases during early studies of a new coupling's stability.

Critical Observation: Using different, specialized codes in regions may give better performance, but the representation differences (real-space vs Fourier, alternate meshing, etc.) can cause stability difficulties, both mathematically and computationally.

In Practice: In contrast to the points below, solutions for these issues will be problem-dependent, and thus are beyond the scope of our middleware implementations. For subsequent sections we will highlight the specific technologies employed and provide pointers for prospective users.

B. *Modeling and Implementing the Coupled Physics*

The first two applications to be coupled in the WDM project are GENE [6] for the core and XGC [7] for the edge. Both codes solve for the evolution of the distribution function f of each plasma species according to the same 5D gyrokinetic Boltzmann equation. Brief descriptions that explain each code and offer an introductory sense of their differences are included in § II-B1 and § II-B2, with comments regarding their coupling in § II-B3; readers are referred to the provided GENE and XGC publications for thorough details concerning the codes.

[1] https://www.iter.org/

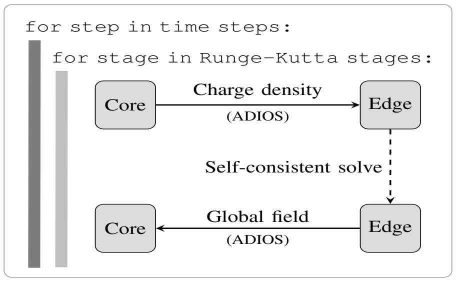


Fig. 2. Prescription for kinetic strong coupling of core and edge fusion simulation applications, using a self-consistent global field solve, with needed communication facilitated through our ADIOS-based framework. Here, GENE is the core executable and XGC is the edge executable. See § II for further physics details.

1) GENE: GENE is an Eulerian gyrokinetic code that solves the Boltzmann equation using the method of lines. The distribution function is first discretized on a fixed grid in phase space and then numerically integrated. To exploit the anisotropy of the fluctuations, GENE employs a magnetic field-aligned coordinate system to represent the fluctuation fields in configuration space. The binormal direction y is Fourier transformed and because of the tokamak axisymmetry, linear modes have fixed Fourier mode k_y, which is related to the toroidal mode number n. GENE makes use of the so-called δf splitting technique, expressing the actual distribution as the sum of an equilibrium part f_0 (a local Maxwellian is the XGC-coupled context) and a fluctuating component f_1, which is numerically evolved by the code.

2) XGC: XGC is a full-f/total-f/δf Particle-in-Cell (PIC) code that can simulate the whole plasma volume, including the region near the outside of the tokamak volume where the magnetic field lines are open instead of closed like in the core. Since this region is included in the calculation, magnetic field-aligned coordinates cannot be used due to singularities at a location called the X-point. Instead cylindrical coordinates are used to allow for a general representation. The torus is discretized along its axis of rotation into poloidal planes, using the same unstructured triangular mesh for each plane. This meshing allows for the representation of complex diverted geometries and wall structures. The node points of the poloidal mesh follow the magnetic field lines to minimize interpolation errors. In general, XGC's background distribution function slowly evolves over time, but the evolution can be forced to zero for consistency with GENE's f_0.

3) GENE-XGC Coupling: An important aspect of the code coupling between XGC and GENE concerns the consistency of the fields with respect to the particle charge density. To ensure consistency, the coupling must be made at the level of the gyrokinetic distribution function, not only the charge or only the fields. Details of the kinetic coupling are beyond the scope of this paper but are fully detailed in Dominski et al. [8]. The computational aspects of coupling were implemented using two independent XGC executables: one for the core and one for the edge. The kinetic turbulence coupling model in the XGC-XGC study is general (except that grid interpolation was not needed), and the same kind of approach is now being used in XGC-GENE coupling, which we present in this paper. Fig. 2 shows the data flow between the applications, implemented in our framework. At each stage of the Runge-Kutta time integrator, charge density information is exchanged between the codes with ADIOS, prior to solving and sending-back the global field equation self-consistently. Similar ADIOS-based data movement between executables will be exemplified and further explained in § III along with the connection to other support services with our framework, such as in situ visualization and analysis.

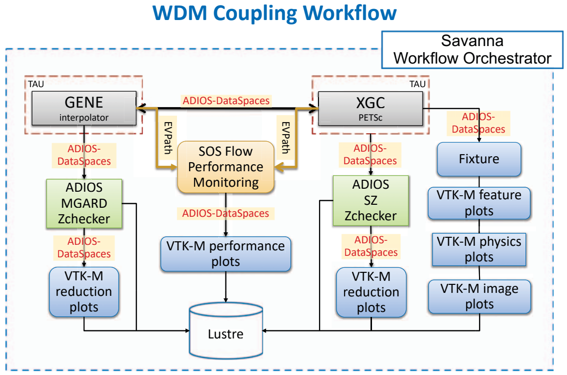


Fig. 3. The Whole Device Modeling fusion application (WDM) workflow.

C. Assembling the Components

Fig. 3 is the workflow of a representative live demonstration. We ran the coupled XGC-GENE simulations on the OLCF's supercomputer Titan, along with 9 other online analysis and visualization applications in parallel, using 3,072 MPI processes for XGC, 1,024 MPI processes for GENE, and 1 process for each analysis/visualization service. and fuller descriptions will be given in § III, where we focus on the details of the framework used for the demonstration.

All applications were independently developed and first tested with file-based data transport, then switched to use memory-based I/Os during the demonstration for better performance. The framework enables a transparent transition from one to the other and also provides performance monitoring capabilities to users in either case. Fig. 1 demonstrates the online information produced during a representative run. Though exhaustive performance studies are not the focus of this paper, we will make a few brief comments. We compared the approach of writing everything to the parallel file system versus using in-memory coupling. For an average total computation time of 680s, writing files took an average of just shy of 7s (a little more than 1% of total time) with substantial variability. In-memory coupling, without any additional tuning, reduced this to 1.7s (or a 0.25% decrease in run time).

Critical Observation: The software framework allows for the rapid integration of different software components through well-defined interfaces, with communication costs comparable to what would be expected for raw rewrite as a monolithic code.

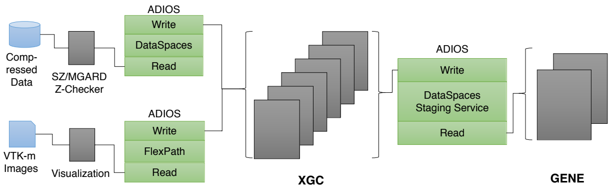


Fig. 4. ADIOS service composition. Taking a deeper look at some of the specific connections in Fig. 3, we see how the ADIOS read and write invocations can be connected by a number of different service implementations.

In Practice: All components leverage the ADIOS API to obtain easily switchable access to different communication and I/O transports through a common interface. ADIOS is mature software available for download at https://www.olcf.ornl.gov/center-projects/adios/.

III. Coupling Framework and In Situ Processing

Many HPC scientific applications involve complex workflows. The workflow we have presented in Fig. 3 and orchestrated using our framework is built upon several technologies. Here we describe those technologies and their interactions in our environment. Our goal is to create an HPC environment that is both flexible and performant for scenarios with many coupled components. Our implementation is able to run the N applications in parallel, with flexible I/O abstractions provided by ADIOS as a backbone for moving data between them. We leverage state-of-the-art HPC staging software, parallel visualization infrastructure, compression kernels, and performance monitoring tools, as well as compose the workflow within a runtime system that allows one to deploy the job in different configurations.

The section is organized as follows. In § III-A we discuss our use of ADIOS for data movement, with the supported staging services highlighted in § III-B. § III-C focuses on the visualization technology and § III-D turns to the compression algorithms used in the workflow. Supported performance monitoring is described in § III-E. Finally, § III-F concludes with the workflow composition and runtime deployment software.

A. Leveraging Flexible I/O Abstractions

Our approach for supporting code coupling relies on three primary functionalities related to the I/O software:

1) an abstract interface so users can easily switch between file-based (for persistence when debugging/checking) and staging-based (for higher throughput) I/O,
2) services to manage the staging areas,
3) online analysis routines to apply to the data in motion.

We leverage diverse ADIOS I/O capabilities to help satisfy these needs. ADIOS [9] is an I/O framework designed primarily for scalable and portable performance on high performance computing platforms. As detailed by the authors of [9], there is a layered software stack internal to ADIOS that allows a user to be isolated from the details of a particular memory-to-memory or storage optimization technique. As seen in Fig. 4, the connections between the components occurs at different scales and frequencies, using different transport properties, but always using the same read and write interface in the exectuables. For example, when users need to run on a parallel Lustre file system, they can leverage a file-based method optimized for Lustre. If users want to run multiple applications in a synchronized fashion, they can easily select a memory-based staging method. Staging is a technique used to buffer data in memory alongside application data, or to move data to an additional set of staging nodes. It can provide a performance improvement compared to the use of files to hold intermediate data, which can incur high costs associated with I/O operations. The ADIOS file vs. staging abstraction provides a convenient way to develop and run coupled simulations. Once each application can independently read and write ADIOS files, users can seamlessly switch to using one of the staging methods in the coupled application, without changing the interface (i.e., without any code rewriting).

We also leverage ADIOS plug-ins for online data transformations. For example, users can compress their data and collect statistics about compression performance while data is in motion and before it is written to disk. Compression methods implemented in ADIOS include lossy compression algorithms such as MGARD, SZ, and ZFP, and lossless techniques such as LZ4, GZIP, and SZIP. The Z-checker [10] system provides for collecting compression-related statistics and other diagnostics.

B. Staging services

ADIOS currently supports a wide range of staging services. However, here we focus on two main staging services, called DIMES and FlexPath. While DIMES [11] is a method for memory-to-memory data transfer based on the hardware supported remote direct memory access (RDMA) protocol, FlexPath [12] is based on a general-purpose event transport middleware layer called EVPath [13]. Further descriptions for each follow below.

The best choice for users will depend on various factors in the workflow. DIMES will perform well in systems with RDMA-based interconnects, though it provides a fall-back mechanism using a general-purpose protocol (TCP/IP) when RDMA is not available. FlexPath supports a wide-range of protocols, which can be used easily for remote site connections (e.g., off-site remote visualization). While DIMES is based on a client/server model that requires extra server services to coordinate data flows, FlexPath uses conventional peer-to-peer style communications with no extra processes or management services.

1) DIMES: DIMES [11] is built on an RDMA-based asynchronous memory-to-memory data transport layer called DART [14]. DART includes a predefined set of communication primitives that takes advantage of RDMA [15] technology and the underlying interconnect network to provide low-latency and high-throughput data transfer capabilities. Through providing high-level data write/read abstractions, DIMES leverages DART and its capability of memory-to-memory coupling, to create a coordination framework that allows different HPC application codes to exchange data asynchronously at runtime.

The DIMES framework is built on a peer-to-peer (P2P) model, where writers hold their data in their own buffer area (i.e., in writer's memory space) and then readers connect to read data directly from the writer's memory. Before establishing such P2P connections, they need to consult with the servers which maintain only meta data (i.e., the location of data). The DIMES server processes are executed on a staging area that is built using a set of dedicated computing nodes, separate from where workflow computations are performed. In order to ensure data consistency, locking mechanisms are provided. DIMES provides infrastructure for the various simulation, analysis, and visualization components to share their data and allows data access in an asynchronous manner.

2) FlexPath: The FlexPath tranport [12] in ADIOS takes the I/O interface offered to the application and implements process-to-process connections using a Publish/Subscribe paradigm. In practice, this means that read and write operations in the two applications are treated as message access or submission (respectively). FlexPath is built upon the EVPath [13] library, which offers an infrastructure for constructing advanced messaging services, including data type management, dynamic routing overlays, and the ability to multiplex over different types of interconnect and networks.

As a result, FlexPath is a very configurable, peer-to-peer system for ADIOS that can leverage both RDMA interconnect transports and IP-based networking protocols (TCP, reliable UDP, multicast, etc.). There is an initial rendezvous operation that must occur so that the readers and writers can find one another; the EVPath-level connection token allows for both sides to advertise all of their available connection opportunities on systems where there are multiple options. Once connected, all subsequent configuration and decisions, such as different selection criteria within ADIOS read operations, are handled through a scalable peer-to-peer protocol.

Critical Observation: Leveraging staging technologies makes it easier for an end user to gain access to advanced networking and interconnect technologies such as RDMA.

In Practice: FlexPath and DIMES enable easy-to-use interapplication communication. The ADIOS Manual (available at https://www.olcf.ornl.gov/center-projects/adios/) contains extensive information about enabling these staging services. Once enabled, codes using the ADIOS API can be switched to use these transports by a simple modification to the ADIOS configuration file.

C. *Visualization*

Visualization plays an important role in both post-hoc understanding of the science and in runtime physics diagnostics for the coupled science simulation. Visualizing data while it is in memory, called in situ visualization [16], [17], allows users to display these diagnostics and to monitor simulation data without interfering with the running simulation. In situ visualizations are performed for a number of different components of our coupling demonstration including: derived physical quantities like flux, simulation mesh data, system performance data, and compression performance/statistics. Examples of these were shown in Fig. 1.

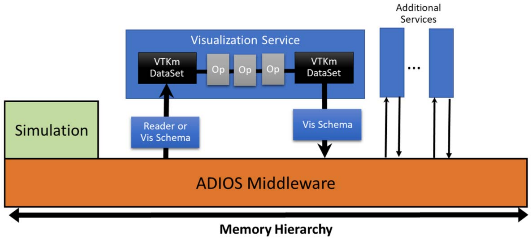


Fig. 5. Overview of the construction of a third party visualization service using ADIOS interfaces.

The key aspect of in situ visualization is to maintain a low profile in I/O for accessing data during a running simulation and to perform visualization functions as efficiently as possible. As shown in Fig. 5, we integrate these service interfaces directly with ADIOS to achieve efficient I/O performance for accessing data from simulation. In addition, users can make use of the ADIOS VisSchema [18] to provide data model semantics describing the ADIOS data stream.

For high performance visualization, these visualization services are built using the Visualization Toolkit for multi- and many-core architectures (VTK-m) [19]. VTK-m provides the analysis and rendering for our workflow. Originally created to address feature gaps in the well-used VTK library [20], VTK-m primarily provides visualization algorithms with a fine degree of concurrency that behave well on modern processor architectures like multi-core CPUs and GPUs. Like its namesake, VTK-m provides both a collection of ready-to-use visualization algorithms and a framework to simplify the addition of new algorithms, port these algorithms across various devices, and integrate these algorithms together. VTK-m's flexible data models and portability simplify its integration into the workflow.

Critical Observation: All simulation output requires some analysis and visualization. Accelerating the time to delivery of these results can be made cheap, thereby enabling higher quality runtime diagnostics and faster time to science.

In Practice: VTK-m is available for download at http://m.vtk.org. Flexible and reusable visualization and analysis components built with VTK-m can be easily connected to an ADIOS-based workflow using the ADIOS VisSchema mechanism.

D. *Compression*

To handle large volumes of data, data compression is an active research and development area in scientific data. Unlike the traditional lossless data compression, lossy compression is getting attention, specifically, but not restricted to, to handle data streams for online processing while data is in memory. We integrate state-of-the-art lossy compression algorithms, MGARD and SZ, with ADIOS and provide an integrated environment to compress data while data is in memory.

1) MGARD: MGARD is a lossy data compression algorithm for multidimensional scientific data inspired by multigrid methods [21], [22], which frequently occur in the solution of differential equations on regular domains and in timestepping applications [23]. In particular, the approach permits the use of nonuniformly spaced grids, which can prove problematic for many types of data reduction methods. Such grids arise, for example, in the simulation of turbulent flows, where Chebyshev nodes are often used.

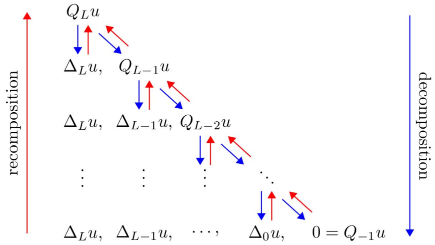

$Q_L u$
$\Delta_L u,\ \ Q_{L-1} u$
$\Delta_L u,\ \ \Delta_{L-1} u,\ Q_{L-2} u$
$\Delta_L u,\ \ \Delta_{L-1} u,\ \ \cdots,\ \ \Delta_0 u,\ \ 0 = Q_{-1} u$
recomposition
decomposition

Fig. 6. Recursive decomposition and recomposition steps in MGARD.

An important feature of MGARD's approach is the provision of guaranteed, computable bounds on the loss incurred by the reduction of the data. Many users are leery of lossy algorithms, and will only consider using them provided that numerical bounds on the pointwise difference between the original and the reduced datasets are given. Accordingly, MGARD provides the user with techniques for bounding the loss measured in the max norm [24]. These bounds are realistic in the sense that they do not significantly overestimate the actual loss. The resulting loss indicators are used to guide the adaptive reduction of the data so that the reduced dataset meets a user-prescribed tolerance or memory constraint.

In addition to providing realistic error bounds, MGARD also provides the user with the possibility of generating partial decompressions. This is illustrated in Fig. 6, where the decomposition step consists of successively splitting the input u into $\Delta_\ell u$ and $Q_{\ell-1} u$ where $\ell \in [0, L]$. Here L corresponds to the maximum number of refinements (levels), and Q_ℓ is the projection operator to level ℓ. The decomposition then continues by splitting $Q_{\ell-1} u$ while $\Delta_\ell u$ can be sent to storage. Recomposition in MGARD mirrors the decomposition, so $Q_\ell u$ is repeatedly constructed from $\Delta_\ell u$ and $Q_{\ell-1} u$. Hence the user can choose to perform a partial decompression by deciding on the number of decomposition cycles. This is possible as $(Q_\ell u)_{\ell=0}^{L}$ is a sequence of representations of u with increasing fidelity.

2) SZ: SZ [25], [26] is a state-of-the-art, error-bounded lossy compressor for significantly reducing the data size of extreme scale scientific simulations. SZ compression contains three fundamental steps: (I) value prediction on each data point for the sake of decorrelation (as illustrated in Figure 7), (II) linear-scaling quantization surrounding the predicted value with equal-sized bins (as illustrated in Figure 7), and (III) variable-length encoding used to encode the integer indices of the bins. In step I, SZ performs a single-dimensional or multi-dimensional prediction for each data point based on its neighbor data points (the dimension of the prediction depends on the dimension of the data set). A set of consecutive bins with each twice the error bound in length are constructed in Step II, and the index of the bin containing the real value of the data point (called the located bin) are encoded by a customized Huffman encoding in step III. Steps I and III are both lossless procedures, which means that these two steps will not introduce data loss during their corresponding decompression steps.

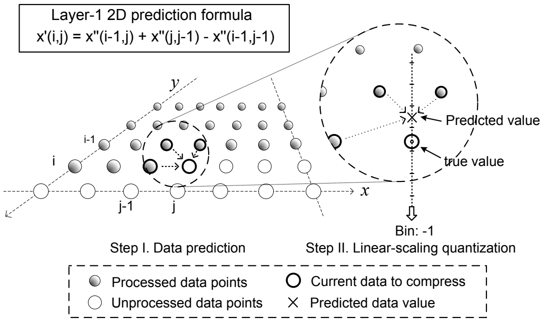


Fig. 7. Illustration of Step I and II in SZ compression for 2D data.

In order to guarantee that the difference between the original data value and the decompressed value is always bounded within the user-specified error bound, SZ performs the data prediction based on the **decompressed** values instead of the original data values. As shown in Figure 7, in order to strictly respect the error bound for the compression of data point (i,j), we need to predict its value by the decompressed values $x''(i{-}1,j)$, $x''(i,j{-}1)$, and $x''(i{-}1,j{-}1)$ rather than the original values $x(i-1,j)$, $x(i,j-1)$, $x(i-1,j-1)$. If the predicted values are too far from their real values, they are treated as unpredictable data points and compressed by truncating the insignificant bits of their IEEE 754 binary representations according to the required error bounds.

SZ provides multiple ways to control the lossy compression errors for users on demand: (1) absolute error bound is a constant value uniformly applied onto each data point; (2) relative error bound is a ratio value compared with each data point's value for the error control (i.e., the larger the value, the larger the compression error of that data point); (3) peak signal-to-noise ratio (PSNR) is a statistical metric to evaluate the overall compression error and SZ allows users to compress data by a given PSNR. SZ also allows combining the above different types of error bounds in the compression to fit diverse data sets more flexibly.

3) Z-checker: Z-checker [10] is an efficient toolkit/library for assessing the lossy compression quality of specific datasets. Compared to the previous version of Z-checker [10] that can only work with an offline mode (performing offline analyses based on existing data files), the new version coupled with the ADIOS framework supports an online execution mode. The complete design framework of Z-checker is illustrated in Figure 8. With online execution support, users can run Z-checker with HPC simulations in parallel, such that the compression quality of each snapshot/timestep can be dynamically

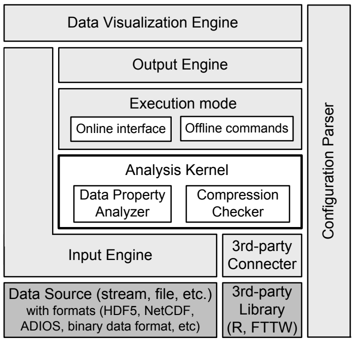


Fig. 8. Design framework of Z-checker with online execution mode.

observed at runtime. It can help compression developers and application users deeply understand the impact of data changes on compression quality.

By integrating the online interface of Z-checker, our coupling framework supports a rich set of parallel analysis metrics related to lossy-compression quality, including 1D auto-correlation, 3D auto-correlation, distribution of compression errors, spectrum analysis based on Fast Fourier Transform FFT), structural similarity index (SSIM), KS-test, peak signal-to-noise-ratio (PSNR), compression/decompression rate, compression ratio, entropy, etc.

Critical Observation: In lossy compression, there is trade-off between the fidelity of the reconstructed data and the degree of compression. Users need detailed diagnostic information to understand these choices between methods and parameters in order to achieve their overall goal.

In Practice: A variety of compression routines have been incorporated into the ADIOS Transformation Layer. Users should follow the ADIOS build instructions to link specific compression routines for use at runtime. The Z-checker toolkit is available at http://github.com/CODARcode/Z-checker.

E. Performance monitoring

One of the goals of the coupled application was to provide a performance "dashboard" for the user, in order to monitor the distribution and general characteristics of the overall workflow. Through such an interface, the user could identify whether the allocation was being used efficiently and/or identify potential problems such as out-of-memory errors or load imbalances, address them if possible, and potentially correlate those behaviors with the runtime state. To provide application performance measurement, we integrated the TAU Performance System [27], which contains a broad set of utilities and measurement libraries for parallel applications. Global, runtime access to performance information from multiple distributed codes requires an aggregation infrastructure, and for this project we integrated SOSflow [28], [29] – a flexible, scalable, and programmable framework for observation, introspection, feedback, and control of HPC applications.

The TAU (Tuning and Analysis Utilities) Performance System gathers performance data from parallel applications through several methods, including source instrumentation, binary instrumentation, profiling interfaces, runtime callbacks, and periodic sampling. For the purposes of the integrated performance monitoring, the XGC and GENE applications were each linked with the TAU measurement library – no other modification to the binaries was necessary. TAU uses the MPI standard profiling interface [30] to provide instrumented measurement of communication events within each of the applications. In addition, version 1.13 of the ADIOS library provides a callback API to allow development tools to monitor ADIOS calls without the need to instrument or modify the ADIOS library. TAU has integrated support for the ADIOS callback API and can measure the communication between XGC and GENE as the interpolation boundary is exchanged between the applications, as well as when each of the applications writes out checkpoint data, analysis data or any other output using the ADIOS library. TAU is also integrated with PAPI [31], which provides portable access to hardware counters. The measurement was configured to capture time spent in application code, floating point operations, and monitor the memory high water mark (HWM) and resident set size (RSS). TAU was also configured to integrate with SOSflow.

The Scalable Observation System (SOS) performance model used by SOSflow allows a broad set of online and in situ capabilities including remote method invocation, data analysis, and visualization. SOSflow can couple together multiple sources of data, such as application components and operating environment measures, with multiple software libraries and performance tools, efficiently creating holistic views of performance at runtime. For this integration, SOSflow was configured to aggregate data from all of the application processes over a network of listener and aggregator daemons organized as a broad, shallow tree. For the large scale execution, 256 listeners and 4 aggregators were used to collect the TAU performance data. An analysis client was launched on the same node as one of the aggregator processes. This client queried each of the aggregator daemons to extract the performance data from the most recent time period, as it became available. This data was then staged as ADIOS data to be visualized in the user's performance desktop, as shown in Fig. 1.

Critical Observation: Having performance information enables detecting and identifying runtime problems which typically are difficult to anticipate or to observe in a complex computing environment with coupled applications.

In Practice: TAU is well-established, and available for download at http://www.cs.uoregon.edu/research/tau/home.php. SOSFlow may be found at https://github.com/cdwdirect/sos_flow. We recommend that users contact the authors of [28] for assistance in integrating SOSFlow with existing workflows.

F. Scalable workflow management

Managing the runtime execution of coupled workflows can become quite burdensome. Several applications may need to be launched, each with its own input parameters and parallelization setup. Different architectures or schedulers may require different submission syntaxes. Resource provisioning between the applications may have a significant impact on performance, and how to achieve the best load balance is not necessarily obvious.

We leverage the Savanna runtime infrastructure [32] in our experiments to orchestrate the complete workflow consisting of the two simulation applications and multiple analysis and visualization components. Savanna provides a way to manage complex workflows consisting of multiple coupled components, with the objective of exporting a comprehensive list of data events and associated actions that can be taken dynamically. Savanna provides a Python-based specification that can be used to define an experiment. An experiment specification consists of a definition of all application components, how to launch them, and the dependencies between them. Savanna works on several leadership class supercomputers, including Titan. Performance of the runtime components was generated at runtime using the SOSflow library and visualized using VTK-m.

Application scientists can use Savanna to create an abstract workflow specification that is independent of the underlying system. The specification describes the different codes that compose the workflow and how they interact with each other. Savanna provides native support for all the tools we have described in this paper including ADIOS, DIMES, FlexPath, TAU, and SOSflow. Different I/O options can be explored such as turning on transforms or staging. Parallelization settings (the number of MPI processes, threads, etc.) can be adjusted and nodes can be configured to co-located multiple applications if desired. Savanna automates actions such as spawning the correct number of DIMES servers and SOSflow daemon processes depending on the configuration of all these workflow components and determining the minimum number of nodes to run the job.

Critical Observation: Executing coupled applications and analysis components requires more sophistication than launching a monolithic MPI code. Support from workflow environments to make composition and deployment easier for end users is a requirement for broader adoption.
In Practice: The Savanna runtime is currently packaged together with the Cheetah exascale codesign testing framework. It is available at https://github.com/CODARcode/cheetah.

IV. Results and Discussion

We present here some of the performance details from the runs described in § II-C. Each piece represents a specific set of performance measurements on either the science data or on the infrastructure of the framework. Other than the staging performance survey, the others are drawn from Fig. 1.

Staging performance: Given the emphasis in this work on using memory-to-memory coupling technologies within the ADIOS I/O framework, it is natural to ask what the performance comparison is between traditional ADIOS files and an ADIOS memory-based method, DIMES. By using the XGC-GENE coupling case, presented in § II-C, we measured the I/O time spent in XGC for writing field data and reading density data from GENE and compared i) the file-based method (i.e., writing/reading file objects through parallel filesystem) and ii) in-memory coupling with the DIMES method (i.e., writing/reading in process memory) on Titan at ORNL. We also performed the same experiment on Cori at NERSC, where we could also evaluate file coupling by way of the DataWarp Burst Buffer system [33].

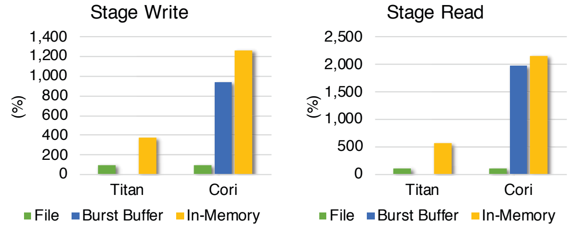


Fig. 9. Comparison of the performance improvements with different coupling methods, normalized against the file-based method which appears at 100%. On Cori, we also measured the relative improvement of file-based coupling with the DataWarp system.

In Fig. 9, we present the percent improvement in I/O time relative to the file-based methods on Titan and Cori. We observed about 3.8x and 12.7x writing performance improvement and 5.7x and 21.5x reading performance improvement with the in-memory method on Titan and Cori, respectively. We also observed 9.4x and 19.7x improvement with the file-based method on the Cori DataWarp system, which is a clear improvement over the parallel file system but still less than the in-memory based method. These performance gains open the door to higher fidelity coupling at an increased frequency.

Memory high-water mark: TAU integration with ADIOS can provide an option to easily turn on performance monitoring of the applications' I/O usage through ADIOS, just as the PMPI tool interface allows for MPI. Users can monitor at runtime application's memory usage, CPU performance, MPI communications, etc.

In the demo, we built XGC and GENE with ADIOS and TAU integration. During execution, we captured a set of performance numbers at every second (which is about 3-4 simulation time steps). Fig. 10 shows one of those performance numbers captured during the run. In Fig. 10, (a) high-water memory usage and (b) total flops are plotted at each rank. The first 3,072 ranks represent XGC processes and the last 1,024 ranks are for GENE. In the plot, we identified an unusual memory usage pattern; memory usage in XGC is not well balanced, which might affect performance in the later time steps. Developers can pay attention for this memory issues in the next development step.

Entropy and compression ratio: In the XGC-GENE simulations, plasma turbulence is modeled, and the coupled application outputs data to monitor the turbulence's behavior. This output is saved at high frequency, and lossy data reduction was tested, with statistics about the reduced data's properties

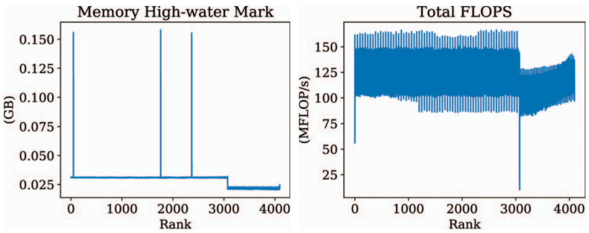
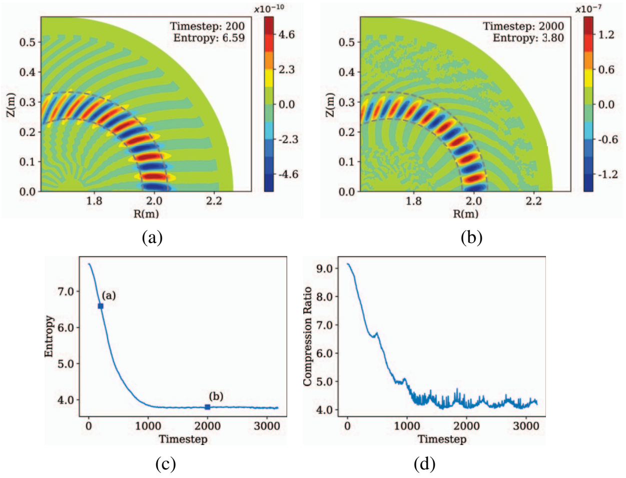


Fig. 10. Performance information captured by TAU during the XGC-GENE coupling performed on Titan. The memory high water mark and total FLOPS measurements of each MPI rank for XGC (rank $<$ 3,072) and GENE (rank $>$ 3,072) were captured. These measurements are at about timestep 2,000.



(a)

(b)

(c)

(d)

Fig. 11. Compression ratio and entropy results for early (a) and late (b) timesteps. Panels (c) and (d) show the changes in the quantities as a function of time. MGARD and SZ were used for data compression. (SZ plots are shown in the figure.)

collected in transit. Here, we focus on the compression ratio and entropy measurements collected by Z-checker.

The online compression and analysis was performed at each time step, applying SZ and MGARD to the appropriate data through Z-checker. Fig. 11 shows the results. Panels (a) and (b) visualize the evolution of the plasma: (a) is an early step (timestep 200) and (b) shows a later timestep (timestep 2000). Panels (c) and (d) show the changes of the entropy and compression ratios with time.

Qualitatively, the trends are plausible. The simulations are seeded with random variations, which is why the entropy starts the highest. Over time, the fluctuation become more correlated. Compression ratios decrease because the fluctuations grow in magnitude, which is harder to compress under the constraint of point-wise error limits.

Physics Diagnostics: Several of our in-situ visualizations (Fig. 1) were diagnostics to help study the stability of the coupling. Though explaining the physics content of all of these is beyond the scope this paper, panels (a) and (b) of Fig. 11 themselves offer a simple qualitative check. The dashed boundary is the coupling region between GENE and XGC in which data is exchange. If there were numeric instabilities or other problems with the coupling algorithm, a likely outcome would be discontinuities or suppression of the features in this region. No such issues appear. Other diagnostics compared the results from the coupled runs to results from reference simulations, to confirm that both indeed converged to the same answer.

V. Conclusions

With the growing computational complexity of science as well as that of new and emerging hardware, it is time to re-evaluate the traditional monolithic design of computational codes. Code coupling, also known as coupled simulation or co-simulation, is widely used in multi-scale and multi-physics studies. However, the complexity of code coupling frameworks and of the novel hardware and deep memory and storage hierarchies for future computing (exascale and post-Moore's-Law) poses a huge burden to users.

To address these concerns, we present a new framework that is constructed by leveraging high performance capabilities such as in-memory communications, workflow scheduling on HPC resources, and continuous performance monitoring. Specifically, we utilize a combination of the Adaptable I/O System (ADIOS), the Savanna/Cheetah runtime, the MGARD and SZ compression libraries, the VTK-m visualization toolkit, and the TAU performance monitoring system. Evaluated at several leading computing facilities, this framework has been applied to solve a complex multi-physics simulation of Tokamak fusion devices.

Through this effort we have collected a large number of critical observations on the requirements of any such framework when working with real scalable science simulations. Of these, the key overall observation is that integrative science teams need new support and computational services in order to function effectively. Balancing the productivity requirements and the science complexity that comes with a push towards exascale science is difficult for a single research team or monolithic code base to achieve. However, with attention to management of the interfaces between collaborating components, exascale-ready frameworks can offer both performance and the runtime diagnostic capabilities that make full use of the promise of extreme scale platforms today and tomorrow.

Acknowledgement

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 and by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research and Office of Fusion Energy Sciences under Contracts DE-AC02-06CH11357, DE-AC02-09CH11466, and DE-AC05-00OR22725.

This work used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, the National Energy Research Scientific Computing Center, and the Oak Ridge Leadership Computing Facility, which are DOE Office of Science User Facilities supported under Contracts DE-AC02-06CH11357, DE-AC02-05CH11231, and DE-AC05-00OR22725, respectively.

References

[1] Y. Babuji, A. Brizius, K. Chard, I. Foster, D. S. Katz, M. Wilde, and J. Wozniak, "Introducing Parsl: A Python Parallel Scripting Library," Aug. 2017. [Online]. Available: https://doi.org/10.5281/zenodo.891533

[2] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, "Scientific workflow management and the kepler system," Concurrency and Computation: Practice and Experience, vol. 18, no. 10, pp. 1039–1065, 2006.

[3] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. F. da Silva, M. Livny et al., "Pegasus, a workflow management system for science automation," Future Generation Computer Systems, vol. 46, pp. 17–35, 2015.

[4] M. Wilde, M. Hategan, J. Wozniak, B. Clifford, D. Katz, and I. Foster, "Swift: A language for distributed parallel scripting," Parallel Computing, vol. 37, no. 9, pp. 633–652, 2011.

[5] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. R. Pocock, P. Li, and T. Oinn, "Taverna: a tool for building and running workflows of services," Nucleic acids research, vol. 34, no. suppl_2, pp. W729–W732, 2006.

[6] T. Görler, X. Lapillonne, S. Brunner, T. Dannert, F. Jenko, F. Merz, and D. Told, "The global version of the gyrokinetic turbulence code gene," Journal of Computational Physics, vol. 230, no. 18, pp. 7053–7071, 2011.

[7] S. Ku, R. Hager, C.-S. Chang, J. Kwon, and S. E. Parker, "A new hybrid-lagrangian numerical scheme for gyrokinetic simulation of tokamak edge plasma," Journal of Computational Physics, vol. 315, pp. 467–475, 2016.

[8] 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, vol. 25, no. 7, p. 072308, 2018.

[9] Q. Liu, J. Logan, Y. Tian, H. Abbasi, N. Podhorszki, J. Y. Choi, S. Klasky, R. Tchoua, J. Lofstead, R. Oldfield, M. Parashar, N. Samatova, K. Schwan, A. Shoshani, M. Wolf, K. Wu, and W. Yu, "Hello ADIOS: the challenges and lessons of developing leadership class I/O frameworks," Concurrency and Computation: Practice and Experience, vol. 26, no. 7, pp. 1453–1473, may 2014. [Online]. Available: http://doi.wiley.com/10.1002/cpe.3125

[10] D. Tao, S. Di, H. Guo, Z. Chen, and F. Cappello, "Z-checker: A framework for assessing lossy compression of scientific data," The International Journal of High Performance Computing Applications, vol. 0, no. 0, p. 1094342017737147, 0. [Online]. Available: https://doi.org/10.1177/1094342017737147

[11] F. Zhang, T. Jin, Q. Sun, M. Romanus, H. Bui, S. Klasky, and M. Parashar, "In-memory staging and data-centric task placement for coupled scientific simulation workflows," Concurrency and Computation: Practice and Experience, vol. 29, no. 12, 2017.

[12] J. Dayal, D. Bratcher, G. Eisenhauer, K. Schwan, M. Wolf, X. Zhang, H. Abbasi, S. Klasky, and N. Podhorszki, "Flexpath: Type-based publish/subscribe system for large-scale science analytics," in Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on. IEEE, 2014, pp. 246–255.

[13] G. Eisenhauer, M. Wolf, H. Abbasi, and K. Schwan, "Event-based systems: opportunities and challenges at exascale," in Proceedings of the Third ACM International Conference on Distributed Event-Based Systems. ACM, 2009, p. 2.

[14] C. Docan, M. Parashar, and S. Klasky, "Dart: a substrate for high speed asynchronous data io," in Proceedings of the 17th international symposium on High performance distributed computing. ACM, 2008, pp. 219–220.

[15] J. Pinkerton, "The case for RDMA," RDMA Consortium, May, vol. 29, p. 27, 2002.

[16] J. Kress, S. Klasky, N. Podhorszki, J. Choi, H. Childs, and D. Pugmire, "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. ACM, 2015, pp. 1–6.

[17] D. Pugmire, J. Kress, J. Choi, S. Klasky, T. Kurc, R. M. Churchill, M. Wolf, G. Eisenhower, H. Childs, K. Wu et al., "Visualization and analysis for near-real-time decision making in distributed workflows," in Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International. IEEE, 2016, pp. 1007–1013.

[18] R. Tchoua, J. Y. Choi, S. Klasky, Q. Liu, J. Logan, K. Moreland, J. Mu, M. Parashar, N. Podhorszki, D. Pugmire, and M. Wolf, "Adios visualization schema: A first step towards improving interdisciplinary collaboration in high performance computing," pp. 27–34, 10 2013.

[19] K. Moreland, C. Sewell, W. Usher, L. ta Lo, J. Meredith, D. Pugmire, J. Kress, H. Schroots, K.-L. Ma, H. Childs, M. Larsen, C.-M. Chen, R. Maynard, and B. Geveci, "Vtk-m: Accelerating the visualization toolkit for massively threaded architectures," IEEE Computer Graphics and Applications, vol. 36, no. 3, pp. 48–58, May/June 2016.

[20] W. Schroeder, K. Martin, and B. Lorensen, The Visualization Toolkit: An Object Oriented Approach to 3D Graphics, 4th ed. Kitware Inc., 2004, ISBN 1-930934-19-X.

[21] M. Ainsworth, O. Tugluk, B. Whitney, and S. Klasky, "Multilevel Techniques for Compression and Reduction of Scientific Data—The Univariate Case," Comput. Vis. Sci., vol. submitted, 2017.

[22] ——, "MGARD: A multilevel technique for compression of floating-point data," in DRBSD-2 Workshop at SuperComputing 2017, Colorado, USA, 2017.

[23] M. Ainsworth, S. Klasky, and B. Whitney, "Compression Using Lossless Decimation: Analysis and Application," SIAM J. Sci. Comput., vol. 39, pp. B732–B757, Aug. 2017.

[24] M. Ainsworth, O. Tugluk, B. Whitney, and S. Klasky, "Multilevel Techniques for Compression and Reduction of Scientific Data—The Multivariate Case," SIAM J. Sci. Comput., vol. submitted, 2018.

[25] S. Di and F. Cappello, "Fast error-bounded lossy HPC data compression with SZ," in 2016 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2016, Chicago, IL, USA, May 23-27, 2016, 2016, pp. 730–739.

[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 2017, Orlando, Florida, USA, May 29-June 2, 2017, 2017.

[27] S. S. Shende and A. D. Malony, "The tau parallel performance system," The International Journal of High Performance Computing Applications, vol. 20, no. 2, pp. 287–311, 2006.

[28] C. Wood, S. Sane, D. Ellsworth, A. Gimenez, K. Huck, T. Gamblin, and A. Malony, "A scalable observation system for introspection and in situ analytics," in 2016 5th Workshop on Extreme-Scale Programming Tools (ESPT), Nov 2016, pp. 42–49.

[29] C. Wood, M. Larsen, A. Gimenez, C. Harrison, T. Gamblin, and A. Malony, "Projecting performance data over simulation geometry using SOSflow and ALPINE," in 4th International Workshop on Visual Performance Analysis (ESPT), 2017.

[30] The MPI Forum, MPI: A Message Passing Interface Standard, Version 3.1. University of Tennessee, Knoxville, Tennessee, 2015.

[31] S. Browne, J. Dongarra, N. Garner, K. London, and P. Mucci, "A scalable cross-platform infrastructure for application performance tuning using hardware counters," in Supercomputing, ACM/IEEE 2000 Conference. IEEE, 2000, pp. 42–42.

[32] I. Foster, M. Ainsworth, B. Allen, J. Bessac, F. Cappello, J. Y. Choi, E. Constantinescu, P. E. Davis, S. Di, W. Di et al., "Computing just what you need: Online data analysis and reduction at extreme scales," in European Conference on Parallel Processing. Springer, 2017, pp. 3–19.

[33] W. Bhimji, D. Bard, M. Romanus, D. Paul, A. Ovsyannikov, B. Friesen, M. Bryson, J. Correa, G. K. Lockwood, V. Tsulaia et al., "Accelerating science with the nersc burst buffer early user program," Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States), Tech. Rep., 2016.