Plasma fusion code coupling using scalable I/O services and scientific workflows

Norbert Podhorszki, Scott Klasky, Qing Liu
Oak Ridge National Laboratory
Oak Ridge, TN, 3738, USA
{pnorbert,klasky}@ornl.gov

Ciprian Docan and Manish Parashar
Center for Autonomic Computing
Rutgers University,Piscataway, NJ 08854, USA
{parashar,docan}@cac.rutgers.edu

Hasan Abbasi Jay Lofstead, Karsten Schwan, Matthew Wolf and Fang Zheng
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332, USA
{habbasi,lofstead,schwan,mwolf,fzheng}@cc.gatech.edu

Julian Cummings
Center for Advanced Computing Research
California Institute of Technology
Pasadena, CA 91125, USA
cummings@cacr.caltech.edu

ABSTRACT

In order to understand the complex physics of mother nature, physicist often use many approximations to understand one area of physics and then write a simulation to reduce these equations to ones that can be solved on a computer. Different approximations lead to different equations that model different physics, which can often lead to a completely different simulation code. As computers become more powerful, scientists can either write one simulation that models all of the physics or they produce several codes each for different portions of the physics and then 'couple' these codes together. In this paper, we concentrate on the latter, where we look at our code coupling approach for modeling a full device fusion reactor. There are many approaches to code coupling. Our first approach was using Kepler workflows to loosely couple three codes via files (memory-to-disk-to-memory coupling). This paper describes our new approach moving towards using memory-to-memory data exchange to allow for a tighter coupling. Our approach focuses on a method which brings together scientific workflows along with staging I/O methods for code coupling. Staging methods use additional compute nodes to perform additional tasks such as data analysis, visualization, and NxM transfers for code coupling. In order to transparently allow application scientist to switch from memory to memory coupling to memory to disk to memory coupling, we have been developing a framework that can switch between these two I/O methods and then automate other workflow tasks. Our hybrid approach allows application scientist to easily switch between in-memory coupling and file-based coupling on-the-fly, which aids debugging these complex configurations.

Categories and Subject Descriptors
J.2 [**Physical Sciences and Engineering**]: Physics

General Terms
Performance, Design, Experimentation.

Keywords
Parallel I/O, workflow design, workflow execution, code coupling, plasma simulation.

1. INTRODUCTION

One of the goals of modeling complex phenomena, such as in the Fusion Simulation [1] is to couple codes, which are either legacy codes, or on-going research in their own domain. The complexity of coupling codes in this project are that some require loose coupling strategies and some require tight coupling. Our motivating example is shown in Figure 1, where we are trying to couple two fusion simulation codes, the GTC [2] and XGC [3] codes for core-edge coupling in a fusion reactor. They both calculate with billions or trillions of individual particles so, in this example, there will be frequent coupling of large amounts of data from large number of processors. In this example, each code separately runs on over 20K processor cores and the amount of data that can be exchanged, as field information, can be well over 1 GB of data, which needs to be exchanged every timestep of the calculation. A timestep may be completed as frequently as every second. Clearly, this operation will stress most file systems since the overhead of writing and then reading from disk can be almost a second on most systems. The approach taken is to couple the data in memory using a shared space abstraction [4]. We want a workflow to monitor the minimum and maximum values of the variables being passed between the codes. By transferring this provenance information over from the coupled simulation to the workflow engine, we allow the workflow to understand what is happening in the simulation and act on some special conditions.

A typical scenario is the following: the maximum value of one of the variables grows greater than some acceptable value during the code coupling. The application scientists then want the ability to transparently switch from memory-to-memory to memory-to-memory and disk so they can monitor the data. By transparently allowing this to occur in the I/O layer of the simulation, the workflow monitoring system can start seeing files being generated, and transfer the file(s) over to another computer system, and then run a series of analysis and visualization tasks. The generated images are accessible on our dashboard [5], a web portal, where the scientist can see the results and then kill the simulation if the results look bad, otherwise let the workflow turn off the memory-to-disk portion of the I/O to allow the coupled simulation to return back to full speed using only memory-to-memory coupling.

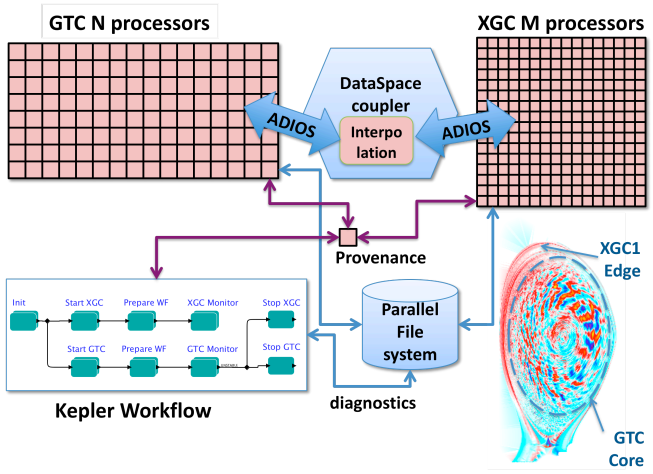

Figure 1. Coupling GTC and XGC for core-edge simulation using ADIOS and DataSpaces. A Kepler workflow monitors the coupling.

Our approach to code coupling is to extend the ADIOS componentized I/O framework [6], described in section 3.1, to support in-memory coupling by switching the transport method from I/O to in-memory coupling. ADIOS allows for I/O multiplexing, where multiple methods can be used simultaneously. For example if a user and/or a workflow actor detects that data should be written to disk as well as moved in memory to another component, it can simultaneously switch ADIOS to do both, allowing for an easy-to-use system for coupling codes together on a single platform or on different platforms and allow the system to automatically switch from memory-to-memory over to memory-to-disk. This gives users the flexibility to switch between these two I/O methods when they are in the process of debugging the system or when new physics is added into the codes. It also allows users to save to disk both the provenance information from the data that is being exchanged from the codes as well as the actual data being exchanged.

In this paper we present the components that enable us to realize the vision above. The provenance framework in Kepler [7] allows us to record information from the simulation in the I/O pipeline. The use of provenance in our existing fusion framework is discussed in section 2. Section 3 presents trends in scalable I/O research. In Section 4 we show how the simulation I/O is connected to the Kepler workflow directly via a new ADIOS method and discuss the events observed in the workflow. Subsection 4.3 discusses different applications of this technique and includes a description of an already existing fusion code coupling application and the effects on the workflow implementations by going from disk-based coupling to memory-to-memory coupling.

1.1 Related work

There are two pieces of related work. The Argonne Model Coupling Toolkit (MCT) is a framework that facilitates model coupling between model components in the Common Community Climate System Model (CCSM). MCT uses a flux coupler to transfer data between physics simulation components. In this approach, message exchange schedules cannot be changed dynamically since they are hard-coded in the source codes. Load balancing is also hard-coded in the source and is performed explicitly in MCT. In order to couple the Weather Research & Forecasting (WRF) model to the Regional Ocean Model System to study hurricanes[1], they combined the WRF I/O with the MCT coupling infrastructure[2]. This coupling system is designed for data exchange between moderately sized codes running on separate Grid resources, but it can have scalability issues and was not made to be integrated with a scientific workflow system.

The Distributed Data Broker (DDB) project [8] also targets the problem of code coupling, but has a different approach than ours. DDB uses a central resource, i.e., a broker, to gather global information about the data distribution of the coupled applications, and to compute required communication schedules between the applications. Data coupling is formulated using the producer-consumer pattern, where the producer uses the pre-computed communication schedules to send data to the consumer. While this is an elegant distributed solution, it assumes predefined and static coupling behaviors, imposes tight synchronization requirements, and implicitly assumes that the end applications can directly communicate with each other.

2. FUSION WORKFLOWS FOR CPES

The Center for Plasma Edge Simulation (CPES) project aims to develop an integrated and predictive plasma edge simulation package [1] applicable to existing magnetic fusion devices and future burning plasma experiments like ITER (the International Thermonuclear Experimental Reactor). The computer science participants of the project are developing the End-to-end Framework for Fusion Integrated Simulation, or EFFIS [9], which addresses all the challenges that code coupling and multi-institutional collaboration of physicist, applied mathematicians and computer scientists face: massive data I/O, data exchange between codes, coordination of code activities/executions, post-processing, archiving, and a collaborative portal for analysis. In this project, we selected Kepler for constructing workflows and extended Kepler for these purposes in [10]. We use Kepler to coordinate code coupling, transfer data for exchange between codes, and to visualize the diagnostic output of each code on the fly so that an application scientist can monitor the progress of the simulation from a web browser. A file-based coupling workflow for the kinetic and MHD codes was described in detail in [11]. Another workflow, described in [12], to monitor XGC1 simulations of micro-turbulence in the plasma edge has been extensively used for hundreds of simulations. Similar monitoring

[1] http://nctr-people.pmel.noaa.gov/cmoore/wrf-roms/index.html

[2] http://www-ad.fsl.noaa.gov/ac/schaffer/mct_wrf_io_api.html

workflows have been built for core turbulence simulations in other projects for GTC, GEM and PIXIE3D based on this workflow. The GTC and XGC1 monitoring workflows are incorporated into of the XGC-GTC coupling workflow.

2.1 Provenance recording from source code to movies

Application scientists need to analyze and visualize data produced by simulations to gain knowledge from the simulations. Both run-time and post analysis require additional meta-data or provenance information to track what data is used to create a visualization or analysis result and what operation(s) were applied to it. Provenance recording is the key to enabling the dashboard to hide the details from the users allowing them to focus on the scientific variables instead of keeping track of the thousands of files that may be generated from one simulation run. The Scientific Process Automation group (SPA) of the DOE Scientific Data Management Center (SDM) developed the Kepler provenance framework [13] that we use in EFFIS to record and retrieve the data lineage. For example, if a user wants to execute an analysis job on the dashboard, the user selects which variables to include in the analysis and executes the analysis without knowledge about the actual location and names of the files [5]. Data for analysis is selected by the user as a movie or a frame of the movie. Data lineage information is used to discover what image is behind that frame and what data file(s) were used to produce that image. Then the analysis job is executed on that data. In a similar fashion, the user might want modifications to the visualization shown on the dashboard and can run new visualizations based on the variables, not on the files. Here the dashboard has to determine what data file and what image creation method was used so that it can rerun that method with the new options.

The workflow also records the system environment information including: the name of the computer where the simulation code was built, the list of libraries and versions used, and the source code of the simulation. After a simulation, the user can review the simulation source files and environment for the application including the source code of the analysis routines. Similarly, the environment in which the code was built can be reviewed.

The Kepler provenance framework records all details about all of the entities in a workflow application, which becomes a large amount of collected meta-data when running workflows. Since these workflows are complex, we had to add several mechanisms for mining the provenance data to track files that exist on disk as well as files that have been archived by the workflow. The recording of provenance information and the algorithms to track the provenance for the dashboard use is described in detail in [14].

3. SCALABLE I/O TRENDS

High Performance Computing (HPC) systems continue to grow in size and complexity. As the systems grow in size, they increase both their aggregate memory and their overall computational power. For example, the recent additions to the Cray-XT5 computer at ORNL have increased the number of processors (cores) to almost 150K cores from 30K cores and it has over 300 TB of memory. In order to write data efficiently, the I/O system was upgraded from 60GB/s to over 220 GB/s. Although these numbers sound outstanding, the ‘actual’ real-world performance can be orders of magnitude less than these results. The I/O system contains thousands of disks. In order to obtain optimal I/O performance, one must be able to use all of the disks to gain maximum parallelism in the I/O system. Furthermore, complex file formats like NetCDF and HDF5, which require writes to be in a contiguous logical format, can stress the shared network on Cray XT and Infiniband architectures. This often causes poor parallel I/O performance as shown in our previous research [16]. Researchers are also forced into making difficult coding decisions to use either complex APIs or simple I/O in binary or ASCII format lacking metadata. These decisions often force them into using non-optimal I/O practices when porting their code to new architectures and may even render the I/O useless.

There are different directions in I/O research to overcome this problem and provide optimal tools for all I/O scenarios code developers face. Our solution is to use ADIOS, which can make coding I/O easy (easy to use APIs) yet provides ways to choose the best performing methods to write data out without modifying the source code of the simulation. DataSpaces is one ADIOS method, described in 3.1, which allows the exchange of data between two codes running on the same computer through memory avoiding file I/O. Staging and in-line processing can further improve the I/O performance when writing files or perform additional operations on the data being exchanged between two codes. These developments are described in the following subsections.

3.1 ADIOS

The ADaptable I/O System, (ADIOS) is a componentization of the I/O layer. It provides an easy-to-use programming interface, which can be as simple as FORTRAN file I/O statements. ADIOS abstracts I/O metadata information and data structures from the source code into an external XML file reducing code pollution and creating the connection between high-level APIs and underlying I/O implementation details, such as buffering and scheduling. By separating the detailed I/O implementation from the APIs, ADIOS also allows users to simply change the declaration of the I/O methods in the XML file without any source code modification.

Some of the key features of ADIOS are that it allows for extremely fast I/O [15], has a high level of resiliency [16], and can be used for creating multiple operations in I/O staging [17]. ADIOS can also be used to select different methods, at or during runtime, for coupling codes together, in memory, as well as file-based coupling. By using the provenance capturing method [18], ADIOS can capture the provenance information in both cases, without changing the codes or the workflow.

ADIOS development has initially concentrated on write performance. In Figure 2 below, twenty GTC code runs measured twice with two different actual I/O methods are shown. We show that we can achieve 70 GB/s when writing data on the Cray XT-5 at ORNL. This level of performance allows researchers to write metadata-rich data, in the ADIOS-BP format.

Our reading performance of ADIOS-BP files on the Cray XT-4 at ORNL has also been able to get excellent performance. In Figure 3, we read in 62 GB data from BP files written from a GTC run on 32K cores. We see that it takes approximately 2 seconds to read in this file from a reasonable number of cores.

Since ADIOS is an I/O componentization that allows users to switch the I/O methods during runtime, we argue that combining this with provenance capturing methods, in situ visualization methods, and code coupling methods, ADIOS can be used as a general framework to couple fusion codes together.

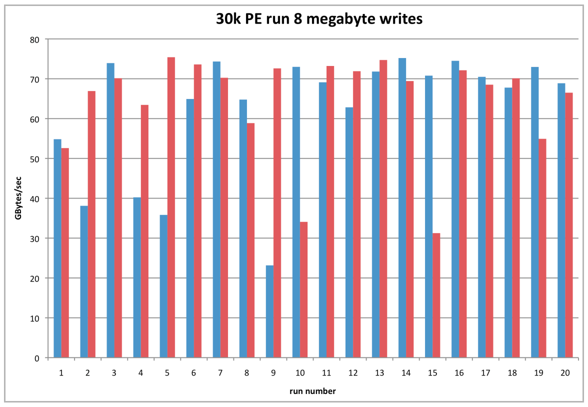

Figure 2. ADIOS write performance. Two I/O methods are shown here with 20 GTC test runs each.

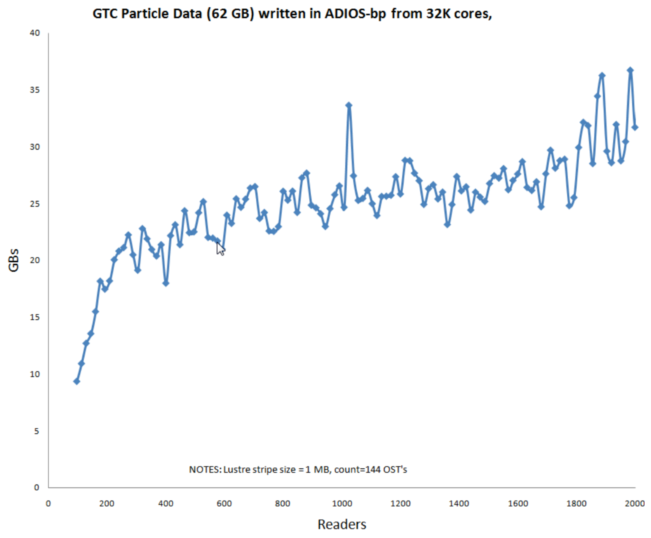

Figure 3. ADIOS read performance for GTC data. Using enough processors to read, over 30GB/sec data rate can be achieved.

3.2 DataSpaces: memory-to-memory data exchange

DataSpaces is an advanced coordination and interaction framework to provide the abstractions and mechanisms to support flexible and dynamic inter-application collaboration at runtime. It builds on ADIOS, specifically the *DART* [19] asynchronous data transport method provided by ADIOS. DART uses RDMA (Remote Direct Memory Access) provided by advanced communication technologies and is optimized for fast, asynchronous data transfers with low latency and small overheads. Furthermore, it enables direct memory-to-memory communication between the nodes of distinct applications through RDMA. DART is particularly suited for high performance applications as it enables the overlap of computations and communication allowing better utilization of the computing resources.

DataSpaces provides the abstraction of a virtual semantically-specialized shared space that can be asynchronously and flexibly accessed using simple yet powerful operators (e.g., *put()* and *get()*) with appropriate selectors. These operators are agnostic of the location, source/destination, the distribution of the data, and the interacting application components. It also supports "in-the-space" manipulation and/or reduction of data using pre-defined and user-defined functions, as well as abstractions for data subscriptions and notifications. In our ADIOS implementation of DataSpaces, we use the I/O interface (write, read) to allow RDMA operations (put, get) without changing the source code. This enables users to easily switch from the DataSpace method to an I/O method such as MPI I/O.

The DataSpaces architecture is composed of three key layers: *communications layer*, *directory layer*, and the *storage layer*. The *communications layer* builds on DART and extends its communication and data transport capabilities to support control and data message exchange. It also adds support for node discovery, registration and notification.

The *directory layer* provides the coordination capabilities and tracks data sources, sinks and distributions. It enables applications to insert, query and retrieve data from peer nodes in the space. It is implemented as a semantically enhanced distributed hash table (DHT), which is dynamically distributed across the nodes that are part of the space to provide load balancing when data is inserted in the space and enables efficient look-ups when data is retrieved from the space. It supports subscribe/notification mechanisms and autonomic cleanup when data is no longer referenced.

The *storage layer* hosts the actual data being shared among the applications (*i.e.*, the objects *put* into the space). This layer implements a coherency protocol, which defines and determines the interactions of an object with the space. For example, the protocol defines and determines when an object can be inserted, updated, destroyed or removed from the space. This layer also preserves data integrity when multiple application nodes access the data simultaneously.

An initial prototype of DataSpaces has been implemented and deployed on the Jaguar Cray XT5 system at Oak Ridge National Laboratory and is being used to support coupled fusion simulations as part of the CPES project. DataSpaces provides asynchronous coupling capabilities allowing the coupled application codes to progress independently at different rates and to exchange data at runtime without making any assumptions about the frequencies of interactions or the relative execution speeds of the codes or forcing synchronizations. Initial evaluations have demonstrated the performance benefits as well as the flexibility of DataSpaces.

3.3 Staging area and in-line data processing

Asynchronous I/O combined with extra compute nodes as a staging area can further improve the performance of the I/O and decrease the latency of I/O in the application. Asynchronous data transport methods such as DataTap [17] and DART [19] for ADIOS have been tested with the GTC and XGC1. Using these methods, data from the 10k+ cores of the simulation is not streamed directly towards the file system, but to the staging nodes (see Figure 4). The processes on the staging nodes can use another ADIOS I/O method to write to a file in parallel (like the MPI-IO method to write ADIOS-BP file(s) or the pHDF5 method to write directly an HDF5 formatted file).

In [17], we extend the idea to perform data analytics in the staging nodes. Operations like data reduction (without loss of scientific validity), compression, indexing for fast data access, and lightweight diagnostic calculation for monitoring the health of the simulation can be placed inside the staging area. As Figure 1 shows, interpolation of the data from the mesh used in GTC to the mesh XGC1 is necessary to couple the two codes. The interpolation can either be executed within one of the codes as an extra step when the exchange is performed or in the staging area itself. The abstract design of Data Services in [17] allows one to put the extra manipulation process into the staging area.

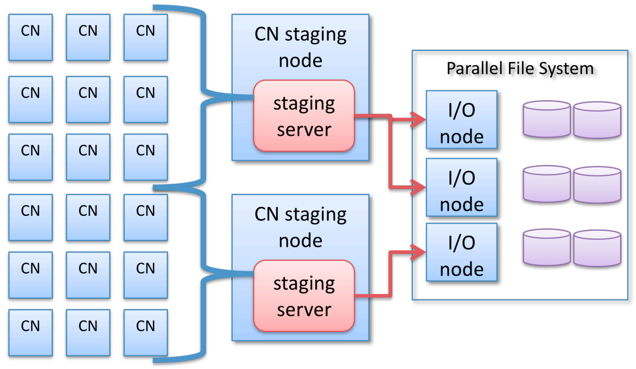

Figure 4. Using staging-nodes to gather data before writing to disk. CN denotes a compute node of the supercomputer

Another useful operation for GTC is sorting of the particle data before it is written to disk. The particle simulation code is calculating with billions of particles (ions and electrons). Each particle has an id, which can be used to trace the movement of the particle. Sorting is necessary when to visualize or analyze a subset of the data on a handful of processors. This allows reading only a small subset of particles for each timestep instead of browsing through large files searching for them. The particle count of 2-8 billion in today's GTC runs mean 64-256 GB files per timestep, and there are 100-1000 write-out timesteps.

4. COMMUNICATION OF I/O SERVICE AND THE WORKFLOW SYSTEM

4.1 Connection from ADIOS to Kepler

The activities of the I/O components can be reported to the workflow so that it can record them and make decisions on what to do. The I/O routines are part of the simulation executable and are running on computational nodes of a supercomputer. These nodes can only communicate with the outside world via the file system and the front-end nodes of the supercomputer through sockets. In our approach, we establish a two-way communication between the I/O component and the workflow. The communication goes through a front-end node of the supercomputer.

The workflow runs on a separate system and is connected to a front-end node of the supercomputer to watch the output directories of the simulation and initiate the data transfers in current workflows. Since the workflow is connected to a front-end node with an SSH connection, we implement the connection by an indirect socket connection between the simulation and the workflow utilizing the remote port forwarding mechanism provided by the SSH server, see Figure 5. We have added support for port forwarding to the org.kepler.ssh[3] package of Kepler so when an SSH session is established to a resource, a list of remote ports and local ports can be supplied to be connected through the virtual SSH tunnel. Kepler actors using this package can listen on one of the local socket ports and receive data from entities that connect to the corresponding remote port on the connected resource. All communication on that remote port is forwarded by the connected SSH server back and forth. This is physically implemented as a separate user level process on the remote machine, which forwards the communication.

Since ADIOS allows the simulation to utilize two or more transport methods for the same data, we created a separate method for the purpose of communicating with the workflow instead of modifying all of the existing methods and requiring this of any new transport method. This new method, named *adios-provenance*, reports the metadata through a socket connection. Working independently from the actual I/O method in the simulation, this method notifies a workflow of the I/O activities in the same form no matter if the other method writes the data into a file or passes it through a staging area to be consumed by a coupled code. Since the metadata, or index, of the data is small, it is gathered on one processor of the application and the connection is established only between this processor and Kepler.

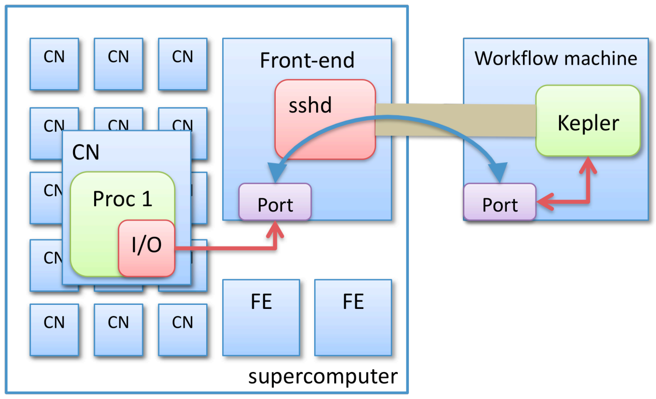

Figure 5. Socket connection from ADIOS I/O component to the remote Kepler workflow. FE denotes a front-end node.

4.2 Information exchanged between the I/O and the workflow

We are interested in the following I/O events in Kepler:

- *Write*. File "foo" is created/ updated.
- *Method*. Which ADIOS method is used for I/O.
- *Variable*. A single variable is written. It includes simple statistics of the variable.
- *Terminate*. Simulation terminated.

When the simulation writes out data of a completed calculation cycle (timestep), the workflow should react immediately and grab the data for post-processing. This means that the workflow must know when the file is actually completed writing, not just opened for writing. In the ADIOS API, an `open()` call opens the file, followed by a series of `write()` calls of individual variables that are buffered, assuming sufficient memory is available, and is terminated by a `close()` call in each timestep. The data actually starts being written to disk by ADIOS at the `close()` call so that it can achieve the best possible I/O performance by writing one large buffer to disk. The only exception to the buffering is if I/O streaming starts earlier or if the available buffer fills. The workflow is only interested in the fact that data has been written

[3] svn repository: https://code.kepler-project.org/code/kepler/trunk/modules/util/src/org/kepler/ssh

out, so the write event should be sent by the `close()` call. Note that in ADIOS the `close()` method is used at each timestep to initiate streaming the data buffer onto disk.

With memory-to-memory methods used for coupling two codes, foo is not a file on disk. If the workflow is going out to grab the data in case of a Write event, it fails to find it. Therefore it needs to know the actual I/O method used by ADIOS to decide if it can process a file or just record the provenance of the I/O activity. This information is available during the `close()` call and is sent too as a Method event.

If more information is needed about what was written, the Variable event describes the variable name, type, size, associated constant attributes and dynamically calculated characteristics, like min/max. The workflow would need this information if it were monitoring a variable and should react if the minimum or maximum value during a timestep exceeds a threshold. ADIOS knows all this information during the `write()` calls. Since the variable is generally not written directly into the file during the write() call, this information is sent from the close() call too.

The termination of the simulation is not directly recognized by ADIOS. However, it has `initialize()` and `finalize()` methods, much like parallel libraries like MPI. When `finalize()` is called, the simulation indicates that there will be no more data produced by any method. This function calls each I/O method that was used during the run to release all resources they might still use. The adios-provenance method sends this single Terminate event out on the socket connection signaling the workflow that no more events will come from this simulation.

The extra work of index interpretation and transformation to text in the I/O component itself slowing down one of the simulation processes is undesirable. In the prototype implementation, therefore, the binary form of the metadata (variable indices and data characteristics) intended to be written as a data block into a file, is sent out by the *adios-provenance* method within the simulation process. Therefore, the above events of interest are generated from the binary index data at the receiver end, i.e. in Kepler. All the Write, Method and Variable events can be generated at the same time from this data when it is received from the simulation. The Stop event is sent by the I/O layer separately in the finalize() method.

4.3 Applications of the I/O-workflow communication

4.3.1 Watching simulation output without polling

The current monitoring workflows, when watching simulations to grab newly generated data, are doing intrusive polling for new files by regularly listing the simulation directory with the SSH Directory Listing actor [10]. On parallel file-systems, the "ls -l" command to list the file names, sizes and date information is an expensive operation that can slow down simulations by accessing the metadata server and individual storage nodes of the file system to gather all information for the potentially large number of files in that directory. This can negatively impact all of the simulations running on the supercomputer when the workflow is running. By delivering the simulation I/O events to the workflow directly when they occur, the workflow can react faster and watch the simulation without a single query on the file system. The only information needed from the metadata is the name of the file(s) created or updated by the simulation (through the *Write* events). Similarly to the listing actor, the downstream pipeline can be fed with the names of files. Thus, only the listing actor is replaced with the new actor in the existing workflows to accomplish the polling free monitoring.

4.3.2 Provenance recording of variables not saved in files

One consequence of in-memory coupling of different codes is that the information exchange is not recorded in files. This prevents an external observer from seeing what is happening or performing additional operations on data, if needed. Using ADIOS in the coupling communications, simple characteristics (dimensions, max, min) of the variables being passed between the coupled codes are sent to the workflow engine. The workflow watches the characteristics along with the communication activities of the coupled codes through the *Variable* events so that certain events can be triggered.

The basic action for the workflow is to make a visualization of the values it receives from the simulation. If, *phirms*, the root mean square of the phi variable is watched to make decisions in the workflow, it generates a "phi root mean square" plot on the dashboard like the one in Figure 6. The data analysis scenario of Section 2.1 then applies to this plot too with the exception that there is no file behind the plot to be found by the provenance framework. Later, the scientist may want to select data of the *phi* variable saved in files for the timesteps when the value of phirms was a certain value and run analysis on that subset of phi data. Provenance recording of the workflow activities helps performing this selection automatically. The relationship between the phirms values and the plot is given by the data lineage. The provenance framework records the input token of the plotting actor, which contains the phirms value as well as the output token, which contains the image file name. The provenance database must be queried for all executions of the plotting actor to gather the input values to those actor executions, getting the phirms values over the whole time series.

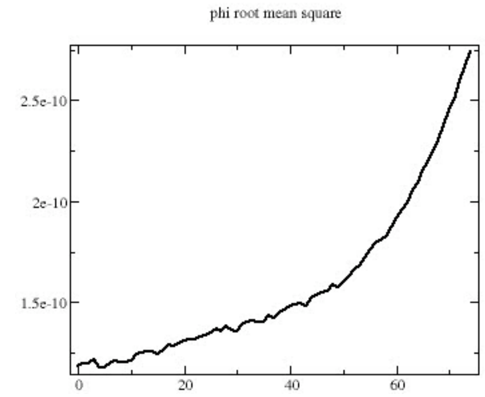

Figure 6. The root mean square of the phi variable in GTC is a plot of single values over time (i.e., an image file). Can you answer the question "*In what timesteps was the value greater than M?*" programmatically?

4.3.3 XGC-M3D coupling with stability checking

The CPES Full-ELM coupling application [11] enables physicists to study the dynamic interaction of kinetic effects that cause a buildup of the edge pedestal in plasma density and temperature profiles and large bootstrap currents with Edge Localized Modes (ELMs) that may limit pedestal growth and tokamak reactor performance. Three different fusion codes are used in this coupled simulation as shown in Figure 7. XGC0 [3] reads in at the beginning a so-called equilibrium data file (g-eqdsk) related to the simulated fusion device and the physical experiment of interest (analytic profiles for the edge plasma density, temperature and magnetic equilibrium data). A shared-memory version of the M3D code [20], M3D-OMP is used to update this equilibrium data during the simulation, using the XGC0 status stored in a file named m3d.in. The refined g-eqdsk data is then read in by XGC0 to maintain self-consistency. To check the linear MHD stability of XGC0, a third code, ELITE [21] is executed in a parameter sweep. The edge plasma density data (p-eqdsk) from XGC0 is needed in addition to the g-eqdsk data for an ELITE run. When some ELITE steps in the parameter sweep finds XGC0 unstable, the XGC0 kinetic model must be stopped and a nonlinear simulation of the ELM using the parallel M3D-MPP code launched. This will eventually recover a modified equilibrium from this simulation that can be used to begin a new XGC0 run that will start the cycle over again.

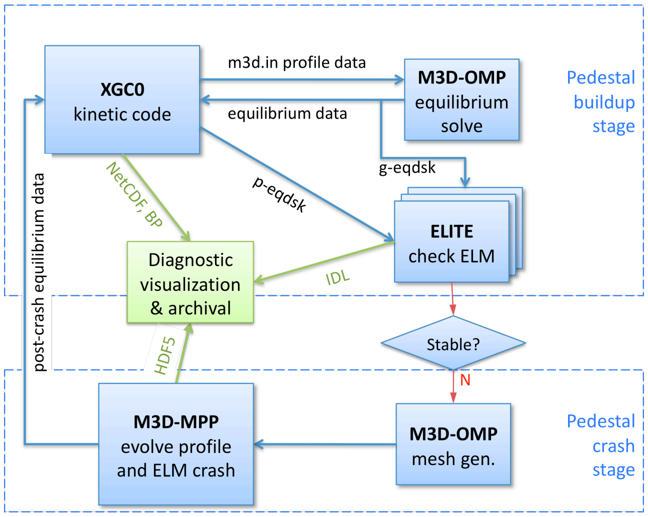

Figure 7. Schematic of full-ELM coupling

In the first implementation of the coupling, presented in [11], a Kepler workflow orchestrates all code executions where the data exchange is performed via files (see Figure 8). The workflow discovers whenever XGC0 creates a new output (m3d.in and p-eqdsk), and then it submits an M3D-OMP job with the current m3d.in and g-eqdsk data. When M3D-OMP is finished, the workflow prepares a parameter sweep of ELITE jobs for which it copies the p-eqdsk data from XGC0 and the corresponding g-eqdsk data produced by M3D-OMP. The workflow also transfers the updated g-edqsk file back to XGC0. When the code becomes unstable, the workflow kills XGC0 and submits an M3D-MPP job after interpolating XGC0 data for its input using M3D-OMP. Besides the coupling orchestration, the workflow continuously monitors the codes for their output and creates images from them. The codes happen to use different file formats (NetCDF, HDF5, ADIOS BP format and text files) and the workflow is using different tools (xmgrace, AVS, gnuplot and IDL) to produce the visualizations shown in Figure 9. These images, updated as new data becomes available, show the status and progress of the coupled simulation.

In the second implementation, presented in [22], XGC0 and M3D-OMP are tightly coupled via memory-to-memory communication using ADIOS and DataSpaces. Data once written into m3d.in and g-eqdsk files are now exchanged via I/O services as illustrated in Figure 10. The implementation consists of a *client component* that is integrated with the two application codes and allows for dynamic data exchange at run time, and a *space component* that runs on a cloud of "staging nodes", which are dedicated for the space and independent of the application nodes. The DataSpaces client on the XGC0 application inserts plasma profile data objects in the shared space and the DataSpaces client on the M3D-OMP application extracts and passes them to the application. In the next step, the DataSpaces client on the M3D code inserts MHD equilibrium data objects back in the space and the DataSpaces client on the XGC0 extracts and passes them to the application.

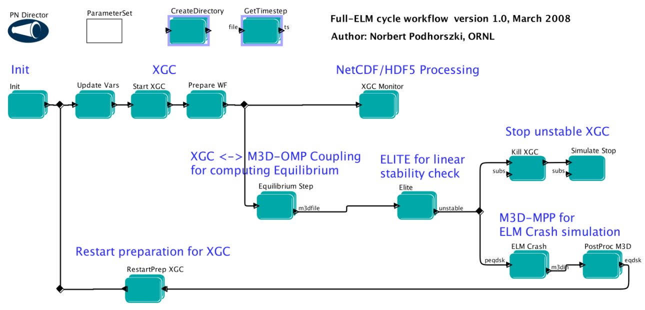

Figure 8. Kepler workflow of file-based Full-ELM coupling

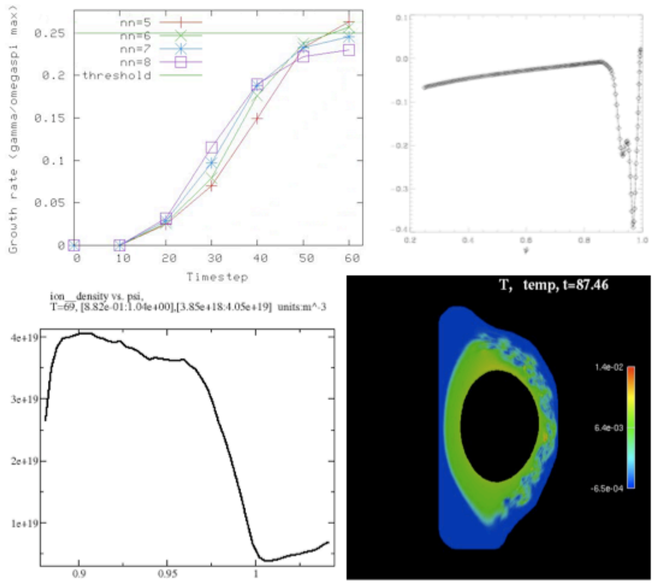

Figure 9. Images generated by the workflow from the data of the different codes participating in the coupled simulation

The tighter coupling of XGC0 and M3D-OMP enables the equilibrium update to be performed more often while the stability checking still takes the same time as with the first implementation. The frequencies of these two steps are now separated. The ELITE coupling is still performed by the workflow therefore XGC0 still generates a p-eqdsk file at regular intervals while M3D-OMP writes the g-eqdsk data to a file too besides putting it to the DataSpaces. The difference between the two workflow implementations is little compared to their complexity. The original workflow watches for m3d.in files (i.e., watches XGC) and executes M3D-OMP as a job to get the g-eqdsk file and then it takes the corresponding p-eqdsk file to run the ELITE jobs. The second workflow watches for the g-eqdsk files to appear on disk (i.e. watches M3D-OMP). However, this workflow could be implemented very similar to the first one. It would watch for the event that corresponds to the writing of the m3d.in to a file in the first implementation, which is the Write event of m3d.in from XGC. On such event, instead of executing M3D-OMP, it would just wait for the Write event of g-eqdsk file from M3D-OMP. Besides this, the workflows are identical in all steps.

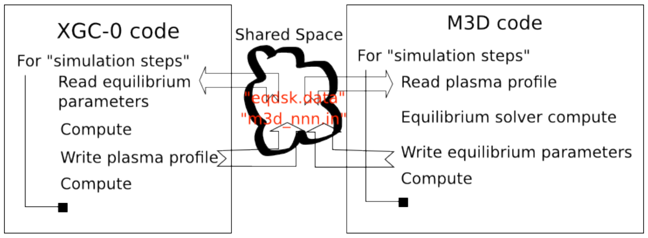

Figure 10. XG0-M3D coupling using DataSpaces framework

The Method event allows a workflow to know which transport method was actually used, i.e. to know if XGC is coupled to M3D-OMP memory-to-memory or the workflow is expected to run M3D-OMP. We can thus build a single coupling workflow, where a logical branch, see Figure 11, within the "M3D-OMP" step handles the different workflow behaviors in the two coupling cases but otherwise everything is the same as in the original workflow.

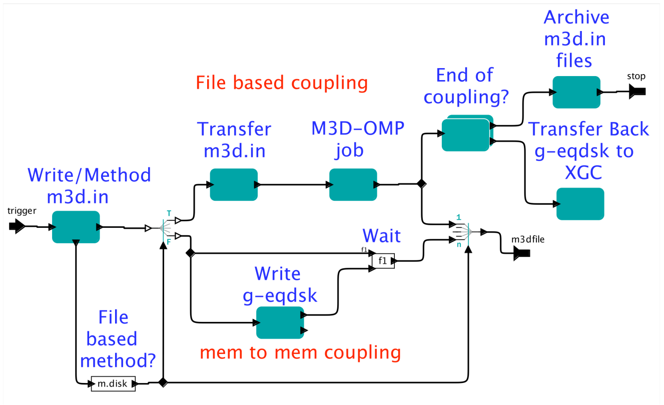

Figure 11. M3D-OMP step (composite actor in Kepler) in the combined workflow. Either M3D-OMP is executed with files as input, or the workflow waits until the tightly coupled M3D-OMP signals the completion with a "Write g-eqdsk" event.

Since ADIOS is designed to allow changing the I/O method during the simulation, we can enable the workflow to talk directly to ADIOS to switch the I/O method. These techniques together enable us to design coupling scenarios where the workflow can turn on and off a tight coupling and replace it with a slower, file-based coupling, but perform additional operations if some conditions are observed.

5. CONCLUSION AND FUTURE WORK

This paper presents our vision of coupling two large fusion simulation codes with memory-to-memory coupling while monitoring the data exchange in a workflow by connecting the code(s) to the workflow via the I/O library used in the simulation to exchange data. The workflow can change from memory-to-memory coupling to a file-based coupling and back to perform additional analysis on the data, if needed. We use the ADIOS componentized I/O library for the communication, which allows for using multiple I/O methods behind a single I/O function call in the simulation. A new method, adios-provenance was created to send metadata about the exchanged data to the Kepler workflow. The connection is currently implemented by an indirect socket connection from the Kepler workflow engine to one of the processes of the simulation utilizing the port forwarding mechanism of ssh servers. The gathering of the metadata allows us to produce image plots on observed variables without them being written to files; to build a workflow that orchestrates and monitors a coupling application with different coupling methods; even change from memory-to-memory coupling to a file-based coupling during a coupled run so that additional analysis can be performed on the data being exchanged; or simply just watch for the output data of a simulation for post-processing without polling the output directory of the simulation.

There are some disadvantages of the direct connection of the simulation and the workflow. In our framework, simulations are submitted by the coupling workflow. The workflow establishes the connection to the supercomputer front-end before the simulation starts so the port information can be a parameter for the simulation. However, in the case of monitoring workflows, the workflow is started later to monitor an independently submitted simulation so the I/O service regularly needs to perform a discovery procedure to find the port as long as the workflow establishes the SSH connection to the front-end node and creates the forwarded port. Events occurring before the connection is established, or between a disconnection and reconnection in case of a failure, should be temporarily stored somewhere. Another disadvantage of our prototype is that the binary form of the metadata emitted by the ADIOS method within the simulation process should be processed in the Java environment of Kepler. A more general framework can be designed so that events of interest are generated from the index binary data and put into a database via some kind of service accessible from the simulation. Then Kepler or some other system could subscribe to the service to receive events from the observed simulation. However, since we want control from the workflow to possibly change the I/O method in the simulation on the fly, we need to design a decoupled way to access the simulation's I/O layer from the workflow, which is straightforward with the direct connection we have now.

6. ACKNOWLEDGMENTS

This work is part of the ongoing research activities within the Center for Plasma Edge Simulation (CPES), a SciDAC Fusion Simulation Prototype center that is supported by the Office of Fusion Energy Sciences and the Office of Advanced Scientific Computing Research within the US Department of Energy. We are grateful to the National Center for Computational Science at Oak Ridge National Laboratory for access to and support of their computing resources. We thank to the Scientific Data Management (SDM) Center, another SciDAC center, which supports the development of Kepler and developed the Kepler Provenance Framework. We thank Zhihong Lin and Yong Xiao, in the GPSC and GSEP projects, and Michael Booth from Sun Microsystems.

7. REFERENCES

[1] C.S.Chang et al. "Toward a first-principles integrated simulation of tokamak edge plasmas", Journal of Physics: Conf. Ser. 125 012042 (7pp)

[2] Z. Lin, S. Ethier, T.S. Hahm and W.M. Tang, "Size Scaling of Turbulent Transport in Magnetically Confined Plasmas", Phys. Rev. Lett. vol. 88, no. 19, p. 195004, Apr. 2002

[3] C. S. Chang, S. Ku and H. Weitzner, "Numerical study of neoclassical plasma pedestal in a tokamak geometry", Phys. Plasmas, vol. 11, p. 2649, May 2004.

[4] L.Zhang, M.Parashar, "A Dynamic Geometry-based Shared Space Interaction Framework for Parallel Scientific Applications", Proceedings of the 11th Annual International Conference on High Performance Computing (HiPC 2004) , Bangalore, India, December 2004

[5] R. Barreto, S. Klasky, N. Podhorszki, P. Mouallem and M. Vouk, "Collaboration Portal for Petascale Simulations". 2009 International Symposium on Collaborative Technologies and Systems, (CTS 2009), pp. 384-393, Baltimore, Maryland, USA, May 2009.

[6] S. Klasky et al., "Adaptive IO System", Proceedings of the 50th Cray User Group meeting (CUG 2008), Helsinki, Finland, May 2008.

[7] 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 & Experience, vol. 18(10), p. 1039, August 2006.

[8] L. A. Drummond, J. Demmel, C. R. Mechoso, H. Robinson and K. Sklower and J. A. Spahr. "A Data Broker for Distributed Computing Environments". In Lecture Notes in Computer Science. Volume 2073/2001, pp. 31-40, January 2001.

[9] J. Cummings et al. "EFFIS: an End-to-end Framework for Fusion Integrated Simulation", submitted to PDP 2010, Pisa, Italy, February 2010

[10] N. Podhorszki, B. Ludäscher, S. Klasky, "Workflow Automation for Processing Plasma Fusion Simulation Data", 2nd Workshop on Workflows in Support of Large-Scale Science (WORKS'07), June 25, 2007, Monterey, California, U.S.A.

[11] J. Cummings et al. "Plasma edge kinetic-MHD modeling in tokamaks using Kepler workflow for code coupling, data management and visualization". Communications in Computational Physics, 4 (2008), pp. 675-702.

[12] B. Ludäscher, I. Altintas, S. Bowers, J. Cummings, T. Critchlow, E. Deelman, D. D. Roure, J. Freire, C. Goble, M. Jones, S. Klasky, T. McPhillips, N. Podhorszki, C. Silva, I. Taylor, and M. Vouk. "Scientific Process Automation and Workflow Management". In A. Shoshani and D. Rotem, editors, Scientific Data Management: Challenges, Existing Technology, and Deployment, Computational Science Series, chapter 13. Chapman & Hall/CRC, 2009.

[13] Ilkay Altintas et al, "Provenance in Kepler-based Scientific Workflow Systems," Poster # 41, at Microsoft eScience Workshop Friday Center, University of North Carolina, Chapel Hill, NC, October 13 - 15, 2007, pp. 82

[14] P. Mouallem, M. Vouk, S. Klasky, N. Podhorszki and R. Barreto, "Tracking Files Using the Kepler Provenance Framework". 21st Intl. Conf. on Scientific and Statistical Database Management, SSDBM'09, LNCS 5566, pp. 273-282, New Orleans, LA, USA, June 2009

[15] J. Lofstead, S. Klasky, M. Booth, H. Abbasi, F. Zheng, M. Wolf and K. Schwan, "Petascale IO using the Adaptable IO System", Proceedings of the 51st Cray User Group meeting (CUG 2009), Atlanta, GA, May 2009.

[16] J. Lofstead, F. Zheng, S. Klasky and K. Schwan, "Adaptable, Metadata Rich IO Methods for Portable High Performance IO", Proceedings of the 23rd IEEE International Parallel & Distributed Processing Symposium (IPDPS 2009), Rome, Italy, May 2009.

[17] H. Abbasi, J. Lofstead, F. Zheng, S. Klasky, K. Schwan and M. Wolf, "Extending I/O through High Performance Data Services", to appear at Cluster Computing 2009, New Orleans, LA, August 2009.

[18] J. Lofstead, S. Klasky, K. Schwan, N. Podhorszki and C. Jin, "Flexible IO and Integration for Scientific Codes through the Adaptable IO System (ADIOS)", Proceedings of the 6th ACM/IEEE International Workshop on Challenges of Large Applications in Distributed Environments (CLADE 2008), Boston, MA, June 2008.

[19] C. Docan, M. Parashar and S. Klasky, "DART: A Substrate for High Speed Asynchronous Data IO", In Proceedings of High Performance and Distributed Computing (HPDC'08), June 23-27, 2008, Boston, Massachusetts, USA.

[20] W. Park, E. V. Belova, G. Y. Fu, X. Z. Tang, H. R. Strauss and L. E. Sugiyama, "Plasma simulation studies using multilevel physics models", Phys. Plasmas, vol. 6, p. 1796, May 1999

[21] H. R. Wilson, P. B. Snyder, G. T. A. Huysmans and R. L. Miller, "Numerical studies of edge localized instabilities in tokamaks", Phys. Plasmas, vol. 9, p. 1277, April 2002.

[22] C. Docan et al. "Experiments with Memory-to-Memory Coupling on Fusion Simulations", submitted for 5th Intl. Conf on e-Science, Oxford, UK, Dec. 2009