International Journal of
HIGH PERFORMANCE
COMPUTING APPLICATIONS

Special Issue Paper

The International Journal of High Performance Computing Applications
1–23
© The Author(s) 2021
Article reuse guidelines:
sagepub.com/journals-permissions
DOI: 10.1177/10943420211019119
journals.sagepub.com/home/hpc
SAGE

The Exascale Framework for High Fidelity coupled Simulations (EFFIS): Enabling whole device modeling in fusion science

Eric Suchyta[1], Scott Klasky[1], Norbert Podhorszki[1], Matthew Wolf[1], Abolaji Adesoji[2], CS Chang[3], Jong Choi[1], Philip E Davis[4], Julien Dominski[3], Stéphane Ethier[3], Ian Foster[5], Kai Germaschewski[6], Berk Geveci[7], Chris Harris[7], Kevin A Huck[8], Qing Liu[9], Jeremy Logan[1], Kshitij Mehta[1], Gabriele Merlo[10], Shirley V Moore[11], Todd Munson[5], Manish Parashar[12], David Pugmire[1], Mark S Shephard[2], Cameron W Smith[2], Pradeep Subedi[4], Lipeng Wan[1], Ruonan Wang[1] and Shuangxi Zhang[2]

Abstract
We present the Exascale Framework for High Fidelity coupled Simulations (EFFIS), a workflow and code coupling framework developed as part of the Whole Device Modeling Application (WDMApp) in the Exascale Computing Project. EFFIS consists of a library, command line utilities, and a collection of run-time daemons. Together, these software products enable users to easily compose and execute workflows that include: strong or weak coupling, in situ (or offline) analysis/visualization/monitoring, command-and-control actions, remote dashboard integration, and more. We describe WDMApp physics coupling cases and computer science requirements that motivate the design of the EFFIS framework. Furthermore, we explain the essential enabling technology that EFFIS leverages: ADIOS for performant data movement, PerfStubs/TAU for performance monitoring, and an advanced COUPLER for transforming coupling data from its native format to the representation needed by another application. Finally, we demonstrate EFFIS using coupled multi-simulation WDMApp workflows and exemplify how the framework supports the project's needs. We show that EFFIS and its associated services for data movement, visualization, and performance collection does not introduce appreciable overhead to the WDMApp workflow and that the resource-dominant application's idle time while waiting for data is minimal.

Keywords
Workflows, code coupling, ECP, in situ, whole device modeling

[1] Oak Ridge National Laboratory, Oak Ridge, TN, USA
[2] Rensselaer Polytechnic Institute, Troy, NY, USA
[3] Princeton Plasma Physics Laboratory, Princeton, NJ, USA
[4] Department of Computer Science, The Rutgers Discovery Informatics Institute, Rutgers University, New Brunswick, NJ, USA
[5] Argonne National Laboratory, Lemont, IL, USA
[6] Department of Physics and Astronomy, University of New Hampshire, Durham, NH, USA
[7] Kitware, Inc., Clifton Park, NY, USA
[8] Oregon Advanced Computing Institute for Science and Society, University of Oregon, Eugene, OR, USA
[9] New Jersey Institute of Technology, Newark, NJ, USA
[10] Oden Institute for Computational Engineering and Sciences, University of Texas at Austin, Austin, TX, USA
[11] University of Texas, El Paso, TX, USA
[12] Scientific Computing and Imaging Institute, University of Utah, Salt Lake City, UT, USA

Corresponding author:
Eric Suchyta, Oak Ridge National Laboratory, Oak Ridge, TN 37831-2008, USA.
Email: suchytaed@ornl.gov

1. Introduction

The development of a whole device model (WDM) for a fusion reactor is critical for the science of magnetically confined fusion plasmas. In the next decade, ITER will realize plasmas that are well beyond the physics regimes accessible in any previous experiments (Aymar et al., 2002), and as summarized in the Report of the Workshop on Integrated Simulations for Magnetic Energy Fusion Sciences, a WDM is a high-priority need for “assessments of reactor performance in order to minimize risk and qualify operating scenarios for next-step burning plasma experiments” (Bonoli et al., 2015). For example, the range of dimensionless physical parameters (determining the rich spectrum of kinetic micro-instabilities) that impose powerful constraints on the quality of energy confinement in a tokamak, and the avalanche of runaway electrons that could be generated in the event of a major disruption in ITER plasmas, will both be in a plasma physics regime inaccessible to all extant experiments. Yet these phenomena are critical to understand, as experiments performed based on erroneous understanding may result in catastrophic failure. Theory and computer simulations must be reliably predictive.

Whole device modeling entails many-physics simulation and coupling. Different models and approximations are appropriate for the edge and core of the plasma. Energy transport occurs on time scales orders of magnitude longer than those needed to simulate micro-turbulence. Boundary and material interactions with the plasma must also be taken into account. A complete WDM must couple all components needed to reliably study ITER or other next-generation devices. Building a WDM is thus an example par excellence of system-level science (Foster and Kesselman, 2006).

The Whole Device Model Application[1] (WDMApp) in the U.S. Department of Energy’s (DOE) Exascale Computing Project[2] (ECP) is developing a high-fidelity model of magnetically confined fusion plasmas, by means visually summarized in Figure 1. The initial-phase objective (in black/yellow text) is first-principles strong coupling of core and edge physics. The 10 year goal then includes (in navy text) the coupling of plasma-material interaction (PMI), radio frequency (RF), neutral beam heating, extended magneto hydrodynamics (MHD), and energetic particles, as well as telescoping to transport time scales. The full undertaking requires a collaboration of application physicists, applied mathematicians, and computer scientists to solve an array of scientific, computational, and technical challenges. Successful achievement of the project’s 10-year target will enable the development of scenarios for successful and stable operation of ITER and other future fusion reactors.

There are three major software activities in the WDMApp project: the development of the physics codes to simulate the core and edge of a fusion reactor, the development of a coupling workflow framework to launch and

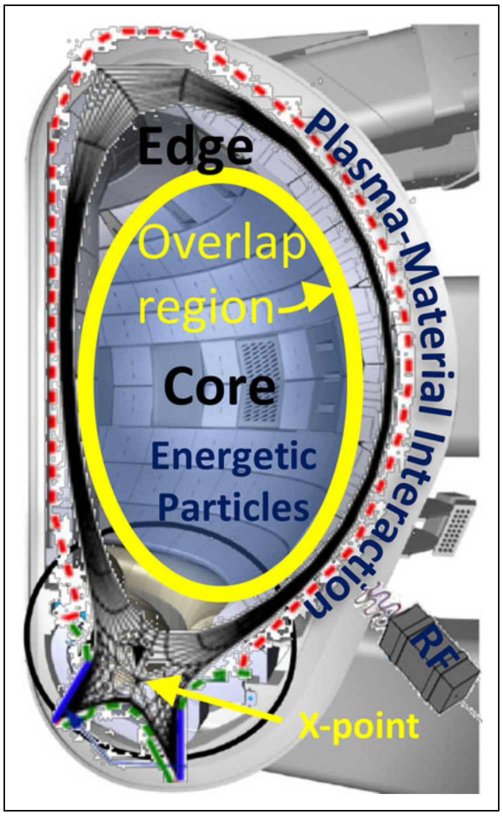

Figure 1. Cross section of an ITER plasma. WDMApp initially integrates models of the Core and Edge regions, in black with overlap region in yellow; it will later integrate plasma-material interactions, energetic particles, and RF heating.

couple the codes, and the optimization of the coupled code on the next generation exascale platforms. Altogether, the project is creating a community framework which will be used to couple together many state-of-the-art fusion codes and allow scientists to run these simulations together to solve the complex physics occurring on all different spatial/temporal scales during a fusion experiment. In this paper, our discussions focus on the software infrastructure related to coupling framework.

We describe the Exascale Framework For Integrated Simulations (EFFIS), as well as the associated enabling technologies and services used by the framework to facilitate WDMApp’s simulations; Figure 2 is a visual summary. The purpose of EFFIS is to allow users to easily compose their coupled applications, including analysis, visualization, and command-and-control, then execute the enhanced workflows on HPC or smaller machines. The project requires solutions to numerous co-design questions, relating in particular to application coupling and online data analysis and reduction (Foster et al., 2020).

EFFIS also contains useful capabilities for non-coupled simulations, such as generating/monitoring results that are delivered to a collaborative web dashboard. The essential enabling technologies for the framework include: ADIOS (Godoy et al., 2020; Liu et al., 2014), a high performance publish/subscribe data mover; PerfStubs,[3] a performance

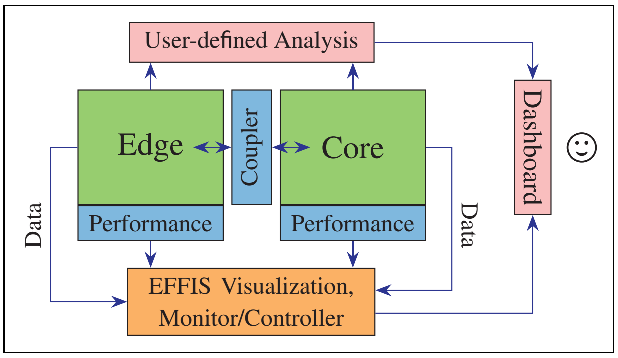

Figure 2. Schematic summarizing EFFIS usage in WDMApp workflows. EFFIS services are shown in pink, EFFIS technology in orange, and enabling technology in blue.

monitoring system based on TAU (Shende and Malony, 2006); and an advanced "coupler" that receives geometry and data from the codes to interpolate/convert data from one code's representation to another.

These design features and technology choices have been informed by three metrics of success defined for the framework: (1) assist the project to achieve its performance target of 50 times faster than the original XGC code running on the Titan supercomputer,[4] (2) allow codes to be coupled easily, and (3) allow new analysis and visualization services to be seamlessly integrated for in situ processing.

The remainder of this paper is as follows. The next section details the physics application codes to be coupled, and the coupling methods that the WDMApp project will use to achieve its physics and performance goals. Then, we review the computer science background to provide context for the different types of workflow systems that have been used by the community, in addition to the requirements that need to be satisfied to meet this project's physics and performance challenges. Next, we give an overview of EFFIS, describing its composition and execution engine, its command-and-control and automated visualization mechanisms, its enabling technologies, and the support services that it provides for user-defined analysis and dashboard integration. Evaluating our aforementioned success metrics, we demonstrate results for WDMApp workloads using the EFFIS framework, then offer concluding remarks in the final section.

2. WDM background

The two major whole device modeling initiatives in the DOE's scientific computing portfolio are ECP's WDMApp and The Advanced Tokamak Modeling Environment[5] (AToM), part of the Scientific Discovery through Advanced Computing[6] (SciDAC) program. While both model tokamak physics, there are different methods between the two. To date, WDMApp has emphasized a *first-principles*-based approach, with *strong coupling* playing a significant role, while AToM is mostly focused on *weak coupling* and employs mainly *reduced models* (Dorf et al., 2016; Meneghini et al., 2015, 2016, 2017; Park et al., 2018); reduced meaning that the gyrokinetic equations (or other essential physics) are approximated or modeled in some way to solve a simpler set of equation.

While first-principles reduce the risk of modeling error, they come at the expense of computational cost. In particular, WDMApp includes X-point-capable edge physics (cf. Figure 1), which is especially demanding computationally. Edge simulation is expected to dominate the project's computing budget. Indeed, the fact that core simulations are less computationally expensive is a primary motivating reason for using different models in the two regions.

Strong coupling has tight constraints on exchange to yield a convergent solution, while *weak coupling* has only loose, if any such constraints. Strong coupling takes place relatively more frequently than weak coupling. For instance, the first-phase WDMApp target includes core-edge strong coupling, which requires that time steps be matched to precisely the same physical time in the core and edge, with data exchange at every Runge-Kutta stage (Dominski et al., 2018). (In this case, there are four Runge-Kutta stages per time step.) One might add a weak coupling for boundary interactions in the tokamak with a period of every hundred or so time steps, which serves as a better boundary condition that is occasionally updated when/if available. Weak coupling can also be a one-directional exchange, e.g. offloading data for visualization.

The next subsection further describes the strong and weak couplings needed by WDMApp. To follow, we provide additional details regarding WDMApp's suite of strongly coupled core and edge physics codes: XGC (Ku et al., 2009, 2016), GENE (Germaschewski et al., 2020; Görler et al., 2011; Jenko et al., 2000), and GEM (Chen and Parker, 2007). These descriptions help set the computational context for the project.

2.1. Types of code coupling necessary for WDMApp

WDMApp requires several types of code coupling for the project to achieve both its short-term (ECP) and future goals. In the first stage of the project, requirements include coupling both fluid (3D) and kinetic (5D) data from the edge code (XGC) to the core code (GENE and/or GEM). The fluid-based coupling, highlighted in Dominski et al. (2018), Merlo et al. (2020), and Cheng et al. (2020) synchronously exchanges the 3D charge density (ρ) and potential fluctuations (ϕ) between the codes, which also need to be transformed (ρ^t, ϕ^t) before use in the other code; this is shown in Figure 3a. The coupling occurs at each Runge-Kutta step, along the overlap region of the codes. The kinetic coupling, highlighted in Dominski et al. (2020) and necessary for first-principles coupling, requires a buffer region whose size dictates the frequency of coupling; the larger the buffer region, the less frequently it needs to be exchanged. This is a favorable condition, because the dataset is much larger than that of the fluid coupling.

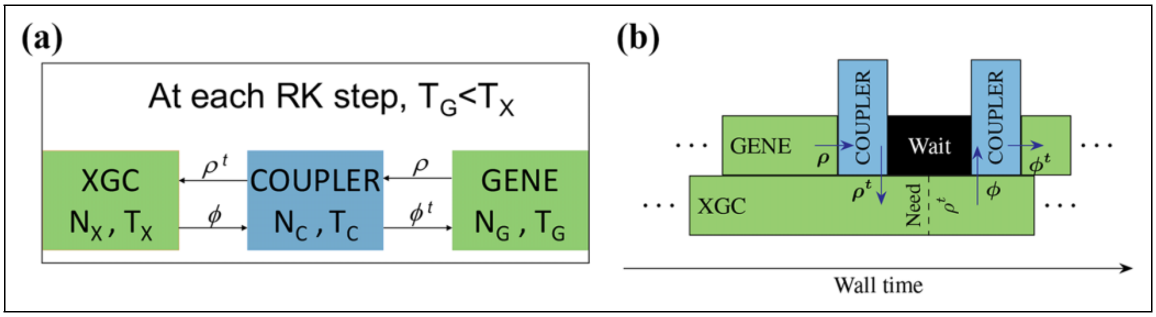

Figure 3. Summary of fluid coupling between XGC and GENE codes. The same figures also apply to XGC-GEM coupling. (a) Simplified processing decomposition. *N* values are the number of MPI ranks, and *T* values are step times (in the absence of any waiting). Charge density (ρ) and potential (ϕ) are coupled, transformed by the COUPLER for use in the other code. In general, the COUPLER can, but does not need to, be run as a third process. (b) Timeline for fluid coupling execution, highlighting one RK cycle.

Returning to Figure 3, Figure 3b illustrates the fluid coupling steps in a simple timeline. GENE is displayed as the core application, but the comments to follow apply analogously with GEM. Since the edge calculation will require many more particles per cell, it requires significantly more work (CPU-s) per time step than the core code (i.e. $N_X > N_G$ in Figure 3a). Given that the two codes will never require precisely the same amount of time (T_X vs. T_G in Figure 3a) to complete a computational time step, (in our case, XGC takes longer than GENE/GEM for each step), a goal is to minimize the resources (CPU-s) spent waiting. Thus, we engineer the workflow to minimize the wait time in XGC. In Figure 3b, this means that the COUPLER has (asynchronously) delivered ϕ^t soon enough such that XGC does not need to wait when it is ready to read it. The COUPLER can run in three modes: as a library compiled into one of the codes, as a separate code run on the same nodes as one of the codes, or as a separate code run on a different set of nodes than the two codes. Performance-wise, it is essential that the time spent moving data to the COUPLER is insignificant compared to the time spent in computation.

In the future, and currently under research and developed in the HBPS[7] SciDAC project, we will weakly couple magnetohydrodynamic (MHD) codes, such as M3D-C1 (Jardin, 2004), and boundary physics codes, such as HPIC (Khaziev and Curreli, 2018), to WDMApp. M3D-C1 coupling will enable the study of low-frequency, long-wavelength instabilities using transport coefficients that are determined from first principles rather than those determined from empirical models. Such coupling can enable predictive modeling of transient events such as tokamak disruptions or edge-localized modes or runaway electron avalanches in ITER that can be harmful for ITER operation. In HBPS, M3D-C1 is being coupled in a weak fashion where data from each code is passed after each simulation run. In the boundary physics coupling, XGC derives background plasma parameters, then HPIC provides as feedback the appropriate sheath boundary layer, impurity source terms arising from the material interaction, and modifications to the edge plasma driven by the radio frequency heating. To date, one-way exchange from XGC to HPIC has been begun. Eventually, time-domain coupling will also be included, evolving the system out to transport time scales.

Finally, coupling diagnostics (from performance monitors or analysis and visualization) is essential to allow scientists to monitor and eventually control the simulations. This is reflected in Figure 2. The data requirements can be large in certain cases, but the coupling can be asynchronous relative to the simulation. Of course, users need the capability to add their own analysis codes. Ideally a framework assists in automating these couplings, both during the simulation or directly following the simulation in post-processing.

In all cases, it is important to be able to switch transparently from writing files (which persist and can be debugged post-run) to high performance coupling methods (e.g. RDMA). These needs to switch easily between strong and weak coupling, using either synchronous or asynchronous techniques, and with either file-based or memory-based data movement, motivated our creation of the EFFIS framework. As we will describe, EFFIS provides this flexibility and allows scientists to compose, execute, monitor, and control complex workflows on exascale resources, HPC clusters, clouds, and laptops.

2.2. WDMApp simulation codes

WDMApp's first step is the spatial coupling of well established extreme-scale gyrokinetic codes: the GENE continuum code (Görler et al., 2011; Jenko et al., 2000) for the core plasma, and/or the GEM particle-in-cell (PIC) code (Chen and Parker, 2007) as an alternative core plasma code, with the XGC particle-in-cell (PIC) code (Ku et al., 2009, 2016) for the edge plasma. These three form the base model for the WDMApp project, into which other (scale-separable) physics modules will be integrated at a later time to complete the whole device model.

The fundamental kinetic equations in magnetic fusion plasma physics are derived from the fully 6D kinetic equation in phase space (that is, coordinate and velocity space) with the Fokker-Planck collision operator, coupled self-consistently to Maxwell's equations. In order to follow dynamics on time scales longer than the rapid gyromotion time scale (in the presence of a strong magnetic field), the kinetic equation can be averaged over the gyrophase to obtain 5D equations in phase space, known as the gyrokinetic equations, which are integrated by GENE, GEM, and XGC. That is, the three codes solve the same underlying equations, but through different methods. We briefly describe each code in the following paragraphs.

XGC has been developed under joint funding by the Advanced Scientific Computing Research[8] (ASCR) and Fusion Energy Sciences[9] (FES) programs through SciDAC, with the mission of addressing multi-scale edge physics on a kinetic level, including the capability to include an X-point in the tokamak geometry. The code achieves excellent weak and strong scaling on all supercomputing platforms that it has been tested on, and is currently being further optimized for Frontier[10] and Aurora,[11] the first U.S. exascale computers (Germaschewski et al., 2019). XGC is the most computationally expensive component of WDMApp; thus to minimize resource wastage, it must compute efficiently, while the core application must run quickly enough to deliver data as soon as XGC is ready for it, to not block it during the synchronous execution.

Normally, XGC simulates the entire plasma volume, in order to avoid possible error propagation from the inner boundary. However, an integrated simulation with a core-specialized code permits a more detailed description of the core at a lower cost. Core codes usually employ a delta-f scheme, which is not valid at the edge, but which has been demonstrated to provide an effective description of the core plasma. Typically delta-f methods use only $O(100)$ particles per cell, whereas full-f methods need over $O(10,000)$ particles per cell. XGC must implement a full-f scheme, needed to model strongly non-thermal plasmas in the edge of tokamaks (Ku et al., 2016).

GENE is optimized for the core plasma, out to near the last closed flux surface. It is an Eulerian gyrokinetic code that discretizes the distribution function on a fixed grid in the five-dimensional phase space by employing a unique, highly optimized combination of finite-difference, finite-volume, and spectral techniques to solve the gyrokinetic equations in the core plasma with numerical schemes also used in Computational Fluid Dynamics (CFD). To carry out a fully predictive global simulation, an outer boundary condition is needed at the edge-facing side, which can be provided via integration with XGC.

GEM is an alternate core code, similar to GENE but using the PIC approach to solve the gyrokinetic equations. Because GEM uses a PIC approach, it avoids a stringent Courant-Friedrichs-Lewy (CFL) condition in the strong coupling, and thus may prove to be easier to couple kinetically to XGC.

3. Computer science background

HPC workflow management frameworks must confront two somewhat discrepant motivations: the need for an *easy* way of composing, running, modifying, and extending workflows, simple enough for users to routinely do themselves, and the need for a *performant* framework for *extreme scale* coupled simulations. These challenges are applicable to the WDMApp project, and here we discuss the computer science context, including subsections about WDM Frameworks, workflow systems, performance, and the history/evolution of EFFIS. The approach we have selected for EFFIS attempts to achieve balance among performance, maintainability, and usability.

3.1. WDM frameworks

Multiple data and workflow management techniques are applicable in the overall WDM context, and each has its pros and cons. Previously, we mentioned the AToM project, which is creating the framework called OMFIT (Meneghini et al., 2015), with a plan to weakly couple over a dozen physics components, using a file-based code-to-code data movement approach. These weak couplings do not require any source code modifications, but typically need to modify the data from one code to another, which requires new connectors. Moreover, since the file-system of HPC machines is progressively outpaced compared to the increase of FLOPS (Foster et al., 2017), this technique is not advisable for strongly coupled codes with large data transfers.

Another approach would be to incorporate and compile all the physics into one executable. Maximal performance is obtainable this way, but the maintainability goes down, since the various physics are routinely over 100 K lines, and combining all of these into a single executable can make it challenging for developers to add new physics and optimize everything on complex resources (such as an exascale computer).

Our hybrid method is based on the principles of Service Oriented Architectures (Erl, 2005). Each application runs as a separate executable, with slight modifications to use a well-defined (universal) interface for all communication of data to other codes. A workflow manager then coordinates the placement and execution of different applications. It uses an advanced publish/subscribe system built for exascale computers, but usable on desktop systems, which has a file-like interface for self-describing I/O that can allow in-memory or file-based communication between the components (Godoy et al., 2020).

There are several reasons for the choice of methodology. First, development of each code can proceed independently, where it is easier to maintain the specializations than in one larger monolithic code base for *all* the physics. Note that they are being used stand-alone for many science runs. Furthermore, WDMApp needs to be extendable with new applications that target different physics or different scales (and may not exist yet), or simply to switch one application for another that simulates the same physics differently. It is easier to integrate new applications into an existing coupled simulation if the data exchange uses a self-describing data interface, whereby different application teams can add their new producer/consumer components independently.

3.2. Performance at scale

Beyond the basic goals of workflow composition and execution however, there is a need to ensure functionality and performance at extreme scales, on the biggest supercomputers with large amounts of simulation data. For one, placement of the multiple codes on the compute nodes can have a large impact on the overall speed of the coupled simulation. Codes run faster when placed on isolated, separate sections of the computer, but communication between the codes is faster when the corresponding processes are close to each other. Ideally, any user-directed placement strategy, such as node sharing, is described through a high level abstraction in the composition (workflow) script, and the details of the placement execution are entirely the responsibility of the workflow manager at run-time.

Since understanding and improving performance is critical in any ECP application's success, it is important to be able to visualize the performance data, in addition to the science data. For WDMApp this means capturing the performance of each code and I/O stream involved in the coupled simulation. Our framework provides plotting capabilities to generate performance graphs with EFFIS, such that the plots can be published to a dashboard alongside the visualizations of the simulation data.

A code coupling algorithm between separate code bases, with a high-level description of the data production and consumption, makes it easier to implement the coupling interface from one code to yet-to-be-known codes, but the performance of data exchange is a natural concern to developers. The core-edge coupling requires mathematically strong coupling, and therefore the performance of data exchange is critical. The success of our approach depends on the performance of the data movement infrastructure that connect the application codes (c.f. Data Movement). We use the ADIOS ECP software technology, which can achieve over 1 TB/s writing to the largest parallel file systems, and can use multiple approaches (using MPI, RDMA, RUDP, TCP) to implement in-memory coupling (Godoy et al., 2020; Wan et al., 2019).

3.3. Workflow management systems

A 2019 ASCR report highlights the potential for enabling scientific discovery from diverse data sources through in situ data management (Peterka et al., 2019). A growing need is in situ, heterogeneous coupling of multiple tasks and applications into sophisticated scientific workflows. Dynamic command-and-control capabilities are an important future requirement for these workloads (Deelman et al., 2018). Dynamic modifications may be needed for various reasons, including occurrence of certain events, efficient resource utilization, and performance optimization.

Various workflow management systems provide a subset of features required for composing and enacting sophisticated in situ workflows. It is worthwhile to consider their suitability for the needs of WDMApp. Some popular systems such as Pegasus (Deelman et al., 2015) allow constructing task pipelines through a graphical user interface, but only offer limited support for in situ composition. Fireworks (Jain et al., 2015) focuses on quality of service for long-running ensembles of tasks with recovery mechanisms, but only supports coarse-grained, loose coupling of applications. RADICAL-Pilot (Merzky et al., 2015) and Parsl (Babuji et al., 2019) provide a programmatic interface for composing workflows on heterogeneous resources, but do not focus on strong coupling/performant data movement. Frameworks such as Swift/T (Wilde et al., 2011) amount to custom scripting languages centered around task management, but this is not a formulation in which WDMApp scientists are accustomed to operating, and its bag-of-tasks model is not well-matched to the WDMApp use case.

EFFIS differs from other workflow management systems in that it provides a service oriented architecture with fine-grained support for constructing in situ workflows. In particular, it r/elies on a publish/subscribe mechanism for moving data, which allows the workflow management system to subscribe to the data streams, thereby enabling EFFIS to be used for command-and-control contingent on the data. It is a workflow coordinator and coupling framework that allows users to add, remove, and configure analysis and visualization services transparently without impacting the running simulation. Built upon the ADIOS data management library (Godoy et al., 2020) and Cheetah/Savanna in situ runtime (Mehta et al., 2019), EFFIS supports file-based, in-memory, and WAN-based coupling, with options to control data flows between components.

3.4. EFFIS 1.0

The EFFIS framework has its roots in the SciDAC program, where it was originally created by researchers from the fusion and computer science research community to loosely couple fusion codes in an easy-to-use framework (Cummings et al., 2010). EFFIS 1.0 combined the Kepler workflow engine (Altintas et al., 2004), along with ADIOS for I/O (Liu et al., 2014), eSimMon to operate as a dashboard (Tchoua et al., 2010), and a MySQL[12] database.

Even though EFFIS 1.0 was successful in allowing scientists to compose, launch, monitor, and analyze their jobs, there were several features that made it difficult to maintain. Figure 4 shows an example workflow for coupling the XGC code to the M3D code. Inside this workflow there was a control loop, where Kepler would examine the

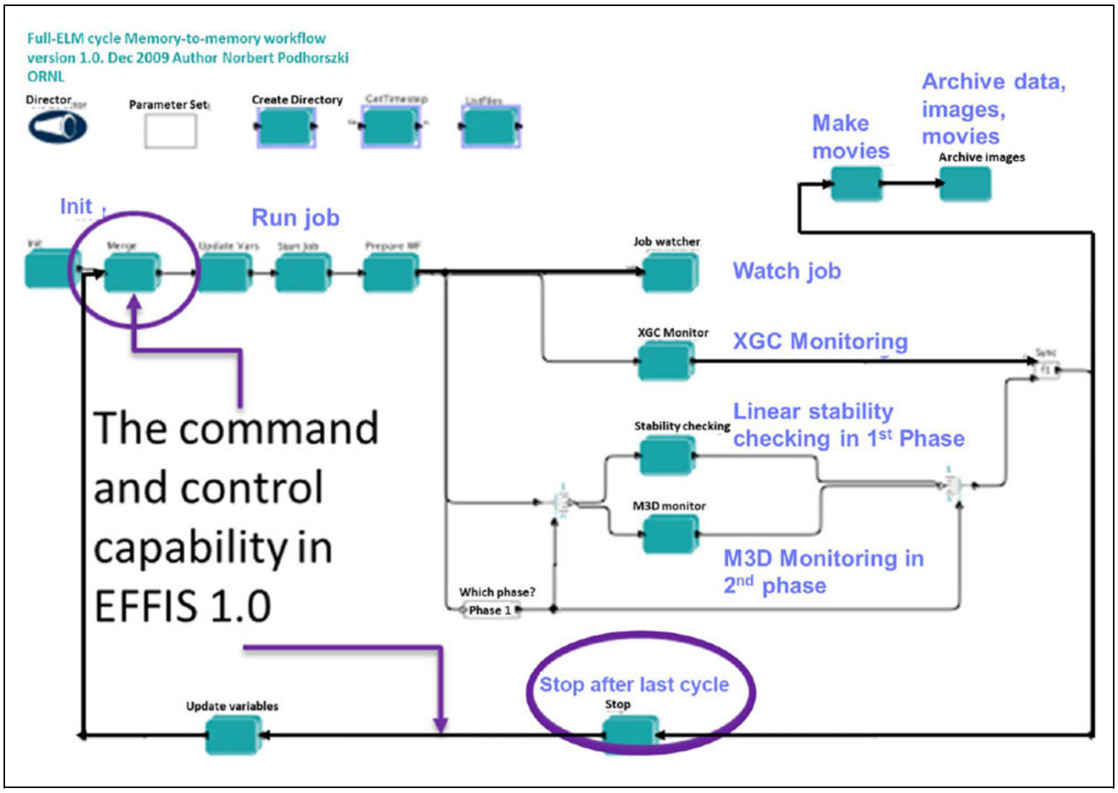
Full-ELM cycle Memory-to-memory workflow
version 1.0. Dec 2009 Author Norbert Podhorszki
ORNL
Director
Parameter Set
Create Directory
Archive data, images, movies
Make movies
Archive images
Init
Run job
Job watcher
Watch job
XGC Monitor
XGC Monitoring
Linear stability checking in 1st Phase
Stability checking
M3D monitor
M3D Monitoring in 2nd phase
Which phase?
Phase 1
The command and control capability in EFFIS 1.0
Stop after last cycle
Stop
Update variables

Figure 4. A typical XGC/M3D coupled job using EFFIS 1.0.

output of M3D, then stop XGC when a certain condition was met. The difficulty with this implementation was that when modifications needed to be made to the stopping criteria, it required several days of the team's workflow expert's time to modify the complex workflow to suit the needs of the scientist. Furthermore, the method that Kepler receives its parameters was through opening and reading an ASCII file, which becomes inadvisable at the largest scales.

Over several years, as the complexity of the computational resources grew, as well as the complexity of the physics of the coupling, EFFIS 2.0 has had to be re-imagined to help enable scientific discovery in the exascale age. For the remainder of this document, unqualified references to EFFIS refer to EFFIS 2.0, the current version of the software.

4. EFFIS description

As introduced in the preceding sections, EFFIS is a code coupling and workflow manager, whose primary workload target is concurrently running applications on large-scale platforms. While the framework has been developed as part of the WDMApp project to support its workloads, the software techniques are suitably general to be more broadly applicable. Like WDM, applications in other plasma physics contexts, seismology, molecular dynamics, computational fluid dynamics, additive manufacturing, astrophysics, and other domain sciences are deployed as part of sophisticated HPC workflows with multiply-coupled physics components and auxiliary processes. Beyond WDMApp, we currently have three initial engagements with EFFIS. First, we have begun tests with another fusion simulation code, GTC,[13] where we are using EFFIS to couple GTC with analysis and visualization services, and to enable collaborative monitoring on our dashboard. Similarly, we have worked with the radio astronomy SKA project (Wang et al., 2020), and will integrate some parts of EFFIS into their workflow. Finally, we plan to work with the PIConGPU (Bussmann et al., 2013) team to integrate data from their Photon source to their digital twin, enabling interactive, in-memory analysis and simulation steering.

As we have described, a scientific workflow often includes tasks beyond the simulation codes (cf. Figure 2). Analysis and visualization, as well as monitoring processes, are important accompanying components. Data are shared between workflow components, with in situ readers consuming data produced by the simulation(s) or other writers. EFFIS aims to facilitate the necessary data movement between the different processes in a performant way and more broadly, to ease the integration of support services into the workflow, automating actions application scientists routinely perform, such as data plotting, performance monitoring, and remote dashboard visualization. The framework is also customizable and extensible by the user. Extensions can include user-defined analysis or other run-time programs, as well as the ability to couple through different types of data movement and custom user-directed command-and-control. We have designed a state-of-the-art

COUPLER, which can subscribe to data from multiple codes, and then perform the mathematics to couple them.

In designing EFFIS, our attempt is to make it flexible, yet also easy to use. For example, we have chosen a pragma-based approach for library integration, which provides the hooks to advanced features such as command-and-control. This minimizes the application scientists' need for explicitly adding new APIs to their source code distributions and facilitates trivial EFFIS-less compilation if it is not available. Furthermore, we use a simple text-based keyword-value interface for workflow composition. The intent is such that the WDM application scientists need not learn new programming/scripting languages or workflow paradigms. EFFIS' workflow composition, submission, and execution process is the same from platform to platform. EFFIS was made to be a portable framework, which can run on all DOE exascale-track machines as well as Linux-based clusters/workstations, macOS, and Windows O/S; it allows users to execute their coupled jobs using many different job schedulers.

Synergistic use of complementary ECP software is essential to the EFFIS framework. ADIOS provides I/O middleware suitable for both the strong and weak coupling in WDMApp. The Cheetah/Savanna software from the ECP CODAR project offers a back-end to assist in job composition and execution. PerfStubs integration and TAU form a basis for run-time performance monitoring.

The remainder of this section further elaborates what makes up the EFFIS framework, explaining the finer details of what we have introduced heretofore. First, we further document EFFIS' design and implementation. Then, we consider enabling technologies, essential software that EFFIS leverages to enable its functionality. Finally, EFFIS services are discussed, characteristic of the useful add-on features that the framework can facilitate.

4.1. Design

Figure 5 illustrates EFFIS' basic structure and usage. The framework consists of a library, a set of run-time daemons, and command line tools. These components work together to enable workflow management before, during, and after production jobs. In this section, we describe the various design features of Figure 5. Roughly speaking, we proceed in an order that follows the development life cycle. The first subsection begins with application library integration through a pragma-based approach. Then we describe workflow composition formatting and the processing along the left half of Figure 5, which results in the construction the job to run. Next, run-time execution is discussed. This includes descriptions of EFFIS run-time daemons and workflow orchestration, such as plotting processes and command-and-control functionality.

4.1.1. Pragma-based source code integration. Integrating EFFIS into applications (cf. top right of Figure 5) is accomplished by introducing EFFIS-specific *pragmas* into the

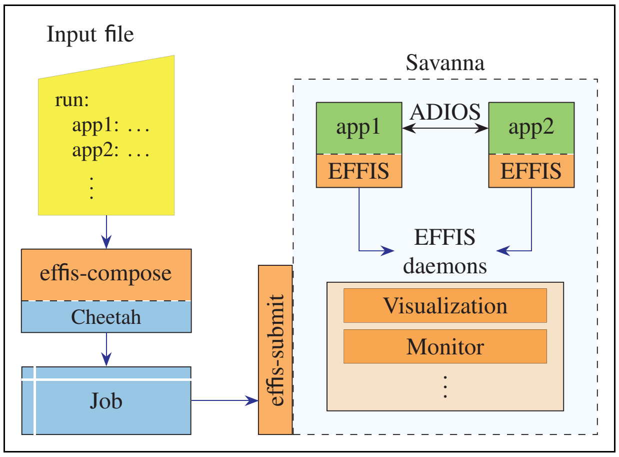

Figure 5. Schematic of EFFIS design, workflow orchestration, and enabling technology usage. EFFIS components are colored orange. The user edits an input (YAML) text file to configure how the workflow runs, then an EFFIS command line utility, built upon Cheetah's composition library, compiles the job. After job submission, EFFIS uses Savanna as the execution engine, launching the applications, along with EFFIS support daemons, which perform tasks such as plotting, dashboard coordination, and command-and-control. Data movement between applications or to EFFIS processes occurs through ADIOS.

source code. These amount to code comments, which do not affect compilation if EFFIS is not used. The EFFIS distribution includes a pre-processor (`effis-cpp`), which performs the necessary source-to-source translation to modify the existing APIs to include the EFFIS ones. Currently, Fortran, C++, and Python bindings are supported.

EFFIS APIs that have been integrated by the pre-processor present the opportunity to propagate additional control (meta)data through the workflow. For example, there is a pragma to record the physical time for a simulation time step (`@effis-timestep physical=<float time>, number=<integer step number>`). This time can then be made available to services that plot data from the simulation. The internal EFFIS APIs are also the channel through which command-and-control can operate. One needs appropriate pragmas for use in the user application wherein the EFFIS library listens for signals coming from elsewhere in the workflow.

EFFIS applications/services operate within the publish/subscribe data movement paradigm. Data producers "put" sets of variables that can then be accessed by one or more consumers in a "get." I/O pragmas can be associated with these I/O groups, which make the variables within the I/O groups accessible to EFFIS services. Included variables then might be automatically plotted and viewed on a dashboard or monitored with the possibility of generating feedback. Listing 1 and Listing 2 show pseudo-code examples of the I/O pragma markup, `@effis-begin` and `@effis-end` lines wrapping sets of ADIOS put and get APIs respectively. These examples are based on the GENE to XGC strong coupling density exchange. The name before the arrow in the pragma statements matches the name of the

Listing 1. Example EFFIS pragma usage around data put.

```
!@effis-begin 'density_coupling'->'density_coupling'
if (.not.initialized)
 call adios2_declare_io(io, ..., 'density_coupling', ...)
 call adios2_define_variable(dens_id, io, 'data', ...)
 call adios2_open(engine, io, 'density.bp', &
&adios2_mode_write, ...)
 initialized = .true.
endif
call adios2_begin_step(engine, ...)
call adios2_put(engine, dens_id, data, ...)
call adios2_end_step(engine, ...)
!@effis-end
```

Listing 2. Example EFFIS pragma usage around data get.

```
!@effis-begin 'density_coupling'->'density_coupling'
if (.not.initialized) then
 call adios2_declare_io(io, ...,'density_coupling', ...)
 call adios2_open(engine, io, 'density.bp', &
&adios2_mode_read, ...)
 initialized =.true.
endif
call adios2_begin_step(engine, ...)
call adios2_inquire_variable(varid, io, 'data', ...)
call adios2_set_selection(varid, 2, start, count, err)
call adios2_get(engine, varid, data, ...)
call adios2_end_step(engine, ...)
!@effis-end
```

Listing 3. Illustrative EFFIS workflow composition.

```
machine:
  name: summit
  charge: abc123

share: [core, edge]

run:

  core:
    executable_path: /usr/bin/gene
    processes: 64
    processes-per-node: 2

    .density-coupling:
      output_path: density.bp
      adios_engine: SST

    .field-coupling:
      reads: edge.field-coupling

  edge:
    executable_path: /usr/bin/xgc
    processes: 960
    processes-per-node: 30

    .density-coupling:
      reads: core.density-coupling

    .field-coupling:
      output_path: field.bp
      adios_engine: SST

    .diagnosis.1d
      output_path: oneddiag.bp
      adios_engine: BP4
```

declared ADIOS I/O group, which could be remapped to be referenced under a different name in the EFFIS workflow composition file, if desired (cf. Listing 3). However, we choose to maintain the same names.

We note, EFFIS itself is not the data movement library, but a thin layer atop the data movement library that enhances the library's APIs with the EFFIS services. Our current implementation expects ADIOS APIs (Godoy et al., 2020), but extensions are feasible, as long as the format is self-describing and capable of handling multiple time steps, such as HDF5 (Folk et al., 2011). In some contexts, optimized inter-computer data movers such as Globus (Chard et al., 2016) may be appropriate.

No pragma integration/translation is needed merely to compose and run a workflow with EFFIS. However, in this case the feature set will be limited to plotting (without physical time labeling), and no command-and-control or dashboard functionality is available.

4.1.2. Cheetah/Savanna as a back-end. The two sides of Figure 5 show composition on the left and execution on the right. To clarify the terminology, composition concerns the user specifying what needs to be done in the workflow, before it actually runs. EFFIS leverages the Cheetah and Savanna framework as its composition and run-time execution back-ends (Foster et al., 2017, 2020; Mehta et al., 2019): Cheetah is a tool for creating co-design campaigns using Python classes, and Savanna is an execution engine counterpart to Cheetah, which can run Cheetah-created workloads on compute nodes.

Dependencies notwithstanding, users need no knowledge of Cheetah/Savanna to use EFFIS; we have a developed a custom text-based front-end to Cheetah, with a lightweight toolset, which simplifies the required Cheetah specification for application scientists (albeit at the expense of not supporting all Cheetah co-design features, such as parameter sweeps). The `effis-compose` script ingests the configuration file and internally converts it to the Cheetah representation, then the `effis-submit` utility invokes the Savanna runtime to execute the compiled workload. This process is platform independent for users.

4.1.3. Composition. Our primary workload target is concurrently running applications. In this environment, it is more suitable to consider the workflow specification more like a multi-tier service stack rather than attempting to compose EFFIS workflows with a traditional directed acyclic graph (DAG). Our specification centers around a set of processes and data connections between them.

Workflows are composed using a text file specification in YAML format.[14] Users list the executables to run, their task decomposition (total MPI processes, number per node, cores per process, etc.), along with several other settings. For instance, composing a core-edge coupled workflow follows the skeleton pattern shown in Listing 3. The names `core` and `edge` are not special keywords, but YAML variables that are effectively references for the executable code to run, specified in `executable_path`. In this case, only two applications are run, but arbitrarily many could be included. On machines that support it, multiple applications can be configured to share the same nodes, as shown in Listing 3 with the `share` keyword.

Each code includes a section that configures I/O groups that were marked up with EFFIS pragmas, where their reference names in the YAML file are those on the right hand side of the pragma arrow (cf. Listing 1 and Listing 2). These I/O sections are denoted with a leading period/dot character (e.g., `.density-coupling`, `.field-coupling`).

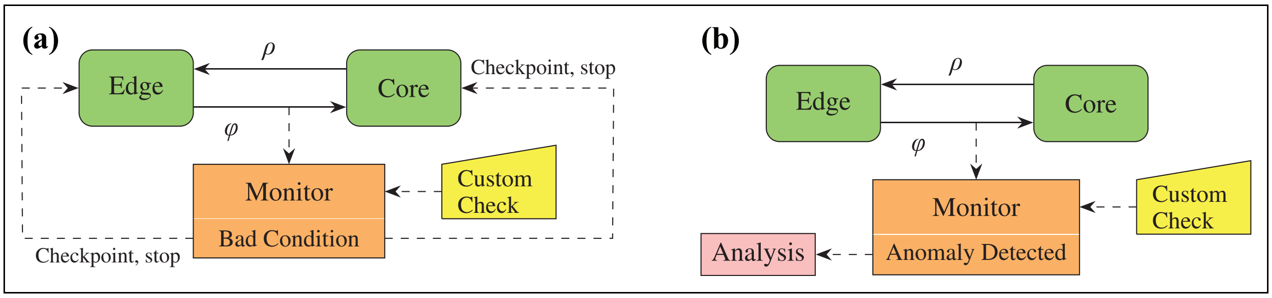

Figure 6. Examples of workflows using command-and-control in WDMApp. (a) Workflow that checkpoints and stops upon finding a bad condition. (b) Workflow with feedback to start an analysis upon anomaly detection.

Readers can be matched to a writer group, e.g. `reads: edge.field-coupling` in `core`, to appropriately configure getting data according to how it was put. In this case SST, a memory-to-memory transport in ADIOS, was used to publish `field-coupling` in the edge code.

Listing 3 is not a complete composition file. Comprehensive syntax documentation is beyond the scope of this publication; we refer readers to the online EFFIS source repository,[15] and the WDM release documentation.[16]

4.1.4. Execution. Internally, the Savanna execution engine is a Python kernel running on the login/service node that invokes subprocess calls to launch the programs run by the workflow. Savanna supports leadership computing machines, such as Summit,[17] Theta,[18] and Cori,[19] as well as commodity Linux clusters and local machines without a scheduler. Built-in scheduler support includes Slurm (Yoo et al., 2003), LSF (Zhou, 1992), Cobalt,[20] and PBS/TORQUE (Staples, 2006).

By default, tasks are launched on the compute nodes, where the bulk of the workload runs. However, we also support execution on the service node. At OLCF,[21] the HTTP-fetchable storage is not visible from the compute node network, which needs to be accessed by the dashboard preparation daemon, further discussed below. Command-and-control workflows also need to be able to access service nodes, as that is the only place where one can launch new MPI processes.

Here, we note compute resource usage for EFFIS components. Following sections will describe EFFIS daemons for automating 1D and 2D data plotting, as well as performance monitoring, Each of these three runs as a single process, but all can be run on the same compute node if the system supports it, or even a node where one or more application runs, if desired. EFFIS daemons that run on the service nodes (e.g. we will discuss ones for command-and-control and dashboard integration) do not require additional resources, as they do not run on the compute nodes. Workflow composition/submission is all on a login node.

4.1.5. Command-and-control. Figure 6 shows two examples of the command-and-control workflows that EFFIS is designed to support. When a job begins, EFFIS launches a monitoring daemon (cf. “Monitor” in Figure 5 and Figure 6), which is subscribed to some data products: the coupling data in this case. Each step, the data is read and an action is triggered if a user-supplied condition is met: checkpointing and stopping the codes in Figure 6a and starting an analysis routine in Figure 6b. For launching new processes, EFFIS allows the possibility of running daemons on the service nodes, where MPI task launching is allowed within a job on HPC resources.

Checks to be run by the monitoring daemon are given as user-supplied Python functions, and we will support issuing a limited number of commands therein: checkpointing, stopping, starting new processes, etc. This list is not finalized as the command-and-control layer in EFFIS is still under development. (Currently, a dummy implementation exists. No commands are actually issued.) EFFIS APIs will be available to the source codes to catch control signals and be suppliable with callback functions where necessary (e.g., how to checkpoint a simulation).

Issuing control signals from applications themselves is another use case possibility. For example, the edge simulation encounters an exception, branching into a checkpoint then stop code sequence, and communicates a signal to the core code to do likewise. At any rate, command-and-control requires passing relatively small messages between multiple running executables, possibly with high process counts. Thus, we are exploring the design of a control signal subsystem, apart from the usual ADIOS data movement, which is not designed for small messages.

This command-and-control functionality uses a publish/subscribe model, where processes can subscribe to *topics* that map to particular subsets of the data generated, and publishers push messages to channels that match these topics. Topics are named resources to which messages are sent by publishers. Subscribers, such as an application waiting for a notification, can specify callback handler functions, which are automatically invoked when matching data is received, and can enable data driven dynamic run-time behavior.

Specifically, the command-and-control functionality is implemented as a simple lightweight library that is persistent for the total duration of the workflow, and applications can join and leave at any point. Architecturally, it is designed as a scalable network of lightweight processes that coordinate to manage subscriptions and to match published data to these subscriptions. In order to segregate messages across different application groups that use this messaging infrastructure, applications specify a *namespace* in addition to a topic as part of their subscriptions. When a message is published to a particular namespace and topic, only applications subscribed to that namespace will be matched.

EFFIS will implement command-and-control using this publish/subscribe messaging framework by publishing control signals, which are subscribed to by relevant signal handlers that use these signals to either trigger new applications or change the behavior of that application. Further development and demonstration of the system will be explored in a subsequent publication.

4.1.6. Visualization. EFFIS includes daemons to auto-generate plots of output one- (1D) and two-dimensional (2D) variables (cf. Figure 2, Figure 5). Currently, those implemented are Python programs with Matplotlib plotting directives, but extension to other back-ends are feasible, and as described below, users can also define their own visualizations. One could invoke these daemon processes directly with the usual application formatting in the workflow composition file, but we have provided shortcuts, e.g.

Listing 4. 1D generative plotting composition.

```
run:
  plot-1D:
    x: psi
    data: edge.diagnosis.1d
```

Listing 5. Alternate generative plotting composition.

```
edge:
  .diagnosis.1d
    output_path: oneddiag.bp
    adios_engine: BP4
    plot-1D:
      x: psi
```

the default of single process. Listing 4 is a non-exhaustive, but indicative illustration of the abbreviated formatting for 1D quantities. (Additional plotting details, such as scaling and axes ranges, can be made available to the user through command line options, but are not explicitly discussed here.) The `data` keyword in the `plot-1D` section selects the relevant I/O group for the plot. Listing 5 is an alternative formatting to do the same, which we will support in future releases, where triggering the plot is directly underneath the data group itself.

Figure 7 shows examples of auto-generated plots 1D and 2D plots. In EFFIS, 1D visualization means plotting y vs. x curves, where x and y are two 1D arrays (or 1D slices of multi-dimensional arrays) of the same number of elements. Figure 7a, for example, plots the 00-moment of charge density against the radial coordinate.

Figure 7b is a 2D color map of potential fluctuations over a toroidal slice of the reactor. EFFIS 2D visualization supports image-like data, i.e. pixel values on a regular grid, and unstructured triangular mesh data, like that of XGC. Extensions to other grid types are possible; we are currently developing a more generalized approach to accommodate more visualization schemas.

Users can identify individual variables to plot or make selections of associated variables within an I/O group, all which share the same coordinates to plot against. For example, `cden00_1d` (Figure 7a) is among about 40 output quantities with the same dimensions as `psi`. All were plotted during the run that generated Figure 7, by specifying `psi` as the generating variable in Listing 4. Similarly, four other variables in addition to `dpot` (Figure 7b) were plotted, sharing the same unstructured triangular mesh.

The labels in Figure 7 include the physical simulation time (in seconds). This is an example of "extra information" that has been inherited through pragma enhancement. Nominally the `diagnosis.1d` I/O group (or analogous group for `dpot`) has no time variables, in

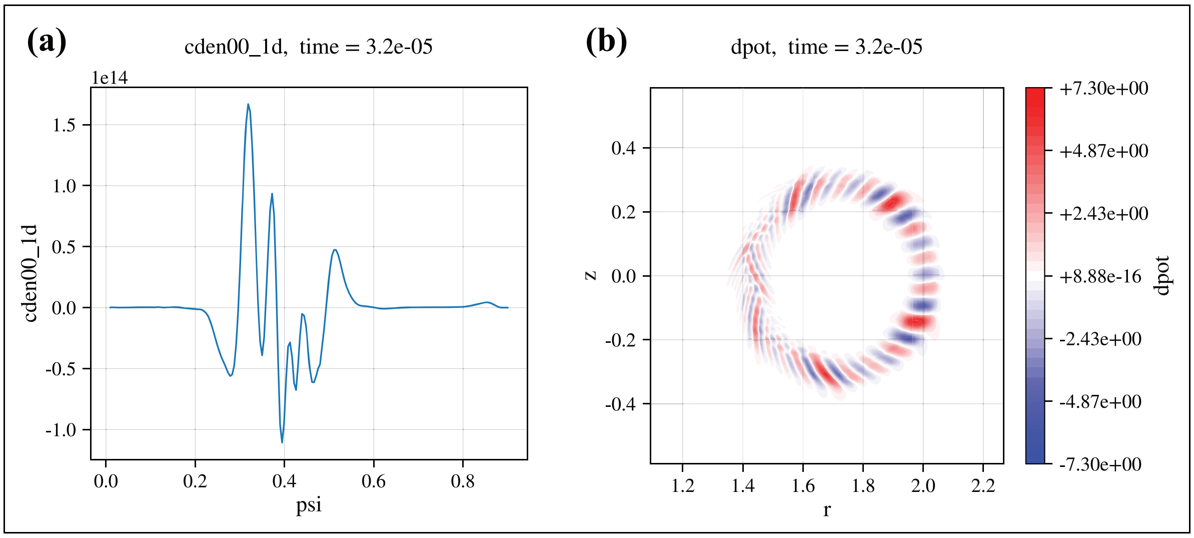

Figure 7. Example output visualizations from EFFIS plotting daemons. (a) 1D y vs. x plot. (b) 2D plot on unstructured triangular mesh.

which case the plotter would have no knowledge of the physical time. When the EFFIS pre-processor encounters the time-publishing pragma, I/O groups marked with pragmas will include the published time if they are to be plotted.

4.2. Enabling technologies

WDMApp and the EFFIS framework leverage several enabling technologies in order to achieve the project's goals. Figure 2 pictures these technologies in the overall workflow. First we discuss why ADIOS is well-suited for the data movement needs. Then, we discuss how performance monitoring software is being instrumented into the workflow. Finally, we explain our COUPLER, an intermediary for data transformations between distinct physics representations in different applications.

4.2.1. Data movement. Previously, we explained the differences between strong and weak coupling, and that WDMApp must support both. Similarly, EFFIS must accommodate both synchronous and asynchronous data movement. Synchronous coupling involves a wait barrier. The receiving process cannot continue until the data is received. Asynchronous coupling involves no such communication block. The sender off-loads its data, which may or may not be consumed. Visualization is an example candidate for asynchronous coupling, while the WDMApp fluid exchange is synchronous. Generally, strong coupling is likely synchronous, and weak coupling is likely asynchronous, but the terms are not mutually exclusive.

The EFFIS code base itself does not implement data movement, rather it relies on external libraries and services. In the current version of EFFIS, the ADIOS (Godoy et al., 2020) framework is used exclusively for several reasons. First, ADIOS provides data movement alternatives for all flavors of coupling mentioned above: strong, weak, synchronous, and asynchronous. Therefore, it is well-suited to serve WDMApp's coupling needs. Furthermore, several fusion codes (e.g. XGC, GENE, GEM) have already been using ADIOS. Thus, there are already ADIOS I/O blocks in the source code that need only some augmentation to connect into the workflow. Nevertheless, EFFIS can be extended in the future to support other libraries that may provide optimized solutions for some specific data movement scenario.

ADIOS' performant and flexible I/O has been an important enabling technology for WDMApp, which uses the following ADIOS capabilities: a) large scale storage I/O for checkpointing the entire coupled application (synchronous), b) memory-to-memory data movement for synchronous strong coupling of fusion codes implemented with one-sided MPI communication, c) memory-to-memory, asynchronous data movement for visualizing data implemented with separate threads performing the data movement and d) continuous storage I/O with asynchronous plotting of diagnostic data. Typical usage sets the ADIOS transport type to `BP4` (the name of ADIOS' file format) for diagnostic or analysis data (i.e. data that should persist after the run) and `SST` for asynchronous RDMA transport during strong coupling.

We note, ADIOS (as well as other familiar scalable I/O packages such as HDF5) runs on the CPU and that all data movement discussed in this paper is CPU data movement, though codes run through EFFIS certainly may run on GPUs. (XGC and GENE can.) But such codes' communication to the GPU devices is for all intents and purposes orthogonal to this paper's discussions. Core-edge data transfer occurs in stages where the data is on the CPU in both codes, and there is no coupling-related overhead to measure relative to the GPUs. ADIOS is researching GPU to GPU data movement in case such scenarios become necessary.

4.2.2. Performance monitoring. An important aspect of job execution is understanding performance. In the case of coupled execution, the goal is to understand the performance of not only individual components, but also interactions between components. Performance monitoring of the coupled execution must also be compatible with existing performance instrumentation used by the developers of the individual codes.

EFFIS supports the use of TAU (Shende and Malony, 2006), a comprehensive set of tools developed to measure the performance of large-scale parallel libraries and applications written in Fortran, C/C++, Python, and other languages. TAU can gather performance information through system-interrupt-based sampling and/or via instrumentation inserted in the source code automatically, provided by callbacks from libraries or interfaces (Python, OpenMP, OpenACC, MPI), or included manually by using the instrumentation API. TAU measurements represent per-OS thread measurements for all processes in a distributed (e.g., MPI) application. TAU measurements are collected as profile summaries and/or a full event trace.

For the purposes of run-time performance monitoring, low-overhead measurements are necessary so as not to distort application performance and to justify having the monitoring available during production runs. For that reason, only high-level profiling data and operating system/hardware monitoring data are collected. It is desirable to generate performance data at the granularity of important code regions, such as major functions, while avoiding excessive instrumentation of frequently called small routines. The three major physics codes used in the WDMApp project—XGC, GENE, and GEM—already contain source code instrumentation at the appropriate granularity. By augmenting or substituting the existing instrumentation with a stub interface, any performance tool that implements that interface can be used to collect performance data and output it in a flexible manner for display or other purposes such as archiving or control.

In the current EFFIS performance monitoring setup, individual codes are instrumented by using the PerfStubs interface (Boehme et al., 2019). PerfStubs is a thin stub interface for instrumenting library or application code. Function calls, which are stubs in the form of function pointers, initialized to `nullptr`, are optionally assigned at runtime by using `dlsym()` calls, as is typical with plugin implementations. If static linking is used, a weak symbol technique is used instead. If the function pointers have the value `nullptr`, then this library is a few more instructions than a no-op. If the function pointers are assigned, the measurement library functions are called. The PerfStubs API includes functions for starting and stopping timers that can have either user-defined or automatically generated names. TAU provides a PerfStubs plugin implementation. Hence, if an application that has been instrumented with PerfStubs is either linked with TAU or run using `tau_exec`, the PerfStubs calls will generate TAU profile data.

When run with `tau_exec` through EFFIS, the TAU performance data values are output as ADIOS variables for each process in the simulation. The data schema includes variables for application timers, as well as timers and counters for MPI events, CUDA events, Kokkos events, OpenACC events, hardware and operating system events, and ADIOS events. The timer variables consist of the number of calls and inclusive and exclusive times. Counter variables consist of the number of samples, minimum value, maximum value, total values, and a sum of squares for computing variance. A TAU plugin for ADIOS output is enabled that will stream the performance data out to ADIOS during pre-defined application periods, such as every N iterations of the outer loop. Thus, performance data for the entire coupled simulation is available in a unified way through use of a consistent schema.

The performance data from each component/executable of the coupled simulation is written to its own ADIOS stream, and visualized using a Python script to generate images from each period of data, analogous to 1D and 2D data plotting. Figure 8 is an example using XGC timing measurements. The currently implemented script can only visualize one stream of output per plot. However in the future, the scripts will be updated to read from multiple output streams and visualize them together as one dataset, using the periodic output intervals for synchronization between applications. Combining the performance data in this manner will make it easier to diagnose performance problems such as load imbalances, under-utilization from inefficient configurations and excessive wait times.

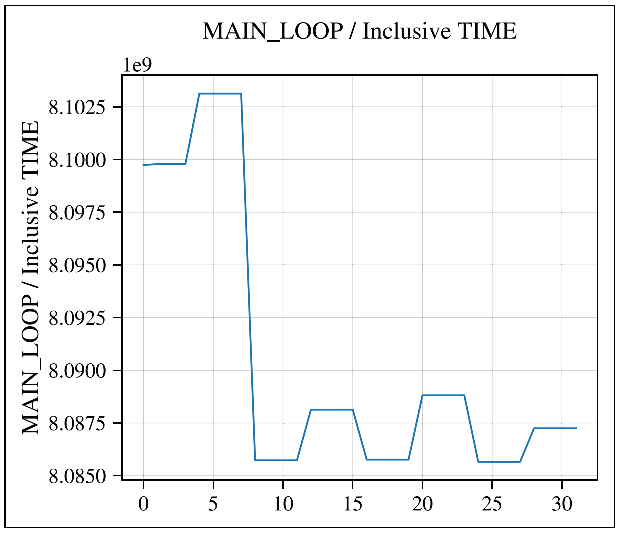

Figure 8. Example of performance measurement collected by PerfStubs/TAU and plotted automatically in EFFIS. The y-axis has units of microseconds, and each entry on the x-axis is an MPI rank. Here, the cumulative duration of XGC's main simulation loop is the quantity graphed.

4.3. COUPLER

An essential ingredient in coupling multiple physics codes for WDMApp is what we call the COUPLER (cf. Figure 3), which reads data from one or more physics codes and prepares it for use as input to another code. WDMApp presents three types of physics couplings for which we are preparing the COUPLER. The first concerns mesh data from codes with different spatial extents (e.g., core and edge), but with an overlapping region. Often such data originates from codes that use different numerical approaches and meshes, in which case the COUPLER is required to transform/interpolate the data from one code to another. The second type of coupling is between codes containing fine-grained time integrators with applications for coarse-grained time integration. The goal is to maintain the physics accuracy of the fine-grained code but over long time scales. In the third type of coupling, codes cover the same spatial region but solve for different physics. Although in the future the COUPLER will address all three scenarios, here we are concerned with coupling the core and edge codes (GENE, GEM, XGC) used in WDMApp.

In all cases, the COUPLER is built to address three objectives: *accuracy*, *performance*, and *abstraction*. Concerning the first, the coupled application must produce a consistent and accurate solution, where the accuracy can be tested from a reference solution (Dominski et al., 2018; Merlo et al., 2018). To ensure performance in an exascale environment, the time spent exchanging data between applications and the COUPLER and the time spent inside the COUPLER must both be much less than that spent in the applications. Abstraction is about separating the orthogonal concerns of data movement and physics. To this end, we create a COUPLER as a separate application with an interface that allows codes to publish and subscribe to data. Then, the COUPLER's sole concern is to receive and transform data provided by one application into a representation required by another application, via the use of methods that maintain required accuracy. We can execute the COUPLER either in the same process space as one of the applications, by using the ADIOS inline engine, which uses the COUPLER as a library in that code, or as a separate application that can be placed on the same nodes as one application or on a completely separate set of nodes.

The COUPLER is currently built to transform the fields between the XGC and GENE codes. In the future, GEM and other code options will be available to the COUPLER as well. The COUPLER contains a pre-process step, which aligns the flux tubes of GENE's mesh with those of XGC's within GENE's spatial domain. This includes setting the MPI structures used to coordinate mesh-to-mesh communication. At each step, the COUPLER carries out the coordinate transformation between GENE's field-aligned coordinates and XGC's cylindrical coordinates (for the mesh nodes located within GENE's poloidal mesh) and makes the forward and backward Fourier transformations along the binormal direction.

ADIOS allows scientists to use files or in-memory approaches, and can allow the COUPLER to live directly inside the process space of one of the codes, or in a separate process space. This allows physicists to conceive of the COUPLER as a place to conceptually write data, read and accurately transform that data, then read that data back into their other physics code(s). While it introduces an extra inter-component communication step, this use of the COUPLER avoids the complexities of modifying the physics codes to execute coupling operations and then maintaining those code portions as the applications evolve. It also enhances the ability to reuse coupling components and/or to extend them with new capabilities. In WDMApp, where the majority of the computation resources are spent in the edge, our goal is to minimize the time in the edge code waiting for data from the COUPLER.

We are also expanding the COUPLER to allow XGC to communicate to GEM, and making the COUPLER easier to extend as we add more codes. An important consideration is understanding where the COUPLER will run (e.g., on the same nodes as GENE/GEM or on different nodes), along with understanding performance impacts. Our design allows for multiple COUPLER instances for multiple types of code coupling, thereby allowing the different codes to utilize the near-optimal placement and resources for each type of coupling. We are exploring optimizations such as MPI message packing and GPU acceleration.

4.4. EFFIS services

EFFIS workflows generate several types of output data, including science datasets, visualizations (as images), and performance results. EFFIS services allow scientists to plug in user-defined analysis and visualization methods, monitor simulations through a web-based dashboard, and run post-processing actions such as data migration. We expand on these topics in the following.

4.4.1. Dashboard. In addition to generating output images themselves, we have been connecting EFFIS visualization results to a dashboard for remote viewing (Figure 9). Our dashboard implementation uses the eSimMon framework.[22] However, the use of a structured interface to expose the images and data allows integration with other software components that would like to consume this output.

When dashboard functionality is enabled, EFFIS runs a service daemon, which is aware of the various visualization processes that are running, for both data and performance. When all images for a given (simulation) time step have been rendered, the service daemon bundles them into a `tar` archive and updates a timing text file (JSON) to record what time steps are ready for dashboard ingest. (EFFIS pragma replacement enables this ability as well; otherwise in general, one is not sure when a time step has finished.) This JSON file is the integration point for external systems that wish to consume the assets generated by the EFFIS service daemon. The JSON file and archive are exposed via a HTTP server in order to be accessible to remote external services.

Running at a remote site, the dashboard ingestion engine periodically fetches the timing JSON file over HTTP. If the values in this file show that data are available, it uploads those data to the dashboard's data management platform, Girder,[23] built on MongoDB[24] with data exposed through a RESTful API. The eSimMon client is a web application that uses the API provided by Girder to present and monitor the data associated with a run. Pictured in Figure 9's screenshot, the client can monitor a live run, dynamically refreshing to display the data for the current timestep or play back a completed run, replaying each timestep. The user has the ability to search and select the particular parameters to monitor. The images for a parameter can also be exported as an MPEG-4[25] file for playback offline.

Our primary focus is HPC resources, such as those found at leadership facilities like OLCF. At OLCF, there is one project-accessible directory space that accepts incoming HTTP requests suitable for image download. However, it is not accessible from the compute nodes; thus in our case, the EFFIS dashboard compatibility daemon must run on the service node.

4.4.2. User-defined analysis and visualization. In addition to the EFFIS Matplotlib-based plotting daemons, users require the flexibility to design their own visualization and analysis services: for example, to provide three-dimensional views of the data. The visualization and analysis must be able to perform efficiently, and have the flexibility to be placed where needed by the system. Compiling with an appropriate pragma can then prepare the images to be compatible for dashboard ingestion.

We have designed specialized visualizations for WDMApp. Figure 10 shows output from a service that calculates the turbulence in the plasma and renders the result. The visualization services are implemented in the VTK-m toolkit (Moreland et al., 2016), which provides portability across CPU and GPU multi-core computing devices. This allows EFFIS to schedule the service in a flexible way on available resources; the same service could run on a single core of a CPU, on multiple cores of a CPU, or on a GPU. Interpretation of the simulation data is done

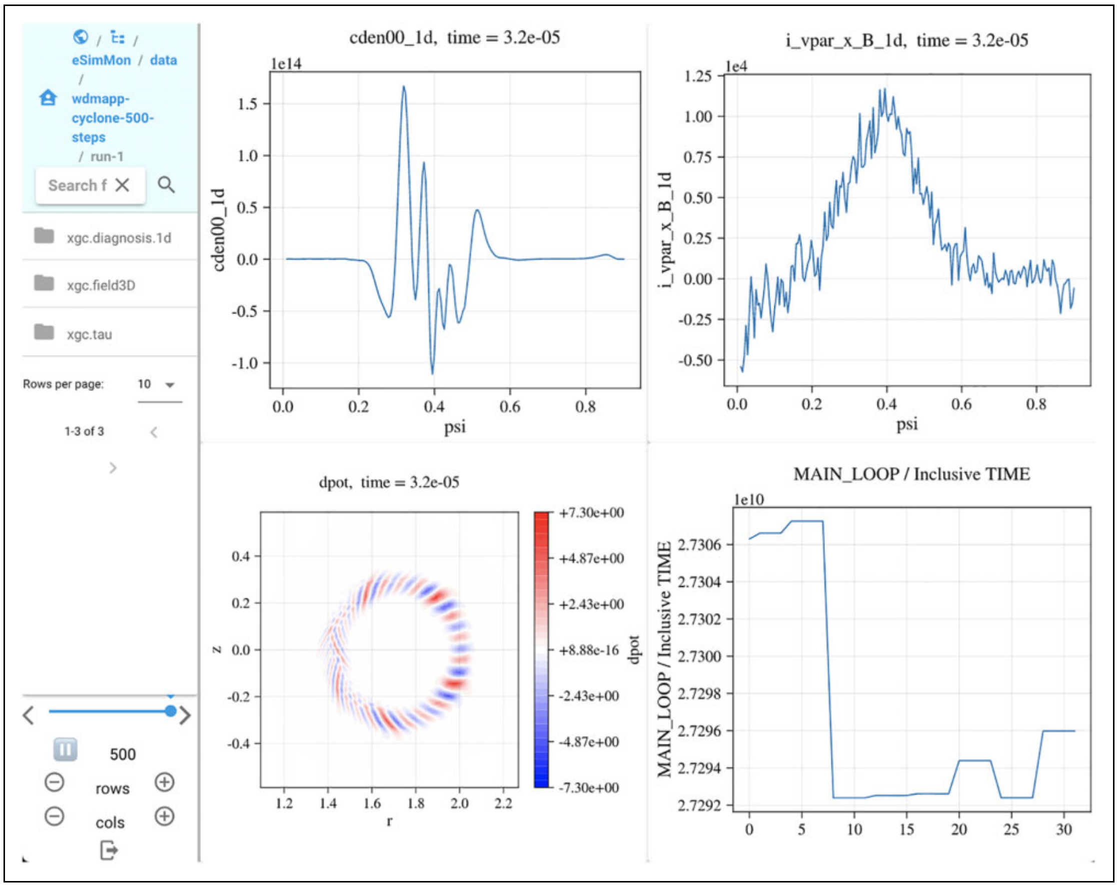

Figure 9. Example of eSimMon client dashboard interface. Four variables have been selected for live monitoring, including two 1D data variables, one 2D data variable, and a performance monitoring variable.

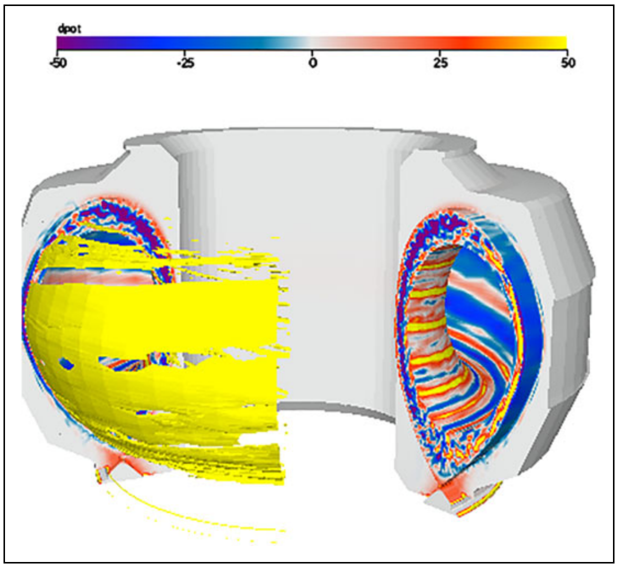

Figure 10. Example from custom EFFIS service to visualize the turbulence in an XGC simulation from a D3D tokamak run.

using the Fides schema.[26] Fides is a schema that annotates data in a stream or file to provide meaning for the data arrays and provide a data model for a mesh-based representation. The schema makes it possible for services to be connected in arbitrary ways and able to understand the data in the underlying streams.

4.4.3. Post processing. There are several post processing events which WDMApp will require to be automated immediately following an exascale simulation. These include: 1) aggregating images across multiple time steps into a video sequence/animation, 2) running additional post processing analysis, and 3) archiving all data in order to manage the metadata across the different outputs from a simulation. While the first two are fairly straightforward, the third beckons further discussion.

Application scientists often launch campaigns that execute complex workflows with different inputs and configurations, raising the question of how to efficiently manage the heterogeneous data produced by such campaigns. WDMApp faces two particular challenges.

First, the data are enormous and have highly variable types and formats. For instance, a single run within a campaign might generate data that includes, but is not limited to, the outputs of edge and core simulations; performance measured by TAU; and images and videos produced by visualization tools. These data are saved in different formats with different characteristics. When scientists launch multiple campaigns for different purposes, where the workflows within each campaign are executed multiple times with different inputs or configurations, the number and size of heterogeneous datasets generated can be problematic.

Second, the data cannot reside in fast storage tiers (such as parallel file systems) indefinitely, due to limited storage capacity. Eventually, data must be purged from the fast storage and moved to a slower archival storage system. For example on Summit, data can only persist on the parallel file system for a maximum of 90 days. It must be moved to the site's HPSS[27] archives or elsewhere off-site, lest it be permanently deleted.

Once data are moved to archival storage, it can take weeks or even months for scientists to retrieve them in order to run their analysis procedures. In such a case, the goal is to achieve efficient data management with a metadata service that not only tracks the placement and movement of heterogeneous data within the multi-tiered storage hierarchy, but also supports flexible and lightweight query operations to enable streamlined access to the needed data for further studies.

Our initial work on this metadata service have led to the design and implementation of a new metadata file format for ADIOS called BP4 (Wan et al., 2019) that significantly reduces metadata construction overhead. Future improvements will rely on two functionalities. First is the construction of a global indexing structure for tracking heterogeneous datasets from all campaigns. Currently, different indexing structures are under evaluation, including B-tree, hash table, and log-structured merge-tree (LSM tree). Second, rich information and attributes must be attached to the global indexing structure to enable a variety of query operations. Since many scientific datasets are already in self-describing formats, the existing metadata and attributes of each individual self-describing dataset can be extracted and inserted into the global indexing structure. Moreover, scientists are able to attach workflow and campaign-related information, such as the intention of each campaign or experiment run, to the global indexing structure.

5. WDMApp demonstration and results

We now exemplify how EFFIS enhances WDMApp. Although the focus of this article is not new physics studies or detailed performance measurements, we present characteristic results related to the challenges and requirements alluded to throughout the manuscript, paying particular attention to metrics of success defined for the framework.

Our main test case is the cyclone base case in circular geometry, with core-edge fluid coupling between GENE and XGC, which has been the subject of several WDMApp milestones and publications (Dominski et al., 2018; Merlo et al., 2020). We present results on two HPC platforms: Summit and Rhea.[28] Strongly coupled charge and field data between XGC and GENE is exchanged via the ADIOS asynchronous RDMA transport and all other data is saved to ADIOS `BP4` files. WDMApp runs have been modest in size to date: here, 32 nodes for XGC and a single node for GENE. However, the same methodologies and results analysis can be applied to larger runs. In the Summit case, the three EFFIS plotters (1D, 2D, performance) shared a single compute node. On Rhea, only a single executable is allowed per node, meaning three additional nodes were added for EFFIS.

Listing 6. Examples of performance monitoring variables: timers (upper block) and usage statistics (lower block).

```
14*{32}  BeginStep / Calls
14*{32}  BeginStep / Exclusive TIME
14*{32}  BeginStep / Inclusive TIME
14*{32}  EndStep / Inclusive TIME
14*{32}  XGC_COUPLING_CORE_EDGE / Inclusive TIME
2*{32}   RESTART_WRITE / Inclusive TIME
14*{32}  DIAGNOSIS / Inclusive TIME
14*{32}  ipc1:PUSH / Inclusive TIME
14*{32}  CHARGEI / Inclusive TIME
14*{32}  POISSON / Inclusive TIME
14*{32}  MAIN_LOOP / Inclusive TIME
16*{32}  MPI_Bcast() / Inclusive TIME
16*{32}  MPI_Gather() / Inclusive TIME
14*{32}  MPI_Reduce() / Inclusive TIME

16*{32}  Heap Memory Used (KB) / Max
16*{32}  Heap Memory Used (KB) / Mean
16*{32}  Heap Memory Used (KB) / Min
16*{32}  Heap Memory Used (KB) / Num Events
16*{32}  Heap Memory Used (KB) / Sum Squares
16*{32}  Memory Footprint (VmRSS) (KB) / Max
16*{32}  Message size for broadcast / Max
16*{32}  Message size for gather / Max
14*{32}  Message size for reduce / Max
16*{32}  Peak Resident Set Size (VmHWM) (KB) / Max
16*{32}  cpu: User % / Mean
16*{32}  io:read_bytes / Num Events
16*{32}  io:write_bytes / Num Events
16*{32}  meminfo:MemFree (MB) / Min
16*{32}  resident set size (kB) / Max
```

The reason we have not run a job occupying a significant fraction of Summit is twofold. First is simply a matter of compute time usage. EFFIS has been developed as part of the WDMApp program, and the majority of the project's compute hours are reserved for physics studies and performance optimization of the individual codes, not the framework manager. Secondly, as mentioned above, WDMApp coupling runs have been modest in size to date. This is because less complex physics is being solved, while appropriate, stable coupling algorithms are being developed. Upping the job size does not gain much; effectively, it over-resolves the simulation more than is really necessary. No EFFIS/ADIOS bottleneck is expected to inhibit larger workloads. In fact, the fractional overload for automated EFFIS processors will decrease. In future publications, larger EFFIS runs will be presented and in additional contexts beyond this WDMApp coupling example.

The first metric for success for the framework is assisting the project in achieving its performance targets. WDMApp workflows run via EFFIS collect hundreds of performance variables periodically throughout the simulation, as outlined in the Performance Monitoring section. Listing 6 shows a small subset of these quantities for the 32 node, 500 time step XGC run. The first block of variables are timers, including times spent in coupling and ADIOS I/O (`BeginStep`, `EndStep`, `XGC_COUPLING_CORE_EDGE`, `RESTART_WRITE`, `DIAGNOSIS`), different

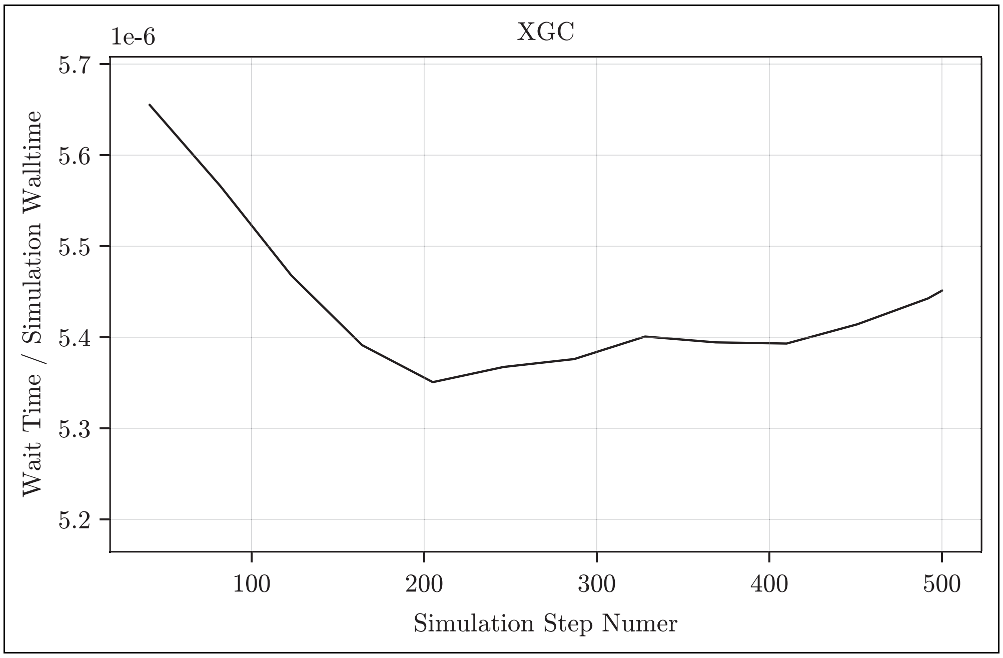

Figure 11. Relative wait time overhead for XGC in WDMApp fluid coupling on Rhea. As XGC dominates computation time, we want that overhead to be small, to minimize resource idling.

computational segments of the code (`ipc1`: `PUSH`, `CHARGE`, `POISSON`, `MAIN_LOOP`), and MPI subroutines. Each timer has three quantities associated with it, the number of calls, as well as the exclusive and inclusive timing measure; however, we have suppressed all but inclusive value after the first variable. The second block of variables are other performance counters: memory, CPU, and I/O usage statistics. Each comes with a minimum, maximum, mean, number of events, and sum of squares variant; but again, we have suppressed all but a single choice after the first example.

As noted earlier, it is important to minimize time that the edge code (XGC) is idling while waiting for coupling data. We use the collected performance measurements to test if this is case. Figure 11 shows the waiting time relative to the full simulation time. It is a small overhead, $\sim$ 0.005% of the execution time. Our framework enables us to make such measurements for each type of new coupling tested, to understand if it is an appreciable performance detriment.

During WDMApp code coupling, EFFIS adds a small library layer atop ADIOS, to support the additional EFFIS features. We confirm that EFFIS does not add a significant overhead to the ADIOS performance. Figure 12 shows the per-step coupling data movement timing, including the EFFIS-added time, for a WDMapp run on Summit with RDMA application-to-application data movement. For simplicity, the COUPLER has been run within GENE instead of as a separate application. (The figure does not include code-to-code representation transformation time.) For comparison, the timing for the entire step is also included at the top of Figure 12. The combined charge and field exchanges are approximately 0.2% of the looping

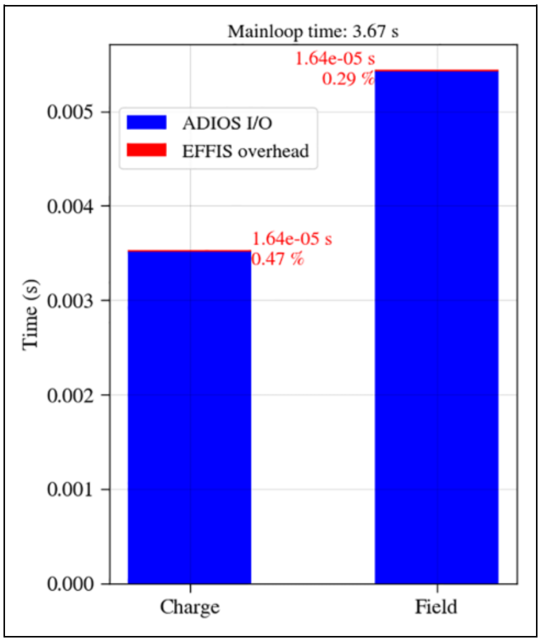

Figure 12. Coupling I/O timings for fluid coupling on Summit. Both the relative I/O expense and the overhead introduced by EFFIS are small.

time, with the EFFIS overhead less than 1% of that amount. Thus, data movement is not a significant cost here in our coupling approach.

Another of our framework goals is to make it as simple as possible to couple and configure as many codes as needed. Listing 7 demonstrates an example of enabled ease. Users need only edit one line of run-time input to change how coupling proceeds, e.g. switching the `adios_engine`

Listing 7. Simple workflow composition configuration, coupling `core`, `edge`, and `custom-analysis`.

```
run:

  core:
    .density-coupling:
      output_path: density.bp
      adios_engine: BP4
    .field-coupling:
      reads: edge.field-coupling

  edge:
    .density-coupling:
      reads: core.density-coupling
    .field-coupling:
      output_path: field.bp
      adios_engine: BP4
    .mesh:
      output_path: xgc.mesh.bp

  custom-analysis:
    executable_path: /usr/bin/field-stats
    .data:
      reads: edge.field-coupling
    .mesh:
      reads: edge.mesh
```

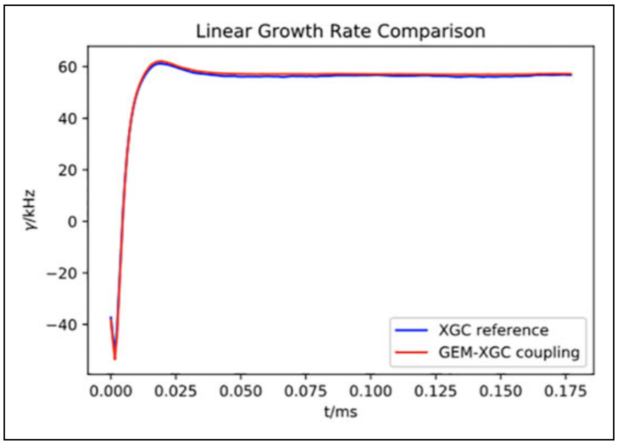

Figure 13. Time evolution of linear growth rate of XGC reference (blue line) and GEM-XGC coupling (red line) results (Cheng et al., 2020). Comparing the growth rate is a good validation check because it should match between the cases.

for the `edge's field-coupling` from `BP4` to `SST` to switch from file-based to memory-based data movement. Moreover, adding another executable only requires adding another section under `run`. Here, `custom-analysis` (weakly) couples through two inputs, `data` and `mesh`, setting `reads` to the proper edge data products. This also demonstrates the metric to allow seamless integration of in situ analysis/visualization. It is merely another workflow component to couple to in the same fashion, which is configured in the same way. The few lines of pragmas added to the codes are analogous to the previously exemplified in Listing 1 and Listing 2.

Custom analysis in EFFIS can be used to check the validity of physics results. For example, we have computed the growth rate in simulations, and then these can be compared in post-processing. Figure 13 is an example, where the plot has been reproduced from Cheng et al. (2020). Here, GEM has been used in the core instead of GENE (with code-to-code representation transformation occurring inside GEM; the COUPLER was not GEM-compatible at the time of publication). A linear growth mode is expected to asymptote to a constant in the plot. The coupled simulation is compatible with the reference non-coupled one.

6. Conclusions

In this paper we have presented EFFIS, a general framework used to compose, execute, and couple multiple codes in workflows. EFFIS' features have been driven by the ECP WDMApp, which requires a high performance framework to couple multiple gyrokinetic simulations on an exascale supercomputer. The software environment contains methods to strongly or weakly couple, in both synchronous or asynchronous fashions. Furthermore, it allows users to monitor the performance and physics data and can be programmed to send control commands to the applications that are contingent on the data. As a workflow manager, EFFIS differs from other systems in that it provides a way through which it can subscribe to data published in the workflow, then provide fine-grained support for deploying in situ workflow services.

The essential elements of EFFIS include:

- An easy-to-use composition engine, which allows scientists to specify the applications that are to execute during a run, and specify the resource and placement of the applications, along with the coupling technique.
- Flexible and performant self-describing I/O support, for both disk storage and in-memory data movement.
- A state-of-the-art COUPLER, which can subscribe to data from one or more codes and perform the mathematics required for exchanging data in a accurate and stable manner.
- A method for performance monitoring of all coupled applications, yielding data feeds that can be subscribed to through EFFIS.
- Automated plotting of output data.
- Advanced command-and-control processing for automating workflow actions contingent on the data.
- Extensibility for user services, e.g. custom analysis and visualization, and run monitoring via a dashboard.
- Post processing for archiving, campaign management, and other purposes.

In the longer term, we envision creating a useful community framework with WDMApp and EFFIS, one that is reasonably easy to use for users who are not experts with the first-principles codes XGC, GENE, and GEM. We have described progress toward this goal, but more research and development remains, such as defining the interface and schema requirements for all applications to be coupled, and designing a validation engine to ingest a supplied configuration and test if it is runable, at least insofar as not to crash early during execution. There is a large possible parameter space, which only grows as more codes are coupled. It is important to be able to identify out-of-range values, misspellings, incompatible data types, etc.

In the context of WDMApp, we demonstrated that the data movement cost in the WDMApp strong coupling case is low, including the associated EFFIS overhead. Furthermore, results indicated that the time XGC spends for waiting during coupling is close to zero. We also exemplified EFFIS plotting services, which can be used by multiple users to monitor WDMApp runs with the eSimMon dashboard, including visualizations from the performance instrumentation. We are continuing to develop EFFIS, to enable greater degrees of command-and-control, more generalized automated plotting, additional post-processing hooks, and more.

In the next step of the WDMApp project, we will enable kinetic coupling, which will further stress our system, and we will introduce new techniques to reduce, analyze, and visualize salient features during a simulation. We are also building up a knowledge repository of EFFIS input scripts, along with the inputs to all of the coupled runs and their corresponding checkpoint files, so that users can run these example runs and then examine the performance, and accuracy. This will allow us to explore more complex, kinetic coupling with the WDMApp, and include more in situ analysis and reduction during the simulation.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: 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 National Nuclear Security Administration. It used resources of the Oak Ridge Leadership Computing Facility, a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725.

ORCID iD

Eric Suchyta https://orcid.org/0000-0002-7047-9358
Ian Foster https://orcid.org/0000-0003-2129-5269
Cameron W Smith https://orcid.org/0000-0001-9258-5226

Notes

1. WDMApp Brochure, https://www.exascaleproject.org/wp-content/uploads/2020/01/ECP_AD_Fusion-Energy.pdf
2. ECP Homepage, https://www.exascaleproject.org/
3. PerfStubs, https://github.com/khuck/perfstubs
4. Titan Homepage, http://www.olcf.ornl.gov/olcf-resources/compute-systems/titan/
5. AToM Homepage, https://atom.scidac.io/
6. SciDAC Homepage, https://www.scidac.gov/
7. HBPS Homepage, https://epsi.pppl.gov/
8. ASCR Homepage, https://www.energy.gov/science/ascr/advanced-scientific-computing-research
9. FES Homepage, energy.gov/science/fes/fusion-energy-sciences
10. Frontier Homepage, olcf.ornl.gov/frontier/
11. Aurora Homepage, https://www.alcf.anl.gov/aurora
12. MySQL Homepage, https://www.mysql.com/
13. GTC, http://sun.ps.uci.edu/gtc/
14. YAML Specification, https://yaml.org/spec/1.2/spec.html
15. EFFIS Source Repository, https://github.com/wdmapp/effis
16. WDM Release Documentation, https://wdmapp.readthedocs.io/en/latest/effis/effis-main.html
17. Summit Homepage, https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/
18. Theta Homepage, https://www.alcf.anl.gov/support-center/theta
19. Cori Homepage, https://www.nersc.gov/systems/cori/
20. Cobalt Job Scheduler, https://xgitlab.cels.anl.gov/aig-public/cobalt
21. OLCF Homepage, https://www.olcf.ornl.gov/
22. eSimMon, https://github.com/Kitware/eSimMon
23. Girder Homepage, https://www.github.com/girder/girder
24. MongoDB, https://github.com/mongodb/mongo
25. MPEG-4, https://tools.ietf.org/html/rfc4337
26. Fides Software, https://gitlab.kitware.com/vtk/fides
27. HPSS Homepage, http://www.hpss-collaboration.org/
28. Rhea Homepage, https://www.olcf.ornl.gov/olcf-resources/compute-systems/rhea/

References

Altintas I, Berkley C, Jaeger E, et al. (2004) Kepler: an extensible system for design and execution of scientific workflows. In: *16th international conference on scientific and statistical database management*, Santorini, Greece, 23–23 June 2004, pp. 423–424. IEEE.

Aymar R, Barabaschi P and Shimomura Y (2002) The ITER design. *Plasma Physics and Controlled Fusion* 44(5): 519–565.

Babuji Y, Woodard A, Li Z, et al. (2019) Parsl: pervasive parallel programming in Python. DOI: 10.1145/3307681.3325400.

Boehme D, Huck K, Madsen J, et al. (2019) The case for a common instrumentation interface for HPC codes. In: *IEEE/ACM international workshop on programming and performance visualization tools (ProTools)*, Denver, CO, 17 November 2019, pp. 33–39.

Bonoli P, McInnes LC, Sovinec C, et al. (2015) *Report of the workshop on integrated simulations for magnetic fusion energy sciences*. Technical Report, Office of Fusion Energy Sciences and the Office of Advanced Scientific Computing Research.

Bussmann M, Burau H, Cowan TE, et al. (2013) Radiative signature of the relativistic Kelvin-Helmholtz instability. In: *SC'13: Proceedings of the international conference on high performance computing, networking, storage and analysis*, Denver, CO, USA, 17–22 November 2013, pp. 1–12. IEEE.

Chard K, Tuecke S and Foster I (2016) Globus: recent enhancements and future plans. In: *XSEDE16 conference*, Miami, FL, USA, 17–21 July 2016, pp. 1–8.

Chen Y and Parker SE (2007) Electromagnetic gyrokinetic δ f particle-in-cell turbulence simulation with realistic equilibrium profiles and geometry. *Journal of Computational Physics* 220(2): 839–855.

Cheng J, Dominski J, Chen Y, et al. (2020) Spatial core-edge coupling of the PIC gyrokinetic codes GEM and XGC. *Physics of Plasmas 27(12)*: 122510.

Cummings J, Lofstead J, Schwan K, et al. (2010) EFFIS: an end-to-end framework for fusion integrated simulation. In: *18th Euromicro conference on parallel, distributed and network-based processing*, Pisa, 17–19 February 2010, pp. 428–434.

Deelman E, Peterka T, Altintas I, et al. (2018) The future of scientific workflows. *International Journal of High Performance Computing Applications* 32(1): 159–175.

Deelman E, Vahi K, Juve G, et al. (2015) Pegasus: a workflow management system for science automation. *Future Generation Computer Systems* 46: 17–35.

Dominski J, Cheng J, Merlo G, et al. (2020) Spatial coupling of gyrokinetic simulations, a generalized scheme based on first-principles. *Physics of Plasmas* 28(2): 022301.

Dominski J, Ku S, Chang CS, et al. (2018) A tight-coupling scheme sharing minimum information across a spatial interface between gyrokinetic turbulence codes. *Physics of Plasmas* 25(7): 072308.

Dorf MA, Dorr MR, Hittinger JA, et al. (2016) Continuum kinetic modeling of the tokamak plasma edge. *Physics of Plasmas* 23(5): 056102.

Erl T (2005) *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*. Hoboken, NJ: Prentice Hall.

Folk M, Heber G, Koziol Q, et al. (2011) An overview of the HDF5 technology suite and its applications. In: Baumann P, Bill H, Kjell O and Silvia S (eds) *EDBT/ICDT 2011 workshop on array databases*, Uppsala, Sweden, 25 March 2011, pp. 36–47.

Foster I, Ainsworth M, Allen B, et al. (2017) Computing just what you need: online data analysis and reduction at extreme scales. In: *European conference on parallel processing*, Santiago de Compostela, Spain, 28–29 August 2017, pp. 3–19. Springer.

Foster I, Ainsworth M, Bessac J, et al. (2020) Online data analysis and reduction: an important co-design motif for extreme-scale computers. *International Journal of High-Performance Computing Applications* (in press).

Foster I and Kesselman C (2006) Scaling system-level science: scientific exploration and IT implications. *Computer* 39(11): 31–39.

Germaschewski K, Allen B, Dannert T, et al. (2020) Exascale whole-device modeling of fusion devices: porting the GENE gyrokinetic microturbulence code to GPU. *Physics of Plasmas* (submitted).

Germaschewski K, Chang C, Dominski J, et al. (2019) Quantifying and improving performance of the XGC code to prepare for the exascale. *APS* 2019: BP10–081.

Godoy WF, Podhorszki N, Wang R, et al. (2020) ADIOS 2: the adaptable input output system. A framework for high-performance data management. *SoftwareX* 12: 100561.

Görler T, Lapillonne X, Brunner S, et al. (2011) The global version of the gyrokinetic turbulence code GENE. *Journal of Computational Physics* 230(18): 7053–7071.

Jain A, Ong SP, Chen W, et al. (2015) FireWorks: a dynamic workflow system designed for high-throughput applications. *Concurrency and Computation: Practice and Experience* 27(17): 5037–5059.

Jardin S (2004) A triangular finite element with first-derivative continuity applied to fusion MHD applications. *Journal of Computational Physics* 200(1): 133–152.

Jenko F, Dorland W, Kotschenreuther M, et al. (2000) Electron temperature gradient driven turbulence. *Physics of Plasmas* 7(5): 1904–1910.

Khaziev R and Curreli D (2018) hPIC: a scalable electrostatic particle-in-cell for plasma-material interactions. *Computer Physics Communications* 229: 87–98.

Ku S, Chang C and Diamond P (2009) Full-f gyrokinetic particle simulation of centrally heated global ITG turbulence from magnetic axis to edge pedestal top in a realistic tokamak geometry. *Nuclear Fusion* 49(11): 115021.

Ku S, Hager R, Chang C, et al. (2016) A new hybrid-Lagrangian numerical scheme for gyrokinetic simulation of tokamak edge plasma. *Journal of Computational Physics* 315: 467–475.

Liu Q, Logan J, Tian Y, et al. (2014) Hello ADIOS: the challenges and lessons of developing leadership class I/O frameworks. *Concurrency and Computation: Practice and Experience* 26(7): 1453–1473.

Mehta K, Allen B, Wolf M, et al. (2019) A codesign framework for online data analysis and reduction. In: *IEEE/ACM workflows in support of large-scale science*, Denver, CO, 17 November 2019, pp. 11–20.

Meneghini O, Smith S, Lao L, et al. (2015) Integrated modeling applications for tokamak experiments with OMFIT. *Nuclear Fusion* 55(8): 083008.

Meneghini O, Smith S, Snyder P, et al. (2017) Self-consistent core-pedestal transport simulations with neural network accelerated models. *Nuclear Fusion* 57(8): 086034.

Meneghini O, Snyder PB, Smith SP, et al. (2016) Integrated fusion simulation with self-consistent core-pedestal coupling. *Physics of Plasmas* 23(4): 042507.

Merlo G, Dominski J, Bhattacharjee A, et al. (2018) Cross-verification of the global gyrokinetic codes GENE and XGC. *Physics of Plasmas* 25(6): 062308.

Merlo G, Janhunen S, Jenko F, et al. (2020) First coupled GENE-XGC microturbulence simulations. *Physics of Plasmas* 28(1): 012303.

Merzky A, Santcroos M, Turilli M, et al. (2015) RADICAL-Pilot: scalable execution of heterogeneous and dynamic workloads on supercomputers. CoRR, abs/1512.0819.

Moreland K, Sewell C, Usher W, et al. (2016) Vtk-m: accelerating the visualization toolkit for massively threaded architectures. *IEEE computer graphics and applications* 36(3): 48–58.

Park JM, Ferron JR, Holcomb CT, et al. (2018) Integrated modeling of high β_n steady state scenario on DIII-D. *Physics of Plasmas* 25(1): 012506.

Peterka T, Bard D, Bennett J, et al. (2019) ASCR workshop on in situ data management: enabling scientific discovery from diverse data sources. DOI: 10.2172/1493245.

Shende SS and Malony AD (2006) The Tau parallel performance system. *International Journal of High Performance Computing Applications* 20(2): 287–311.

Staples G (2006) TORQUE resource manager. In: *ACM/IEEE conference on supercomputing*, SC '06, p. 8-es. New York, NY, USA: Association for Computing Machinery. DOI:10.1145/1188455.1188464.

Tchoua R, Klasky S, Podhorszki N, et al. (2010) Collaborative monitoring and analysis for simulation scientists, Chicago, IL, 17–21 May 2010, pp. 235–244. DOI: 10.1109/CTS.2010.5478506.

Wan L, Mehta K, Klasky S, et al. (2019) Data management challenges of exascale scientific simulations: a case study with the gyrokinetic toroidal code and ADIOS. In: *Proceedings of the international conference on computational methods*, vol. 6, Singapore, 9–13 July 2019, pp. 493–503.

Wang R, Tobar R, Dolensky M, et al. (2020) Processing full-scale square kilometre array data on the summit supercomputer. In: *2020 SC20: international conference for high performance computing, networking, storage and analysis (SC)*, Atlanta, GA, USA, 9–19 November 2020, pp. 11–22. IEEE Computer Society.

Wilde M, Hategan M, Wozniak JM, et al. (2011) Swift: a language for distributed parallel scripting. *Parallel Computing* 37(9): 633–652.

Yoo AB, Jette MA and Grondona M (2003) SLURM: Simple Linux utility for resource management. In: Feitelson D, Rudolph L and Schwiegelshohn U (eds) *Job Scheduling Strategies for Parallel Processing*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 44–60.

Zhou S (1992) LSF: load sharing in large heterogeneous distributed systems. In: *Workshop on cluster computing*, vol. 136, Orlando, FL, USA, April 1992, pp. 1–48.

Author biographies

Eric Suchyta earned his Ph.D. in Physics from the Ohio State University in 2015 and currently holds a Computer Scientist position at Oak Ridge National Laboratory, where his research interests include workflows, code coupling, and data management. He is a member of the ECP Whole Device Modeling Application, leading framework design, workflow orchestration, and data movement efforts.

Scott Klasky is a distinguished scientist and group leader in the Scientific Data Group. He is the leader of the ADIOS-ECP project, which previously won an R&D 100 award in 2013. He is also the leader of the MGARD data reduction project, and focuses on workflow automation, data reduction, data movement, code coupling, and in situ processing of data.

Norbert Podhorszki is a senior research scientist in the Scientific Data Group at Oak Ridge National Laboratory. He is one of the key developers of ADIOS that won an R&D100 award in 2013. His main research interest is in creating I/O and staging solutions for in-situ processing of data on leadership class computing systems.

Matthew Wolf is a Senior Computer Scientist in the Scientific Data Group at Oak Ridge National Laboratory. His research interests are in in situ and online workflow systems and data lifecycle management.

Abolaji Adesoji is a second-year PhD student at RPI's Scientific Computation Research Center and he is working on the wdmapp external coupling project. His research interest spans HPC software development, parallel programming and in-situ ML data compression.

CS Chang is a Managing Principal Physicist at Princeton Plasma Physics Laboratory and the head of the SciDAC Partnership Center for High-fidelity Boundary Plasma Simulation. He is also the Co-Head for Science in the ECP WDMApp project. C.S. Chang is a Fellow of American Physical Society.

Jong Choi is a scientist in the Scientific Data Group at Oak Ridge National Laboratory. His research interests are in scalable data management and analysis.

Philip E Davis is a software developer at the Rutgers Discovery Informatics Institute at Rutgers University. His development work focuses on data staging infrastructure for scientific computing workflows.

Julien Dominski is a research physicist at the Princeton Plasma Physics Laboratory. He received his Ph.D. from the Swiss Plasma Center (EPFL), where he explored the influence of passing electrons on electrostatic turbulence in tokamaks. At PPPL, he works on gyrokinetic codes and their application, as a member of the XGC team.

Stéphane Ethier is a Principal Computational Scientist at the Princeton Plasma Physics Laboratory (PPPL) and co-head of the Advanced Computing Group. His work focuses on high performance computing on large-scale systems, particle-in-cell methods for magnetic fusion research, GPU programming, data management, and other related fields.

Ian Foster is a Senior Scientist and Distinguished Fellow, and director of the Data Science and Learning Division, at Argonne National Laboratory, and the Arthur Holly Compton Distinguished Service Professor of Computer Science at the University of Chicago.

Kai Germaschewski is an Associate Professor at the University of New Hampshire. His research interests are in computational plasma physics, and specifically in effectively using state of the art GPU-based supercomputers for large-scale simulations.

Berk Geveci leads the Scientific Computing Team at Kitware Inc. He is one of the leading developers of the ParaView visualization application and the Visualization Toolkit (VTK). His research interests include large scale parallel computing, computational dynamics, finite elements and visualization algorithms.

Chris Harris is a Principal Engineer in the Scientific Computing team at Kitware Inc. His work focuses on enabling complex scientific workflows in a broad range of domains.

Kevin A Huck is a Research Associate and Computer Scientist in the Oregon Advanced Computing Institute for Science and Society (OACISS) at the University of Oregon. His research interests include performance measurement, analysis, and runtime adaptation.

Qing Liu is an Assistant Professor in the Department of Electrical and Computer Engineering at NJIT and Joint Faculty with Oak Ridge National Laboratory. His research interests include high-performance computing, storage, and networking.

Jeremy Logan is a Computer Scientist in the Workflow Systems Group at Oak Ridge National Laboratory. His research interests include Model Driven Generative Techniques for HPC and Scientific Data Management. He is the primary developer of the Skel toolkit.

Kshitij Mehta is a Computer Scientist in the Scientific Data Group at Oak Ridge National Laboratory. He is one of the lead developers of the Cheetah and Savanna toolset. His research interests lie in workflow technologies for dynamically adaptive workflows for scientific computing.

Gabriele Merlo earned his Ph.D. in Physics from the Ecole Polytechnique Fédérale de Lausanne in 2016 and currently is a Postdoctoral Fellow at the Oden Institute at the University of Texas at Austin. He is part of the GENE development team.

Shirley V Moore is an Associate Professor of Computer Science at the University of Texas at El Paso and a senior member of the Association for Computing Machinery. She was a Senior Computer Scientist at Oak Ridge National Laboratory (ORNL) from 2016 to 2020 where she led ORNL efforts on several Exascale Computing Project (ECP) projects. Her research interests are in computer architecture and performance analysis.

Todd Munson is a senior computational scientist at Argonne National Laboratory, a senior scientist for the Consortium for Advanced Science and Engineering at the University of Chicago, and the Software Ecosystem and Delivery Control Account Manager for the Exascale Computing Project.

Manish Parashar is Distinguished Professor of Computer Science at Rutgers University and the founding Director of the Rutgers Discovery Informatics Institute (RDI2), whose research interests are in the broad areas of Parallel and Distributed Computing and Computational and Data-Enabled Science and Engineering. Manish is the founding chair of the IEEE Technical Consortium on High Performance Computing (TCHPC), Editor-in-Chief of the IEEE Transactions on Parallel and Distributed Systems, an AAAS Fellow, an IEEE/IEEE Computer Society Fellow, and an ACM Distinguished Scientist.

David Pugmire is a Senior Research Scientist in the Scientific Data Group at the Oak Ridge National Laboratory. His research interests are in scalable in situ visualization and analysis of scientific data.

Mark S Shephard is the Samuel A. and Elisabeth C. Johnson, Jr. Professor of Engineering, and the Director of the Scientific Computation Research Center at Rensselaer Polytechnic Institute. He is a past President of the US Association for Computational Mechanics and editor of Engineering with Computers.

Cameron W Smith is a Senior Research Scientist at Rensselaer Polytechnic Institute's Scientific Computation Research Center with expertise in simulation automation, parallel computation, load balancing, unstructured meshing, and heterogeneous architectures.

Pradeep Subedi is a Research Associate at the Rutgers Discovery Informatics Institute at Rutgers University. His research work focuses on machine learning techniques, autonomic extreme scale data management, and supporting scalable middleware for scientific workflows.

Lipeng Wan is a computer scientist and R&D staff member of the Scientific Data Group in Computer Science and

Mathematics Division at Oak Ridge National Laboratory. He received his Ph.D. in computer science from the University of Tennessee, Knoxville in 2016. His research interests include scientific data management, high-performance computing and I/O, file systems and non-volatile storage devices.

Ruonan Wang received his Ph.D. at the University of Western Australia on data system design for the Square Kilometer Array radio telescope. Since joining the Scientific Data Group at the Oak Ridge National Laboratory, he has been focusing on designing and optimizing production software systems for data I/O, data staging, and network data transfer, which target extremely large applications running on the world's top supercomputers.

Shuangxi Zhang received his Ph.D in plasma physics from University of Science and Technology of China in 2013 then worked as a postdoc at Kyoto University in Japan and the University of Strasbourg in France. Currently, he works at Rensselaer Polytechnic Institute as the main developer of the COUPLER. His research includes magnetically confined plasma physics, gyrokinetic theoretical models, large scale parallel simulations of magnetized plasmas based on implementing Euler scheme, Semi-Lagrangian scheme and Particle-In-Cell scheme over the gyrokinetic model.