Hybrid Analysis of Fusion Data for Online Understanding of Complex Science on Extreme Scale Computers

Eric Suchyta*‖, Jong Youl Choi*, Seung-Hoe Ku†, David Pugmire*, Ana Gainaru*, Kevin Huck‡, Ralph Kube†, Aaron Scheinberg§, Frédéric Suter*, Choongseock Chang†, Todd Munson¶, Norbert Podhorszki*, Scott Klasky*

*Oak Ridge National Laboratory
†Princeton Plasma Physics Laboratory
‡University of Oregon
§Jubilee Development
¶Argonne National Laboratory
‖Email: suchytaed@ornl.gov

***Abstract*—The current practice for fusion scientists running first principle simulations on high performance computing platforms is to either run their simulations and output their data for post-hoc analysis, or to place in situ analytics into their code. In this paper we examine a complex workflow using XGC fusions simulation run on the Oak Ridge Leadership Computing Facility's supercomputer Summit, which also involve three analyses as part of the results necessary for scientific discovery. We discuss the challenges faced when implementing these algorithms and present an original hybrid staging technique to help enable the physicists to make discoveries during the execution of the simulation. By creating this infrastructure, we can examine complicated physics results, which may not have been possible without the infrastructure. For example, our work enables the online visualization of turbulent homoclinic tangle around the magnetic X-point, breaking the last confinement surface. This visualization could help fusion scientists to better understand and improve the turbulence spread of plasma exhaust heat, which is crucial toward realizing plasmas beyond the currently accessible physics regimes of present-day tokamak reactors. The physics of turbulent homoclinic tangle will be reported in a future physics publication, by utilizing the original online analysis/visualization framework presented in this paper.**

***Index Terms*—Fusion Science, Online Analysis and Visualization, Workflows, Extreme Scale**

I. Introduction

The current practice for scientists from many different domains, such as fusion energy, material science, and climate modeling, is to design High Performance Computing (HPC) simulations, run them on modern HPC systems, and once the simulations are finished, analyze and visualize the produced data. These scientific applications are now overwhelming the storage and I/O systems of the largest supercomputers in the world, often generating over ten Petabytes of data in a single day. Many of these applications will soon require Exabytes of output to make new scientific discoveries, and will need new ways to cope with the growing disparity between supercomputer computation speeds and I/O rates [1].

Over the last two decades, the HPC scientific data management community has researched many new techniques to cope with the challenges related to this ever increasing need for data analysis and storage. Three main areas have been investigated. First, as the size of the data needed by scientists for analysis increases, the need to improve the parallel I/O layer to store data as fast as possible became more urgent. The development of technologies, such as ADIOS [2] and HDF5 [3], drastically improved the writing performance of HPC simulations, achieving a throughput of over 1 TB/s on one of the current leading supercomputers. Second, applying data compression techniques reduces the size of the data output, often by one or more orders of magnitude. For instance, the lossy data compression techniques implemented by MGARD [4], [5] allowed fusion scientists to reduce their data by a factor of 77 while preserving many advanced quantities of interest, such as mass, momentum, energy, and temperature anisotropy [6]. Third, in situ processing of the simulation data, i.e., analyzing and/or visualizing data as it is produced, enables online diagnostics of simulated physics. The generic in situ term now covers a broad range of computing motifs, where data is either processed inline with the simulation on the same resources, or staged and processed on separate analysis nodes, or staged and processed on the same nodes but different cores from where the simulation runs [7]. For several classes of analysis, hybrid scenarios in which part of the calculation runs inline and another part runs on a small number of analysis nodes, constitutes the most efficient approach [8].

As scientists increasingly leverage these techniques to complete their scientific campaigns, they create complex online workflows that need to be efficiently executed. When simulations, analyses, and visualizations are designed in a way that allows the components to be executed within a single workflow, it allows data to be seamlessly streamed in memory, staged over a HPC network, or using files, and allows adaptation of the data compression level to the importance of the information stored. New scientific breakthroughs can be expected in this paradigm. The approach allows simulation data to be processed and monitored on the fly by domain

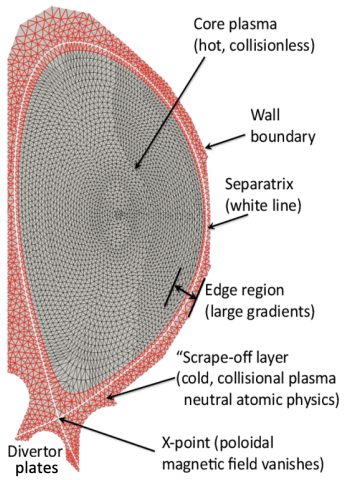

Fig. 1. Poloidal cross-section of plasma region in ITER, showing the magnetic separatrix surface (the last confinement surface) and the divertor plate area.

scientists, computer scientists, and performance engineers, enabling collaboration for multi-domain applications and faster feedback regarding the state of the simulation, which can be used to guide the discovery process and facilitate a faster convergence to insightful knowledge.

In this paper, we design a complex in situ workflow that includes a simulation from the fusion science domain, two types of analysis, and a visualization algorithm, in order to illustrate the associated workflow challenges and optimization opportunities, as well as how these enhancements could help enable new scientific discoveries. Our novel contributions include creating the in situ GPU-based visualization and two CPU analysis calculations, as well as optimizations to the application code itself for hybrid staging, all of which help enable scientific insight at low resource cost. Section II motivates the physics simulation case and its computer science ecosystem, then Section III details our hybrid analysis/visualization enhancements, and Section IV demonstrates the performance impact and physics results of the work, with concluding remarks in Section V.

II. Motivation

High-fidelity first-principles based simulations for fusion science are necessary to understand the fundamental physics of plasma confinement and to enable predictive capability. They help guide design decisions for future experimental facilities, as well as inform successful operation of upcoming experiments, such as ITER [9]. In the next decades, ITER aims to realize plasmas that are well beyond the physics regimes accessible in any present-day tokamak experiments. The 5D kinetic Particle-In-Cell (PIC) code XGC [10] has been developed to study the physics of such cases: foremost, to carefully simulate the complicated multi-physics in the edge of the plasma (red region in Figure 1). These simulations are compute intensive, requiring large HPC systems.

XGC solves the Vlasov kinetic equation after reducing its phase-space dimensionality from 6D to 5D. The relevant turbulence and particle dynamics in magnetized tokamak plasmas have characteristic time scales much slower than the gyro-frequency. Thus, the fast gyro-motion is analytically reduced to gyro-rings, removing the gyro-phase angle dimension. The resulting equation is the 5D "gyrokinetic equation." To reduce particle noise, XGC uses about 10 trillion marker particles for simulation of ITER plasmas, and runs on today's most powerful available supercomputer, Summit [11]. This dimensionality reduction in formulating the gyrokinetic equation is necessary, because it is impossible to perform a full 6D kinetic Vlasov simulation of tokamak plasma even on an exascale computer; it is too computationally expensive.

Following the PIC approach, particles are time-marched in the electromagnetic field that is gathered from a discretized mesh, and next undergo Fokker-Planck collisions, then the charge and current are deposited to the mesh, and the Poisson and Ampere's equations are solved to obtain the electromagnetic field on the mesh. Figure 1 shows an ITER mesh for XGC. The unique feature of XGC is that it accommodates complex edge geometry using an unstructured triangular mesh that includes realistic divertor plates (where the exhaust heat is gathered and dissipated) and the magnetic separatrix (which is the last confinement surface). XGC also includes Monte Carlo neutral atoms, with atomic interactions with plasma, that are generated by bombarded plasma particles on the divertor plates and recycled back into the plasma.

Over the last 13 years, different software frameworks have been tightly integrated into the XGC ecosystem to allow scientists to analyze data in more depth and discover new effects [12], [13]. Most of the older implementations were designed with the workflow framework Kepler [14], a fast I/O framework [15], as well as other HDF5 I/O for some output, and a dashboard [16]. Evolving toward exascale has necessitated updates for even tighter integration of technologies [17], [18]. The inclusion of lossy compression to reduce the total data output is also beginning to be evaluated, using either MGARD [6] or SZ [19], and requiring that the errors of derived quantities remain smaller than a specified bound.

ITER targets energy gain, but there are potentially serious impediments to operating in the target regime where 10 times more internal fusion energy is produced than the external input energy (Q=10), which urgently needs to be understood and resolved. Exploration of present tokamak data show that the exhaust heat on Q=10 ITER could be focused on such a narrow-width channel that the peak power density would significantly exceed the material's endurance limit. This could cause damage to the material wall, called the "divertor plates", even at the very first Q=10 attempt. Remedies to this impediment require difficult operational scenarios that could slow scientific progress. However, recent extreme-scale XGC simulations predict that Q=10 ITER is in a different physics regime and the heat-load width is expected to be 6-8 times wider at about the original design level than what was predicted by extrapolation of previous data. This hints at the

possibility that the operation of ITER could be much easier, allowing faster scientific progress.

A remaining question in the new finding is a fundamental understanding of the physics mechanism that broadens the divertor heat-load width in the Q=10 ITER plasma. This understanding could enable physicists to test the phenomenon in present tokamaks and, if validated, to create methods to broaden the heat-load width even further. The main difficulty is the complexity of the electromagnetic turbulence physics in the vicinity of the last confinement surface.

Understanding the fundamental particle-turbulence interaction dynamics requires workloads containing a few data analyses of the different physics phenomena, and more importantly, to combine these analyses together coherently in space-time. However, running all these analyses inline would slow the simulation to an unacceptable level and post-hoc processing would not capture the nonlinear particle-turbulence interaction self-consistently. Thus, we must couple multiple codes, including XGC and analysis/visualization codes, into one complex workflow. For each of the analyses, there is the decision of where the computation occurs (inline with the simulation, on dedicated resources, or following a hybrid approach), when the analysis is performed (synchronously, right after the simulation's data is produced or asynchronously, happening concurrently with the simulation), and how the communication occurs (files versus memory versus staging). Each decision results in performance trade-offs between multiple parameters and design choices, and strongly depends on the behavior of all the codes involved. Separating the analysis from the simulation allows for higher concurrency and decreases the total time to solution as long as the analysis can scale and keep up with the simulation steps. Conversely, the additional data movements may degrade performance due to network or I/O congestion.

We will demonstrate that by running XGC coupled with analysis and visualization codes at scale in a hybrid workflow, we are able to visualize the turbulent "homoclinic tangle" dynamics online, while an inline or post-hoc [20] workflow would slow the time to insight to an unacceptable level for large simulations. As shown in Section IV, the tangled structure makes the last confinement surface break into wild lobe-like structures around the magnetic X-point. Puncture plots of a dynamic turbulent homoclinic tangle need to be visualized at every simulation output step online, as fundamental understanding of the turbulent phenomena could be key to understanding the spread of plasma exhaust heat for Q=10 ITER operation. While homoclinic tangle science is not the subject of this paper, the hybrid analysis tools established and discussed in this paper will be utilized to advance this science, which will be reported elsewhere in the near future.

III. Analysis Overview

A commonly accepted belief in the HPC community is that because of the ever increasing gap between computing performance and storage throughput in modern supercomputers, writing all the data produced by a numerical simulation has become intractable and an in situ approach should be preferred. This in situ approach consists of intertwining simulation and analysis/visualization to process data as it is generated. Such processing can either use the same computing resources as the simulation or leverage additional resources. In the former scenario, data stays in memory and can be analyzed/visualized without any data movement, but this requires the simulation to effectively pause to run the analysis. In the latter scenario, data are staged to other resources, either cores on the same nodes as the simulation or on a distinct set of nodes, which allows the simulation and analysis to run concurrently. Hybrid methods combine the two approaches; a short processing phase is performed inline with the simulation to leverage data locality before staging data to other resources to perform additional complex operations.

In this section, we overview the hybrid in situ data analysis/visualization strategy that we devised for the XGC workflow. Our intent is real applicability; the chosen analyses/visualizations were motivated by the needs of the XGC application scientists themselves. Three additional workloads run: one for Poincaré puncture plot visualization (Section III-A); another to calculate a heat load analysis (Section III-B); and a third to compute diffusion (Section III-C). Section III-D comments on each specifically, including execution performance implications, explaining challenges and the approaches we adopt to address them. We aim to compose our workflow such that the inclusion of the analysis does not incur significant additional cost to the XGC simulation time.

A. Poincaré Puncture Plot

Efficient fusion reactor design requires the magnetic confinement of the plasma inside the tokamak. Because of this, analysis tools for understanding the dynamic nature of the magnetic field are critical. The complexity of the three-dimensional magnetic field lines makes analysis and visualization difficult. Because the field lines are periodic, this complexity can be reduced by using a Poincaré magnetic field-line puncture map [21]. The Poincaré map is the intersection of a field line with a lower dimensional subspace (called the Poincaré section). In our case, the Poincaré section is a two-dimensional plane that is perpendicular to the axis of the tokamak. Given a set of magnetic field lines, the Poincaré map, or intersections of magnetic field lines with the plane provides a concise representation of the magnetic field that is easier to understand and analyze.

In practice, the Poincaré map is generated by creating a large number of field lines and plotting each intersection, or puncture, with the plane. After a sufficient number of punctures have been collected, patterns in the map characterize the features in the magnetic field. The pseudo-code in Algorithm 1 describes how the Poincaré map is created. The field lines are computed by advecting a number of massless particles through the magnetic field. The intersections generated from a single particle characterizes the features of the magnetic surface at that position. The particles are advected using a differential equation solver, such as the 4^{th} order Runge-Kutta scheme (RK4). In order to characterize the magnetic field, a large

number of particles must be intersected with the perpendicular plane. Because of this, the computation of a Poincaré map can be very expensive. However, the trajectory of each particle needed to compute a Poincaré map is independent from all other particles. Hence, the calculation of can be parallelized over each particle.

```
Data: Plane, VecField, Field line initial positions P
for i ← 1 to N do
    p ← P(i)
    NumberOfPunctures ← 0
    while NumPunc < MaxNumPunc do
        pNext ← RK4Solve(p, VecField)
        Dir ← pNext - p
        Ray ← ray at p in direction Dir
        if Ray intersects Plane then
            P_i ← Intersection of Ray with Plane
            NumPuncs ← NumPuncs + 1
            Record P_i
        end
        p ← pNext
    end
end
```

Algorithm 1: Poincaré map algorithm.

Our implementation uses VTK-m [22], which is a toolkit for efficient and portable analysis and visualization on many-core processors. VTK-m provides a set of data parallel visualization primitives that allows a straightforward mapping of an execution task to data [23]. For the Poincaré map, the execution task is the inner loop in Algorithm 1 that computes the field lines using an RK4 integrator and performs the intersection with the plane. This execution task is applied, in parallel, to all of the particles. This particular task is well suited to running on a GPU because each particle and execution task can be placed in a thread. Peak efficiency can be achieved if there are enough execution threads to keep the GPU busy. The performance of the algorithm is governed by two factors, one minor and one major. The minor factor is the number of particles. VTK-m will place as many particles on the GPU as possible. If the number of particles exceeds the number of threads on the GPU, the particles will be processed in batches. The major factor is the number of punctures that are computed. The most costly operation in the algorithm is the RK4 integration step and the number of times it must be performed is directly proportional to the number of punctures computed.

B. Heat Load Calculation

The divertor heat load is the thermal energy deposition from the plasma to the divertor plates. Estimating the width of heat load on the divertor plates is critical for fusion reactor design and operation [24], [25]. With kinetic simulations and statistical analysis, one can estimate the contributions to the heat load profile, Γ_{heat}, from particle motion, and the origin of the divertor heat load can be identified from the particle trajectories. In collaboration with XGC researchers, we have been developing a fine-grained, time-varying, origin-specific heat load analysis method based on particles. The heat load to the material wall is calculated as

$$\Gamma_{\text{heat}} = \sum_{P_i \in \Psi_j, A_o} w_i E(P_i)/\Delta t \Delta S, \tag{1}$$

where $E(P_i)$ represents the kinetic energy of particle P_i with weight factor w_i when it hits the divertor at segment Ψ_j from origination angle segment A_o. Δt and ΔS are the time interval and the area of the segment Ψ_j. Here, the origin means the location of the particles when it escaped the separatrix.

The approach involves a tracking process, which identifies particles in the relevant parameter space of the tokamak and saves their information when it escapes the separatrix. When the escaped particles are identified, their information is pushed to a database and later retrieved when the tagged particles hit the divertor. The size of this database could grow to hundreds of GBs, or even TBs at scale. While the heat load computation itself is simple, involving only a modest number of boolean operations and summations, the communication costs and the highly distributed nature of the particles in XGC present challenges. The database operations cannot be local-only because which node an XGC particle is hosted on changes during load balancing operations. Furthermore, the number of tagged particles will not be constant between different XGC processes, which leads to imbalanced particle searching. Ultimately all processes synchronize to complete the calculation, meaning a significant number of the processes might be idling due the balance asymmetry.

To mitigate the cost of a full inline heat load calculation inside XGC, we have developed a method for use on external analysis nodes, where dedicated particle tracking processes run in parallel, outside the simulation. XGC tags the particles, then asynchronously writes them with ADIOS, to be ingested by the heat load analysis application, which identifies the particles to sum over, then computes the final quantity. This algorithm is detailed in Algorithm 2.

```
Data: N tagged particles at step t
Result: Heat load calculation
Heatload ← 0
for i ← 1 to N do
    if P_i is escaped then
        UpdateParticleDB(P_i, t)
    end
    if P_i is in divertor then
        P_esc, A_o ← SearchParticleDB(P_i, t)
        Γ_{heat,A_o} ← HeatLoadCalc(P_esc, A_o)
    end
end
// Distribute and merge Particle DBs
AllGather(ParticleDB)
```

Algorithm 2: Heat load calculation algorithm.

C. Diffusion Calculation

Particle dynamics in the edge plasma are of great interest to the fusion community. Turbulent eddies associated with

large-amplitude plasma pressure perturbations are common phenomena and are believed to mediate a significant amount of plasma transport across the magnetic field lines onto plasma facing components [26], [27]. Detailed observations of the plasma particles contributing to this transport channel would allow better characterization of its statistical properties. Moreover, by estimating transport properties, such as diffusion coefficients, these high-fidelity simulations could inform low-fidelity edge plasma fluid modeling. Low-fidelity simulations are regularly performed for predictive modeling of plasma discharges, as well as in other situations where operational constraints do not allow high-fidelity plasma simulations.

Dispersion of an ensemble of particles is measured by the mean-squared displacement:

$$\mathrm{MSD} = \langle (\Delta r - \langle \Delta r \rangle)^2 \rangle, \tag{2}$$

where $\langle \cdot \rangle$ denotes an ensemble average and Δr denotes a particle's change in position over a unit time step. The scaling of the MSD with time characterizes the dispersion characteristic of a particle ensemble. The case where MSD increases linearly with time, $\mathrm{MSD} \sim t$, is called diffusion, and a dependency $\mathrm{MSD} \sim t^{\alpha}$ with α greater or less than unity is called super- or sub-diffusive, respectively.

Considering only cross-field motions, a radial displacement is calculated as $\Delta r = \left(\frac{1}{RB_{\mathrm{pol},0}} + \frac{1}{RB_{\mathrm{pol},1}} \right) \frac{\Psi_1 - \Psi_0}{2}$, where R denotes the plasma major radius, B_{pol} denotes the poloidal component of the background magnetic field, Ψ is the normalized radial magnetic flux, and the numerical subscripts indicate whether the value is taken at the start or the end of the unit time step.

In XGC simulations, the MSD in Equation 2 are calculated for particles that cross from the confined plasma region into the scrape-off layer at a single given time step. Once the simulation reaches this time step, a flag is set in the particle structure of every particle that just crossed the separatrix. In addition, the triangle of the simulation grid where the particle crossed the separatrix is stored. In a following step, this information aids in reconstructing grid quantities using the MSDs. Over the next T subsequent time steps, these flagged particles accumulate via

$$\langle \Delta r \rangle = \sum_{t=1}^{T} \bar{w}_t^{\mathrm{p}} \Delta r_t^{\mathrm{p}} / \sum_{t=1}^{T} w_t^{\mathrm{p}} \tag{3}$$

$$\langle (\Delta r)^2 \rangle = \sum_{t=1}^{T} \bar{w}_t^{\mathrm{p}} \left(\Delta r_t^{\mathrm{p}} \right)^2 / \sum_{t=1}^{T} w_t^{\mathrm{p}}, \tag{4}$$

where $\bar{w}_t^{\mathrm{p}}$ is the geometric weight of $w_0 w_1$ at the beginning and the end of the given time-step. This method is described in Algorithm 3.

While the MSD is a particle quantity, transport coefficients such as diffusion coefficients are fluid quantities and are defined on the simulation grid. In order to gather relevant particle information over the entire grid, our implementation divides the necessary reductions into two separate steps. On the simulation side, every MPI rank accumulates $\langle \triangle r \rangle$ and $\langle (\triangle r)^2 \rangle$ into bins which map to the simulation grid triangles where a particle crossing was registered. This calculation is described in Algorithm 4.

Data: AoSoA of marker particles
Result: AoSoA of marker particles with updated fields Δr and $(\Delta r)^2$
for $p \leftarrow 1$ **to** num_ptl **do**
  if particle[p] *is flagged* **then**
    update particle[p].Δr, Eq. (3);
    update particle[p]. $(\Delta r)^2$, Eq. (4);
  end
end

Algorithm 3: During the particle push, flagged particles accumulate 3 and 4.

Data: AoSoA of marker particles
Result: dr[num_tri] and $dr2$[num_tri]
for $p \leftarrow 1$ **to** num_ptl **do**
  if particle[p] *is flagged* **then**
    ptl_crossed[particle[p].tri_crossed] += 1 ;
    dr[particle[p].tri_crossed] += particle[p].Δr ;
    dr2[particle[p].tri_crossed] += particle[p]. $(\Delta r)^2$;
  end
end
for $i \leftarrow 1$ **to** num_tri **do**
  if ptl_crossed[i] > 0 **then**
    dr[i] /= ptl_crossed[i] ;
    dr2[i] /= ptl_crossed[i] ;
  end
end

Algorithm 4: During the diagnostic time step, flagged particles deposit their accumulated Δr and $(\Delta r)^2$ to the triangle where they crossed the separatrix.

The triangle data is then sent to the data analysis node. At the data analysis node, the accumulated MSDs, binned into their respective triangle, are again averaged over triangles. The resulting quantities are Eqs. (3) - (4), per triangle and averaged over the total number of particles that crossed the separatrix at that triangle. Our hybrid scheme effectively offloads an all-reduce across the MPI ranks from the simulation to the analysis node, as illustrated in Algorithm 5. This comes at the cost of transferring a table to the data analysis node, whose number of rows is given by the number of triangles where particles crossed the separatrix, with three columns: a triangle number, as well as $\langle \Delta r \rangle$ and $\langle (\Delta r)^2 \rangle$, both averaged over an ensemble of particles.

D. Execution Strategy

Computing Poincaré maps for XGC could be carried in several ways, with variations of inline analysis or offloaded analysis on a separate node.

The first method, synchronous inline in situ analysis, assumes that after each XGC time step, the Poincaré analysis

Data: dr[num_tri] and $dr2$[num_tri] for each MPI rank
Result: dr[num_tri] and $dr2$[num_tri]
$dr \leftarrow \text{MPIAllreduce}(dr)$;

Algorithm 5: The analysis node combines triangle-deposited $\langle \Delta r \rangle$ and $\langle (\Delta r)^2 \rangle$ from all MPI ranks.

will be computed on the same nodes. Field lines from the initial particles will be generated and intersected with the perpendicular plane. Because the information for the magnetic field is spatially distributed across the set of computational nodes, communication is required when a particle exits the spatial boundary. This method is inefficient for two main reasons. First, the particles to be advected will be distributed across a large number of simulation nodes. As such, the particle to GPU ratio will be very low. Second, communication at the scale of the simulation will be very inefficient.

The second method outsources the visualization synchronously. After each XGC time step, the magnetic field data will be communicated to a small set of the simulation nodes where the Poincaré map will be computed. This improves the particle to GPU ratio considerably and reduces communication, which will considerably improve the analysis efficiency. However, the rest of the simulation nodes will be blocked until the analysis is complete, which is wasteful.

Another method is asynchronous in transit in situ analysis. After each XGC time step, the magnetic field data will be transferred to a separate set of nodes. Once the transfer has completed, the Poincaré map can be generated, while XGC continues the simulation. Since the computation of the Poincaré map may take longer than an XGC time step, multiple GPUs can be used in a round robin fashion to calculate each time step.

The key to efficiency for algorithms that run on a GPU is to keep the GPU saturated with work. The first method does not achieve this, as the particle to GPU ratio is low, and it requires communication. When attempting an initial implementation, this approach stalled each step with visualization more than an order of magnitude compared to the XGC step time. The second method can be efficient because the particle to GPU ratio is high, but it still stalls XGC. The third provides a high particle to GPU ratio and does not stall the simulation, which is why we chose this approach.

The heat load and diffusion analyses both require synchronization between all nodes. In the case of the heat load, an AllGather at the end of Algorithm 2. In the case of diffusion, the AllReduce at the end of Algorithm 5. As with the Poincaré code, the analysis codes can be run inline or offloaded on a separate node. While the computational parts are simple for both codes and can easily be executed inline, the reductions needed at every step can present challenges when running inline.

The heat load analysis keeps a memory of all the tracked particles in a ParticleDB. During an XGC run there can be up to 10^7 tracked particles, requiring a total of 16 GB of data for each stored step. The average lifespan of a tracked particle is $\sim$ 100 steps, which puts an upper bound of 1.6 TB for the ParticleDB memory. If each process is storing a local view of the database, which can grow with every time step towards this upper bound, the available memory per node could be exceeded. One solution is to load balance the database every few steps, but this will add an extra cost to the execution of XGC. By offloading the analysis to separate nodes, the load balancing is external and done during the external AllGather in Algorithm 2. In addition, since the Poincaré code leverage the GPUs and is not memory intensive, the heat load code can run on the CPUs of the same compute nodes.

The Diffusion analysis code is composed of three steps: i) the accumulation of $\langle \triangle r \rangle$ and $\langle (\triangle r)^2 \rangle$ into bins which map to the simulation grid triangles where a particle crossing was registered; ii) a reduction of all the bins across all the processes; and iii) the averaging of MSDs over triangles. The entire execution could be done inline and use an AllReduce for the reduction phase. While this is not expensive at small scale it could become more significant when a large number of processes are used. It is not particularly memory intensive, so we co-locate it with the heat load analysis in our workflows.

IV. Experimental Results

Our experimental tests consist of XGC fusion simulations and associated in situ analysis/visualization of the data. Section I outlined the necessity of more robust tokamak edge simulations to better understand the operating regime of ITER. Accordingly, we configure XGC to study electromagnetic turbulence in the vicinity of of the magnetic separatrix. The in situ analysis processes are those explained in Section III: the Poincaré puncture plot visualization, along with the heat load and diffusion calculations. All jobs are composed and executed using the EFFIS workflow system [17], whose primary design target is workloads of the variety executed in this paper.

We perform our simulation and analysis on the Oak Ridge Leadership Computing Facility (OLCF) supercomputer Summit. Summit is a 200 PF system of 4,600 nodes, where each nodes consists of 6 NVIDIA Tesla V100 GPUs and 2 IBM POWER9 CPU processors, for a total of 42 cores per node; 512 GB of DDR4 memory is available for use by the CPUs, with 96 GB of High Bandwidth Memory for the accelerators. Nodes are interconnected by a Mellanox EDR 100G InfiniBand network of non-blocking fat tree topology. Summit mounts an IBM Spectrum Scale parallel file system called Alpine, which consists of 77 storage server nodes, with a total capacity of about 250 PB and a maximum theoretical bandwidth of 2.5 TB/s for sequential I/O.

Our measurements begin with a set of XGC scaling runs, on 256, 512, and 1,024 nodes. XGC uses the Summit GPUs, placing one MPI process for each of a node's 6 GPUs, with threading parallelization across the multiple cores. XGC's mesh is constant between runs, with the total particle number multiplying identically with the node count scaling, which amounts to a form of *weak scaling*. We present XGC performance, with time step breakdowns, beginning in Section IV-A.

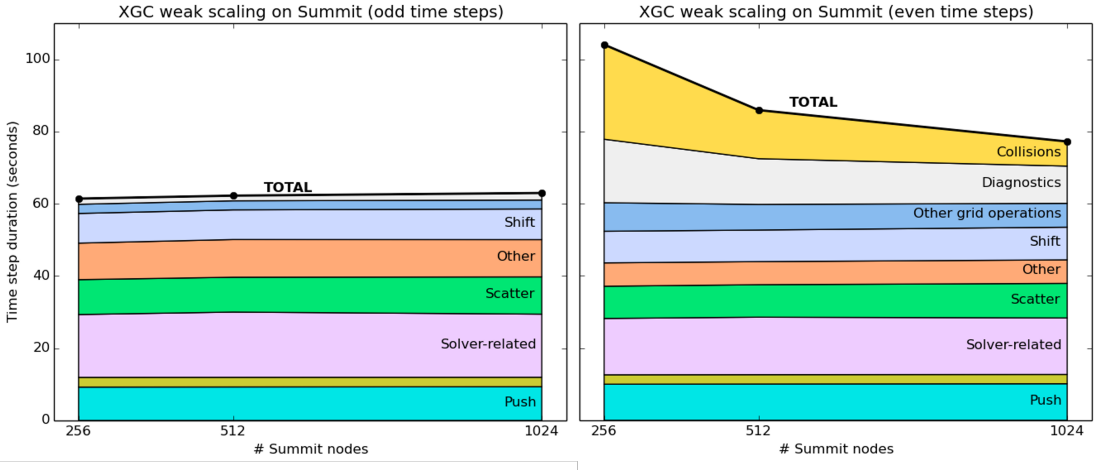

Fig. 2. Breakout by main PIC code phases of the average XGC time step duration on Summit when the number of nodes increases (weak scaling). Odd (left) and even (right) time steps are distinguished.

In each node case, the run is repeated five times, incrementally adding some or all the in situ analysis to the workflow: (1) first running XGC only, (2) next running XGC and the heat load calculation, (3) then running XGC and the diffusion calculation, (4) then running XGC and the Poincaré visualization, and (5) finally, running XGC with all three. In the final case, the Poincaré puncture processing is dedicated its own node, with the heat load and diffusion executables sharing one separate node. Otherwise, any analysis runs on its own single node; XGC is never co-located with them. The heat load executable runs with 5 MPI processes and 7 threads each; the diffusion application with 7 processes of a single thread. XGC executes for 10 time steps, outputting the data for analysis using ADIOS. In each of the five cases, I/O transport for the analysis data has been configured using either files or RDMA staging, and we observe file system/network variance on the shared system. Results for these workflows are presented and discussed in Section IV-C, including an analysis of the resource cost added. In addition, Section IV-B presents scaling performance of each of the analysis/visualization processes themselves; the puncture plotter's performance study requires additional, single-node jobs to understand its time dependence, which are described therein.

While our performance results are aggregated across XGC jobs limited to 10 time steps, our analysis contributions include their applicability across much longer runs, which are needed in real physics studies. It was not cost-feasible for us to run long time baselines for all our experiments, but we include plot figures of the analyses applied to data from related runs provided by the XGC team. These are found in Section IV-D.

A. XGC Performance

Before assessing the performance and impact of adding analysis/visualization to the execution of XGC, we analyze the performance of the fusion code itself. We performed a weak scaling analysis of XGC in which the number of particles increases linearly with the number of nodes, but the mesh remains the same for the different node counts. This results in a super-linear speedup of the execution times. Figure 2 illustrates the causes of this speedup. It details the time spent in each of the main phases of a PIC code. While the time of most phases remain constant as the number of nodes increases, that of the "Collisions" and "Diagnostics" phases decreases. This is because the time complexity of these particular phases is directly related to the size of the mesh and not the number of particles in each cell. Moreover, the "Collisions" phase is only executed every other step, and "Diagnostics" only has a small odd-step time. "Diagnostics" is where XGC computes analysis internally and where output occurs, most of these calculations and writes occurring only every other step. The odd-time "Diagnostics" have been added by the work in this paper, a component in facilitating the offload of the heat load and diffusion calculations to the external analysis nodes. What is needed for the Poincaré plotting is in the usual output occurring every other step.

While our I/O offload strategy ensures that in situ analysis does not pause the simulation progress in the way inline analysis would, it is still preferable not incur significant additional job resource cost; for example, if an analysis application runs longer than the simulation. Even if the in situ analysis itself only occupies up to two nodes, the job is still charged for the XGC nodes, too. Understanding the analysis performance and workflow resource costs occupies the next two subsections.

B. Analysis Performance

Here, we present the respective performance of our three in situ analysis applications: Poincaré puncture plot visualization, heat load width estimation, and particle diffusion calculation. These profiles are key to better understand how to optimize the design of our complex workflow involving cutting edge XGC simulations and the analysis/visualization of data produced by it. For instance, it allows us to consider if specific analysis should be inline with XGC in the future or how much node sharing is appropriate between the three. In the case of the

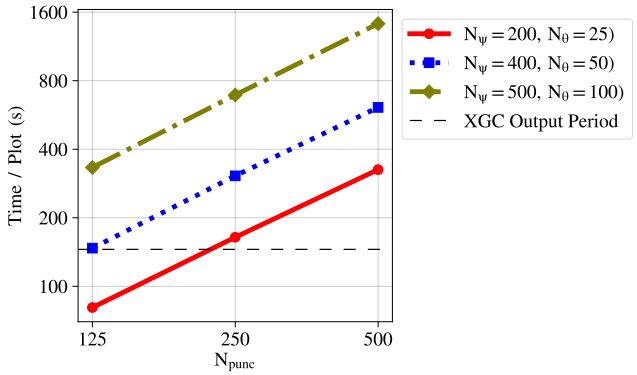

Fig. 3. Poincaré plot GPU performance. Note, both axes are log_2 scaled.

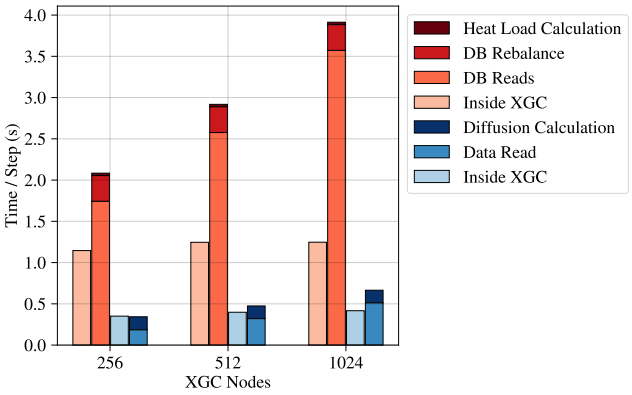

Fig. 4. CPU time spent in the heat load and diffusion hybrid analyses. The lightest shaded bars are the times added to XGC itself to accommodate the analysis, while the remainder are measured from the in situ applications running on the separate data analysis node.

puncture visualization, there is also a question of how to set the parameters affecting the output plot resolution.

The Poincaré puncture plotting only requires data quantities that are discretized over the XGC grid, not any simulation particle data, and hence should not systematically depend on the XGC node count scaling. However, it includes a few parameters that impact the visual fidelity of the output visualization and that significantly affect the application's execution time – N_{punc}: the number of punctures to compute; N_ψ: the number of points chosen uniformly along the radial magnetic axis; and N_θ: the number of points chosen uniformly along the toroidal axis. Here, we do not have any hard requirements on the minimum settings, but increasing them is better for improved figures. See Figure 7 in Section IV-D for an example output of an $N_{punc} = 500$, $N_\psi = 500$, $N_\theta = 100$ output.

Figure 3 plots the time dependence over the three parameters. Each point in the plot has been generated as an average over 256, 512, and 1,024 node XGC data sizes, whose variation between sizes is not visible on the figure; a single choice between them would have sufficed. What is observed in Figure 3 is linear scaling with $\mathrm{N_{punc}}$ along the x-axis, and roughly $\log_2(\mathrm{N}_\psi \mathrm{N}_\theta)$ scaling between the different lines. This parameter space informs how to configure the application in an XGC run. In our case, the puncture analysis runs every other step because that is the XGC output cadence for the data it needs; we conservatively used settings for $(N_{punc}, N_\psi, N_\theta)$ at the bottom left of the plot, which is less than the sum of two steps in Figure 2 (that is, Figure 7's dashed horizontal line). However, we note that our implementation for this analysis runs the heavy calculation on GPUs, dispatching a new data step to a process attached to a single accelerator in a round robin fashion as the new data steps become available. Figure 3 measures the time for one execution time on the single GPU. Parallelized over six GPUs on a Summit node, and even multiple nodes if desired, the visualization processing could keep up even if a step takes longer than an XGC step (possibly cutting off the processing for the latest steps of XGC and computing those post-job). Still, even time values toward the bottom left of Figure 3 would be a significant inline cost to XGC; it would parallelize some over the GPUs of the many XGC nodes, but as explained in Section III-D, performance depends on keeping the GPU saturated, and breaking the work over many nodes would not simply divide the time identically, i.e. indefinite downward strong scaling does not hold. Not incurring this cost is why we have exported the analysis to another node, and Figure 3 supports that this approach can flexibly keep up.

Figure 4 plots performance times for the heat load and diffusion calculations, which run on the CPUs. Included in the plot are the times XGC itself spends for the offload. For the diffusion analysis, the results indicate that the full calculation could reasonably be inlined into XGC without too much penalty. Its data read time can be thought of as upper limit on the additional communication time that would be incurred, (because MPI gathering will outperform ADIOS), and the calculation itself is not heavy either. Less than one second per step total is tolerable. On the other hand, little harm results from the offload. Not much time was spent in the XGC offload phase, whose data movement cost would be diminished for a fully inline version (again, because MPI will outperform ADIOS.) The heat load calculation shows a somewhat higher cost, approaching 5% of the XGC step time. This would decrease if we dedicated more than a single node to the heat load application or if it were inlined to XGC and parallelized across the many running processes. The problem is the high fraction of the time spent in database operations. Communication cost will not simply divide with node count. While an upper limit of four seconds is not an enormous inline cost to potentially add, at any rate, the one second offload is sustainable.

C. Workflow Performance

We now consider overall workflow performance for our job loads: variations in the coupling of XGC to the analyses/visualizations and the costs associated in doing so. Figure 5 examines the suite of 256, 512, or 1,024 node workflows with file-based analysis data offload. Each node count considers

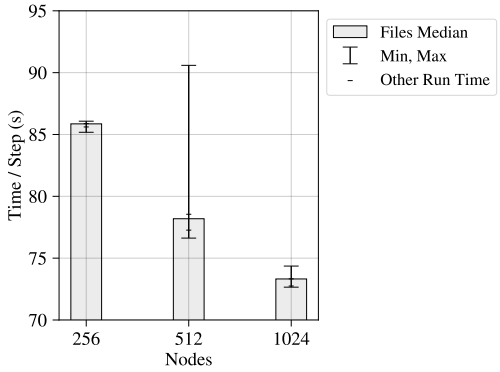

Fig. 5. XGC performance, using file-based analysis data offload. We note that the time spent in XGC can have a significant amount of variance due to the network and file system variability, which occurs in shared HPC environments.

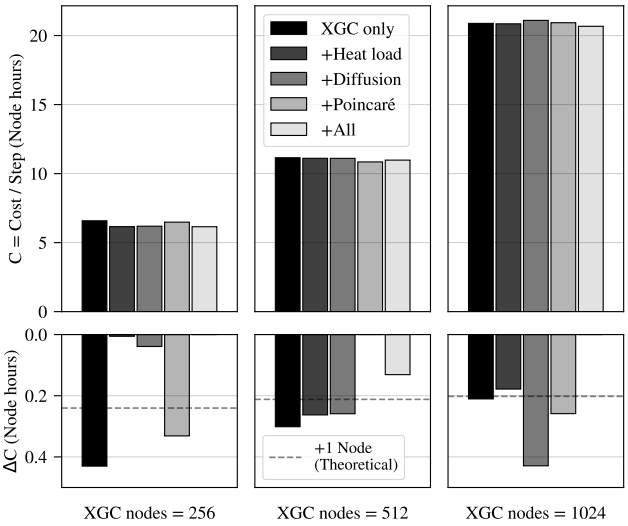

Fig. 6. Workflow job costs with staged data movement. Little additional cost is incurred by adding the in situ analysis. It is often outweighed by system variation, cf. Figure 5.

the five job types we described in the introduction to Section IV: XGC only, XGC plus any of the three single analyses individually, and XGC with all three. The bars indicate the variation between the runs, as a result of the file system and network contention on the shared supercomputer. The largest outlier is 512 nodes, enough to outweigh the super-linear weak scaling. File system variance is not unexpected; shared parallel file systems are known to be vulnerable to cases of I/O overload, where large slowdowns are observed during write operations [28], [29], [30], [31]. XGC operation includes a number of file outputs, (among them are those needed for our in situ analyses), used for many different types of I/O operations, such as simple diagnostic output as text. All this file output impacts all cases in Figure 5 and contributes to the total variance. Network contention is another factor, but is not expected to cause as large of outliers on its own, and impacts both the file and staging output because both use the system interconnect. In the cost analysis of Figure 6 below, we consider only the staged analysis data, as it is the modestly more robust option against large variation outliers. None of the conclusion are impacted in making this choice, but it affords a simplified figure that is easier to read.

Figure 6 measures the job cost of running the five workflows, with the lower panels indicating the difference compared to the lowest in a node group. The results indicate that little extra cost is incurred by adding our in situ analysis to the workflow. Theoretically, it should be the XGC step time multiplied by one in first three analysis cases or multiplied by two in the case with all analyses – that is, the number of one or two extra nodes that have been requested for the job. However, the system variation is typically the larger effect, and in many cases, the jobs with analysis happen to cost less than the XGC-only jobs. Figure 6 represents our foremost performance result, and Section IV-D continues by discussing the physics implications of these analysis results that can be deployed at little resource cost.

For our large job launches, we made the initial decision to separate the Poincaré plotting onto and single node, with the diffusion and heat load calculations on a different node. However, this was not necessary; they could all be co-located. While we were unable to rerun all the XGC jobs again, (which are the computationally expensive pieces of the workflows), we did conduct follow up tests with all three analysis/visualizations co-located, using the data that had been saved from the file-based transport runs, to confirm error-free performance consistency with what has been presented in Figure 3 and Figure 4. However, we choose to include the slightly more pessimistic cost addition of two extra nodes in Figure 5, as that is how the fully online version was run.

The puncture plot GPU visualization is expensive, and best suited to be offloaded to analysis nodes rather than inlined inside XGC. If this analysis is to be included in the workflow, the diffusion and heat load pieces might as well be done on those nodes as well, making use of the free CPUs. While the heat load piece is possibly memory intensive enough to require more than a single node (Section III-D), we did not actually encounter a scenario where it caused an out of memory error in the co-located executables on the same node; more than one node may be preferred for the visualization anyway for increased output fidelity. Post-hoc analysis is an option, but comes at the cost of subideal lag until after the conclusion of a potentially days-long XGC run, where the Poincaré computation could be lengthy too for many time steps and/or high image resolution.

D. Physics Results

Figure 7 is a snapshot Poincaré puncture plot of the turbulent magnetic field lines, obtained from our in situ visualization application, after an XGC run of about 500 time steps ($\sim$ 7 ms) for an ITER plasma modeled for a stationary energy-production state without violent MHD instabilities. We zoom in on the X-point area of the plane. One observes

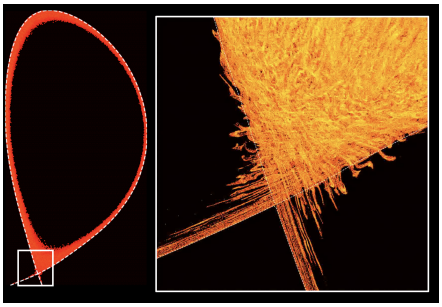

Fig. 7. A snapshot Poincaré puncture plot of the fluctuating magnetic field at the edge of ITER plasma. The plot on the left shows the entire poloidal plane and the image on the right shows a zoomed in region of the highlighted area.

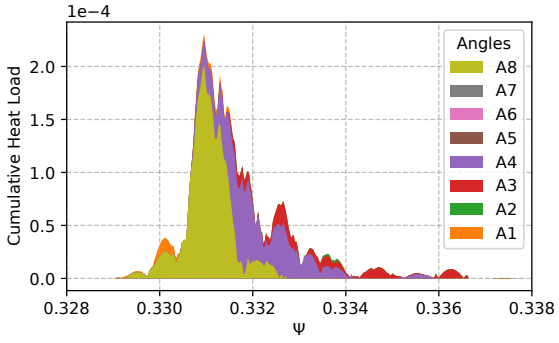

Fig. 8. Heat load footprint analyzed online with turbulence-consistent particle tracing in XGC. Different colors designate particles originating from different poloidal angles on the separatrix.

turbulent tangled magnetic field lines. Identifying, visualizing, and analyzing features of this kind are essential; more study is needed. They are suspected to contribute significantly to the plasma's consequential loss to the material wall, and to the plasma leakage in ITER from inside the stable separatrix surface, which has been assumed to be the last confinement surface in stationary fusion-producing operation. If this visualization were to be generated inline within the XGC nodes, as has been previously considered, due to non-scalability of the visualization process, the job would have taken days longer.

Figure 8 demonstrates physics results computed from the heat load footprint analysis application, after about 600 ITER simulation steps. Different colors designate particles originating from different poloidal angles at the last confinement surface (separatrix). For the first time, this online analysis allows application physicists to study from where the peak heat-load originated. Figure 8 indicates that the highest density heat load is from the separatrix-crossing particles around the poloidal angle A8. This type of analysis informs fusion scientists to closely monitor turbulence, such as turbulent homoclinic tangles, around the poloidal angle A8.

Finally, Figure 9 shows the time-dependent mean squared displacement of electrons and ions across the magnetic surfaces, which are traveling toward the divertor plates, after 10 ms of physical time simulation. In each curve, one point is calculated per simulation time step, with the individual particle

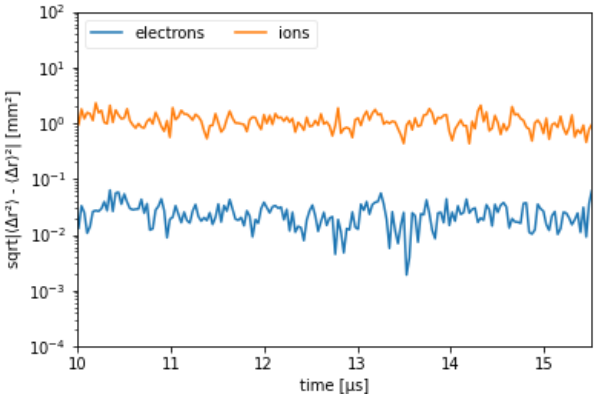

Fig. 9. Time evolution of the mean square displacement of plasma electrons and ions across the magnetic surface that are on the to the divertor plates.

displacement data obtained at each time evaluated inline in XGC, then sent to the analysis node for online statistical handling, visualization, and physics discovery.

V. Conclusion

In the next decade, ITER's objective is to realize plasmas that are well beyond the currently accessible physics regimes in present-day tokamak experiments. To best understand the complicated multi-physics in the edge of the ITER plasma at the most fundamental level, one needs specialized 5D gyrokinetic simulations, which are currently only possible with the XGC code. As more physics have been added to simulations to study the possible damage to the divertor plates from exhaust heat, new data outputs and analysis of those outputs are needed for the measurements. Our team has inserted code directly into XGC, as well as used additional nodes for running complex analysis, to form a hybrid staging approach to the problem.

In this work, we have examined three important data analysis additions to the XGC results: Poincaré puncture plots, heat load footprint computation, and particle diffusion calculation. Because the Poincaré plots are very expensive, the analysis is most seamlessly executed by staging to a few extra nodes and using a round robin approach to keep up with the simulation, making full use of the GPUs on Summit. Furthermore, the CPUs on these nodes can then be used to perform the heat load and particle diffusion calculations. Importantly, the extra analysis nodes come at a much lower additional resource cost compared to that of the XGC workload itself. By creating this infrastructure, we are able to examine complicated physics results, which may not have been possible without this infrastructure. For example, our work on online visualization of turbulent homoclinic tangles allows fusion scientists to better understand and potentially improve the turbulence spread of plasma exhaust heat. The physics of the turbulent homoclinic tangle will be reported in a future physics publication and will fully utilize the online framework described in this paper. While we have focused on an XGC-specific case, the hybrid staging approach is more generally applicable; similar

considerations have been discussed elsewhere, e.g. for laser-plasma with WarpX and analytics [32].

VI. Acknowledgments

This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725.

This research was supported by the Exascale Computing Project (ECP), Project Number: 17-SC-20-SC, a collaborative effort of two DOE organizations—the Office of Science and the National Nuclear Security Administration—responsible for the planning and preparation of a capable exascale ecosystem—including software, applications, hardware, advanced system engineering, and early testbed platforms—to support the nation's exascale computing imperative.

References

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

[2] W. F. Godoy, N. Podhorszki, R. Wang, C. Atkins, G. Eisenhauer, J. Gu, P. Davis, J. Choi, K. Germaschewski, K. Huck, A. Huebl, M. Kim, J. Kress, T. Kurc, Q. Liu, J. Logan, K. Mehta, G. Ostrouchov, M. Parashar, F. Poeschel, D. Pugmire, E. Suchyta, K. Takahashi, N. Thompson, S. Tsutsumi, L. Wan, M. Wolf, K. Wu, and S. Klasky, "ADIOS 2: The Adaptable Input Output System. A framework for high-performance data management," *SoftwareX*, vol. 12, p. 100561, 2020.

[3] The HDF Group. (1997-NNNN) Hierarchical Data Format, version 5. Https://www.hdfgroup.org/HDF5/.

[4] M. Ainsworth, O. Tugluk, B. Whitney, and S. Klasky, "Multilevel techniques for compression and reduction of scientific data—the multivariate case," *SIAM Journal on Scientific Computing*, vol. 41, no. 2, pp. A1278–A1303, 2019.

[5] ——, "Multilevel techniques for compression and reduction of scientific data-quantitative control of accuracy in derived quantities," *SIAM Journal on Scientific Computing*, vol. 41, no. 4, pp. A2146–A2171, 2019.

[6] Q. Gong, X. Liang, B. Whitney, J. Y. Choi, J. Chen, L. Wan, S. Ethier, S.-H. Ku, R. M. Churchill, C.-S. Chang *et al.*, "Maintaining trust in reduction: Preserving the accuracy of quantities of interest for lossy compression," in *Smoky Mountains Computational Sciences and Engineering Conference*. Springer, Cham, 2021, pp. 22–39.

[7] H. Childs, S. Ahern, J. Ahrens, A. Bauer, J. Bennett, E. W. Bethel, P.-T. Bremer, E. Brugger, J. Cottam, M. Dorier, S. Dutta, J. Favre, T. Fogal, S. Frey, C. Garth, B. Geveci, W. Godoy, C. Hansen, C. Harrison, B. Hentschel, J. Insley, C. Johnson, S. Klasky, A. Knoll, J. Kress, M. Larsen, J. Lofstead, K.-L. Ma, P. Malakar, J. Meredith, K. Moreland, P. Navrátil, P. O'Leary, M. Parashar, V. Pascucci, J. Patchett, T. Peterka, S. Petruzza, N. Podhorszki, D. Pugmire, M. Rasquin, S. Rizzi, D. Rogers, S. Sane, F. Sauer, R. Sisneros, H.-W. Shen, W. Usher, R. Vickery, V. Vishwanath, I. Wald, R. Wang, G. Weber, B. Whitlock, M. Wolf, H. Yu, and S. Ziegeler, "A Terminology for in situ Visualization and Analysis Systems," *International Journal of High Performance Computing Applications*, vol. 34, no. 6, pp. 676–691, 2020.

[8] J. C. Bennett, H. Abbasi, P.-T. Bremer, R. Grout, A. Gyulassy, T. Jin, S. Klasky, H. Kolla, M. Parashar, V. Pascucci *et al.*, "Combining in-situ and in-transit processing to enable extreme-scale scientific analysis," in *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 2012, pp. 1–9.

[9] see https://www.iter.org.

[10] S. Ku, C. S. Chang, R. Hager, R. M. Churchill, G. R. Tynan, I. Cziegler, M. Greenwald, J. Hughes, S. E. Parker, M. F. Adams, E. D'Azevedo, and P. Worley, "A fast low-to-high confinement mode bifurcation dynamics in the boundary-plasma gyrokinetic code xgc1," *Phys. Plasmas*, vol. 25, p. 056107, 2018.

[11] "Summit User Guide," https://docs.olcf.ornl.gov/systems/summit_user_guide.html, accessed: 2022-07-31.

[12] B. Ludäscher, I. Altintas, S. Bowers, J. Cummings, T. Critchlow, E. Deelman, D. De Roure, J. Freire, C. A. Goble, M. B. Jones *et al.*, "Scientific process automation and workflow management." *Scientific Data Management*, vol. 10, no. 3, pp. 476–508, 2009.

[13] J. Cummings, A. Pankin, N. Podhosrzki, G. Park, S. Ku, R. Barreto, S. Klasky, C. Chang, H. Strauss, L. Sugiyama *et al.*, "Plasma edge kinetic-mhd modeling in tokamaks using kepler workflow for code coupling, data management and visualization," *Communications in Computational Physics*, vol. 4, no. 3, pp. 675–702, 2008.

[14] I. Altintas, B. Ludaescher, S. Klasky, and M. A. Vouk, "Introduction to Scientific Workflow Management and the Kepler System," in *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, Tampa, Florida, 2006.

[15] J. F. Lofstead, S. Klasky, K. Schwan, N. Podhorszki, and C. Jin, "Flexible io and integration for scientific codes through the adaptable io system (adios)," in *Proceedings of the 6th international workshop on Challenges of large applications in distributed environments*, 2008, pp. 15–24.

[16] J. Cummings, J. Lofstead, K. Schwan, A. Sim, A. Shoshani, C. Docan, M. Parashar, S. Klasky, N. Podhorszki, and R. Barreto, "Effis: an end-to-end framework for fusion integrated simulation," in *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*. IEEE, 2010, pp. 428–434.

[17] E. Suchyta, S. Klasky, N. Podhorszki, M. Wolf, A. Adesoji, C. Chang, J. Choi, P. E. Davis, J. Dominski, S. Ethier *et al.*, "The Exascale Framework for High Fidelity coupled Simulations (EFFIS): Enabling whole device modeling in fusion science," *International Journal of High Performance Computing Applications*, vol. 36, no. 1, pp. 106–128, 2022.

[18] J. Dominski, J. Cheng, G. Merlo, V. Carey, R. Hager, L. Ricketson, J. Choi, S. Ethier, K. Germaschewski, S. Ku *et al.*, "Spatial coupling of gyrokinetic simulations, a generalized scheme based on first-principles," *Physics of Plasmas*, vol. 28, no. 2, p. 022301, 2021.

[19] S. Di and F. Cappello, "Fast error-bounded lossy hpc data compression with sz," in *2016 ieee international parallel and distributed processing symposium (ipdps)*. IEEE, 2016, pp. 730–739.

[20] T. E. Evans, R. K. R. abd J. A. Carter, and B. I. Rapoport, "Homoclinic tangles, bifurcations and edge stochasticity in diverted tokamaks," *Contrib. Plasma Phys.*, vol. 44, pp. 235–248, 2004.

[21] A. Sanderson, G. Chen, X. Tricoche, D. Pugmire, S. Kruger, and J. Breslau, "Analysis of recurrent patterns in toroidal magnetic fields," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 6, pp. 1431–1440, 2010.

[22] K. Moreland, C. Sewell, W. Usher, L. Lo, J. Meredith, D. Pugmire, J. Kress, H. Schroots, K.-L. Ma, H. Childs, M. Larsen, C.-M. Chen, R. Maynard, and B. Geveci, "VTK-m: Accelerating the Visualization Toolkit for Massively Threaded Architectures," *IEEE Computer Graphics and Applications (CG&A)*, vol. 36, no. 3, pp. 48–58, May/Jun. 2016.

[23] K. Moreland, R. Maynard, D. Pugmire, A. Yenpure, A. Vacanti, M. Larsen, and H. Childs, "Minimizing Development Costs for Efficient Many-Core Visualization Using MCD^3," *Parallel Computing*, vol. 108, p. 102834, Dec. 2021.

[24] C. Chang, S. Ku, A. Loarte, V. Parail, F. Köchl, M. Romanelli, R. Maingi, J.-W. Ahn, T. Gray, J. Hughes, and et al., "Gyrokinetic Projection of the Divertor Heat-Flux Width from Present Tokamaks to ITER," *Nuclear Fusion*, vol. 57, no. 11, p. 116023, Aug 2017.

[25] C. Chang, S. Ku, R. Hager, R. Churchill, J. Hughes, F. Köchl, A. Loarte, V. Parail, and R. Pitts, "Constructing a New Predictive Scaling Formula for ITER's Divertor Heat-Load Width Informed by a Simulation-Anchored Machine Learning," *Phys. Plasmas*, vol. 28, p. 022501, 2021.

[26] B. LaBombard, R. L. Boivin, M. Greenwald, J. Hughes, B. Lipschultz, D. Mossessian, C. S. Pitcher, J. L. Terry, and S. J. Zweben, "Particle transport in the scrape-off layer and its relationship to discharge density limit in alcator c-mod," *Physics of Plasmas*, vol. 8, no. 5, pp. 2107–2117, 2001. [Online]. Available: https://doi.org/10.1063/1.1352596

[27] R. Kube, O. Garcia, A. Theodorsen, A. Kuang, B. LaBombard, J. Terry, and D. Brunner, "Statistical properties of the plasma fluctuations and turbulent cross-field fluxes in the outboard mid-plane scrape-off layer of alcator c-mod," *Nuclear Materials and Energy*, vol. 18, pp. 193–200, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S235217911830187X

[28] J. Lofstead, F. Zheng, Q. Liu, S. Klasky, R. Oldfield, T. Kordenbrock, K. Schwan, and M. Wolf, "Managing variability in the io performance of petascale storage systems," in *SC'10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2010, pp. 1–12.

[29] B. Xie, J. Chase, D. Dillow, O. Drokin, S. Klasky, S. Oral, and N. Podhorszki, "Characterizing output bottlenecks in a supercomputer," in *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 2012, pp. 1–11.

[30] Q. Liu, N. Podhorszki, J. Logan, and S. Klasky, "Runtime i/o re-routing + throttling on hpc storage," in *5th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 13)*, 2013.

[31] L. Wan, M. Wolf, F. Wang, J. Y. Choi, G. Ostrouchov, and S. Klasky, "Analysis and modeling of the end-to-end i/o performance on olcf's titan supercomputer," in *2017 IEEE 19th International Conference on High Performance Computing and Communications; IEEE 15th International Conference on Smart City; IEEE 3rd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE, 2017, pp. 1–9.

[32] F. Poeschel, W. F. Godoy, N. Podhorszki, S. Klasky, G. Eisenhauer, P. E. Davis, L. Wan, A. Gainaru, J. Gu, F. Koller *et al.*, "Transitioning from file-based hpc workflows to streaming data pipelines with openpmd and adios2," in *Smoky Mountains Computational Sciences and Engineering Conference*. Springer, Cham, 2021, pp. 99–118.