Online and Scalable Data Compression Pipeline With Guarantees on Quantities of Interest

Tania Banerjee*, Jaemoon Lee*, Jong Choi†, Qian Gong†, Jieyang Chen‡,
Choongseok Chang§ Scott Klasky†, Anand Rangarajan*, Sanjay Ranka*
*University of Florida, USA
†Oak Ridge National Laboratory, USA
‡University of Alabama at Birmingham, USA
§Princeton Plasma Physics Laboratory, USA

***Abstract*—Data compression is becoming critical for data-intensive scientific applications. Scientists require compression techniques that accurately preserve derived quantities of interest (QoIs). Prior work has shown that a pipeline can be built to guarantee error on the primary data (PD) within user-defined bounds and achieve near-floating point QoI errors.**

In this paper, we present novel computational approaches for accelerating the pipeline and demonstrate results that enable concurrent execution of compression in parallel with the simulation nodes. This allows compression, including the writing of the required compression data, for the previous time step to be completed while the simulation proceeds with the current time step. Overall, the approach presented in this paper results in a 6-8 times improvement in computational overhead compared to previous work. These results were obtained using data generated by a large-scale fusion code called XGC, which produces hundreds of terabytes of data in a single day.

I. Introduction

The speed and quantity of data produced by scientific applications have been increasing faster than the growth rate of computing power, network, and storage capabilities [20]. To cope with this unprecedented growth, data reduction or compression techniques have become essential. While lossy data compression techniques have been successful in image and video processing, their direct applicability to scientific applications is limited. In scientific applications, it is crucial for derived quantities from primary data (PD), known as quantities of interest (QoIs), to have quantifiable errors to inspire confidence among scientists regarding the reduced data [23].

A compression pipeline was presented in our previous work [5]. It consists of three stages:

1) In the first stage, the data is decomposed using domain decomposition into small subdomains, and each subdomain is assigned to a different node. Multiple processors on each node process data in a subdomain independently, resulting in a high degree of parallelism.
2) In the second stage, data compression is performed using MGARD [2], [3], [4] to ensure error bounds on PD. MGARD achieves these guarantees through truncated basis function representations. The input domain is recursively divided into subdomains, with each subdomain represented by a fixed basis function to encode the data. If the fixed basis cannot meet the desired error requirements, the subdomain is further divided until the error guarantees are met. This iterative process is applied to each image, ensuring that the error guarantees are maintained.
3) The third stage uses a nonlinear projection of the primary data into a subspace spanned by the $\mathcal{O}(1)$ QoI constraints expressed by Lagrange multipliers, λs. Instead of directly storing the QoIs, this nonlinear projection operation aligns the compressed primary data with the constraints. The number of λs is proportional to the number of QoIs and can be further truncated to increase the level of compression.

Using the above approach, this paper presents a workflow that seamlessly integrates the simulation of the XGC application with an online data compression pipeline. XGC is a large-scale nuclear fusion simulation code that produces hundreds of terabytes of data in a single day. The important QoIs for this application include moments of the particle histogram, with the number of QoIs being $\mathcal{O}(1)$ in the cardinality of the PD. Similar to our previous work in [5], our data compression pipeline guarantees error on the PD within user-defined bounds, and the QoI errors are generally less than floating-point error.

The current paper introduces three novel additions to this pipeline.:

1) Concurrent Execution of Compression and Simulation: In previous work [5], it was shown that compression can be achieved by a process reading the raw data stored on storage servers by the simulation. In this paper, we present a pipeline consisting of simulation nodes and compression nodes working concurrently. Compression for the previous time step, including writing the required compression data, is completed while the simulation proceeds with the current time step. We develop a software approach for concurrently executing compression and simulation on a disjoint set of nodes and for communication between them.
2) GPU-based Optimization of MGARD to Reduce Its Overhead by an Order of Magnitude: By building an end-to-end GPU compression pipeline with customized optimizations for XGC data, we have significantly improved the compression throughput, reducing the overall overhead of compression.
3) Novel Post-processing Algorithm: Unlike the iterative approach presented in [5], we have developed a direct

approach that reduces the overall time to compute the λ's by an order of magnitude. It utilizes a projection of the primary data into a subspace spanned by the $\mathcal{O}(1)$ QoI constraints expressed by Lagrange multipliers, λs.

Overall, the data compression pipeline is highly parallel, scalable, and practical for applying on-the-fly compression while guaranteeing errors on QoIs that are critical to scientists. The software uses ADIOS-2 [38] to achieve high-performance, self-describing parallel I/O. ADIOS can reach over 1 TB/s on the Summit supercomputer GPFS file system.

Our experimental results on Summit, a supercomputer at Oak Ridge National Laboratory [24], show that the proposed pipeline can compress the data by two orders of magnitude with guarantees on the primary data. Similar to the post-processing schemes in [5], the new post-processing optimization technique achieves high accuracy (near floating-point errors) on the QoIs. The number of compression nodes required is roughly half a percent of the nodes required for simulation, representing a factor of 6-8 improvement in terms of the compression overhead presented in [5].

The rest of the paper is organized as follows. Section II presents the background, including the XGC application. Section III describes our compression pipeline and novel aspects of GPU processing and post-processing algorithm. Section IV explains our concurrent simulation and compression workflow. Section V presents our experiments. Section VI presents related work, and finally, we conclude in Section VII."

II. Background

In this section, we briefly describe the XGC application and MGARD [2], [3], [4], a hierarchical decomposition algorithm that we use for guaranteeing errors on primary data.

A. XGC

The X-point Gyrokinetic Code (XGC) [9], [26] is a fusion simulation software designed to study the kinetic transport properties of tokamaks, which are fusion reactors. Tokamaks confine extremely hot plasma within a toroidal structure using magnetic fields and are considered the most promising path to achieving fusion power. However, managing the complex particle and energy interactions near the plasma edge presents challenges in plasma maintenance. To address this, XGC has been developed to simulate trillions of particles simultaneously, with a focus on the edge plasma, to gain a better understanding of the intricate plasma behavior within a tokamak. XGC simulations enable scientists to identify potential improvements in tokamak design and operation, ultimately enhancing the efficiency and reliability of fusion power generation.

To be more precise, XGC employs the Particle-In-Cell (PIC) method, which generates and advances a large number of computational particles in a time-stepping manner, based on the five-dimensional gyrokinetic equations of motion. These particles are subsequently interpolated or "gathered" onto distinct mesh points to solve the electric and magnetic field equations that influence particle motion in subsequent stages. The resulting particle distribution functions, denoted as F (Figure 1), can be effectively represented as two-dimensional tensors within a three-dimensional grid, representing the particle count in both physical and velocity spaces. The data size of F can range from several gigabytes to tens of terabytes per iteration, depending on the mesh resolution.

The significant size of the particle distribution functions generated by XGC presents challenges related to I/O and storage [11]. To tackle these I/O bottlenecks and storage issues, various data compression techniques have been proposed. One active research area focuses on applying lossy data compression to F, which involves encoding and subsequently decoding the data as needed. However, this approach inevitably leads to information loss. Therefore, it is crucial to develop techniques that can compress and decompress F data quickly and efficiently while maintaining acceptable error limits. This will enable scientists to extract valuable insights from these simulations without compromising essential physics information.

The XGC code offers high flexibility and has been utilized to model various tokamak experiments and devices. In this paper, we utilize the ITER simulation as a demonstration of our pipeline. The scientists are particularly interested in four quantities of interest (QoIs): density (n), parallel flow velocity ($u_\parallel$), perpendicular temperature ($T_\perp$), and parallel temperature ($T_\parallel$). Additionally, two additional QoIs can be derived by manipulating these four quantities.

To account for the discrete nature of the XGC data available on a mesh, the QoI constraints need to be discretized. At a specific time step and mesh location (x, t), the linearized equations for the four QoIs of the 2D histogram $\boldsymbol{f}$ can be expressed as follows:

$$\sum_{i,j} f_{ij}\text{vol}_{ij} = n^c, \tag{1}$$

$$\sum_{i,j} f_{ij}\text{vol}_{ij} v_j^\parallel = n^c u_\parallel^c, \tag{2}$$

$$\frac{1}{2}m\sum_{i,j} f_{ij}\text{vol}_{ij}\left(v_i^\perp\right)^2 = n^c T_\perp^c, \tag{3}$$

$$\frac{1}{2}m\sum_{i,j} f_{ij}\text{vol}_{ij}\left(v_j^\parallel\right)^2 = n^c\left(T_\parallel^c + \frac{1}{2}m\left(u_\parallel^c\right)^2\right), \tag{4}$$

where m, vol_{ij}, $v_j^\parallel$, and $v_j^\perp$ represent particle mass, mesh node volume, parallel velocity, and perpendicular velocity, respectively, and they can be obtained from the XGC metadata. The four QoI constraints can be expressed in the form $A\boldsymbol{f} = \boldsymbol{b}$, where $A = \left[A^1, A^2, A^3, A^4\right]^T$ with $A_{ij}^1 = \text{vol}_{ij}$, $A_{ij}^2 = \text{vol}_{ij}v_j^\parallel$, $A_{ij}^3 = \frac{1}{2}m\text{vol}_{ij}\left(v_i^\perp\right)^2$, $A_{ij}^4 = \frac{1}{2}m\text{vol}_{ij}\left(v_j^\parallel\right)^2$, and $\boldsymbol{b} = \left[n^c, n^c u_\parallel^c, n^c T_\perp^c, n^c\left(T_\parallel^c + \frac{1}{2}m\left(u_\parallel^c\right)^2\right)\right]^T$. The constraint matrix A is specific to each instance because it varies over the mesh points. However, it can be constructed solely from the metadata and remains constant across all simulation time steps. Therefore, we do not consider the construction

of the A matrix as an additional cost in the compression framework.

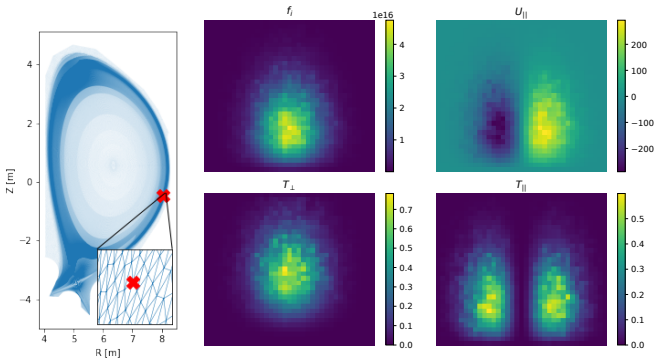


Fig. 1: A cross-sectional structure of tokamak [19] that XGC simulates with an example of XGC ITER mesh and the F data and their derived quantities (bottom). The marker shows the location of F data, each point corresponds to a 2D tensor.

B. MGARD-based Compression

There are a variety of data compressors, such as MGARD [2], [3], [4], SZ [39], [42], and ZFP [30]. We chose MGARD because prior work has shown that MGARD offers tighter error bounds on L^2 norm of errors and can be parameterized to better preserve either low frequency or high frequency components during data compression [22], [27], [29].

MGARD operates in two modes: compression and decompression. In this discussion, we will focus on the compression algorithm, noting that the decompression algorithm is essentially its inverse. The compression algorithm comprises three stages: multilevel decomposition, quantization, and lossless encoding. The input to the compression algorithm is an array $\mathtt{u}$, potentially in d-dimensions, of floating-point numbers. MGARD interprets the numbers as the values taken by a continuous function u on a grid $\mathcal{N}_L$ having the same grid structure as $\mathtt{u}$. The decomposition stage aims to transform the data $\mathtt{u}$ into a representation that is more suitable for compression. This is achieved by decomposing the data into a set of multilevel coefficients, denoted $\mathtt{u\_mc}$, residing on a hierarchy of grids, denoted by $\mathcal{N}_L, \ldots, \mathcal{N}_0$, which decrease in resolution and eventually reach the coarsest subgrid $\mathcal{N}_0$. Following the decomposition, the quantization stage introduces error according to user-defined error bounds, enabling compression in the final lossless encoding stage. MGARD employs linear-scaling quantization. The result of quantization is a set of quantized multilevel coefficients $\tilde{\mathtt{u}}\mathtt{\_mc}$ which encode an approximation $\tilde{u}$ of u. In the last stage, compression is achieved by losslessly encoding the quantized coefficients $\tilde{\mathtt{u}}\mathtt{\_mc}$. MGARD employs Huffman encoding [35] and GZIP/ZSTD [16], [13] lossless compression on CPU or nvCOMP [36] lossless compression on GPU to convert the quantized $\tilde{\mathtt{u}}\mathtt{\_mc}$ into a reduced representation. Similar to other state-of-the-art lossy compressors, MGARD supports L^2 and L^∞ error control. MGARD also supports H^s error control, where H^s is the Sobolev space of index s.

Due to space limitations, we refer the readers to [3], [4] for further details on the MGARD's L^2, L^∞, and H^s error control.

III. Scalable Compression Pipeline

XGC's tokamak simulation generates an enormous volume of data. A tokamak has a toroidal shape, and the simulations encompass multiple cross-sections of the torus, which are referred to as planes (see Figure 1). Each plane is discretized using a triangular mesh overlay. The particles gathered at each mesh node are histogrammed based on their velocity, with discretized velocity components along the directions parallel and perpendicular to the magnetic field. The input data for our compression pipeline consists of the particle histograms in the two-dimensional velocity space at the mesh points on each plane. Consequently, the data size varies depending on the resolution of the underlying mesh

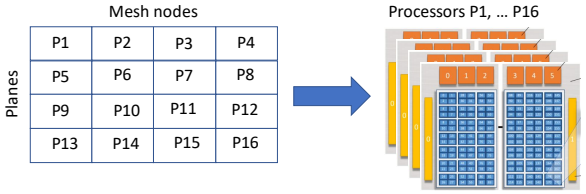


Fig. 2: Data decomposition scheme for XGC dataset. The mesh points on each of the four planes are assigned to a set of processors. Each mesh node consists of 2D particle velocity histograms.

The compression pipeline consists of three main stages. The first stage involves decomposing the data using domain decomposition into small subdomains and assigning them to processors. In the second stage, MGARD is utilized to ensure error bounds on the primary data. The third stage employs a post-processing optimization technique, which entails a non-linear projection of the primary data into a subspace spanned by the $\mathcal{O}(1)$ QoI constraints expressed by the Lagrange multipliers, λs. These three stages will now be described in detail.

A. Stage 1: Domain Decomposition

Domain decomposition is a technique used to divide the input tensor into smaller chunks and distribute them to parallel processors for individual compression. In this stage, domain decomposition is employed to divide the tensor into smaller chunks and distribute them to parallel processors for separate compression. The data is efficiently read and decomposed using the ADIOS framework. Figure 2 illustrates the data decomposition scheme. Each XGC plane's data is processed by a subset of processors, with the mesh points on each plane divided roughly equally among the processors assigned to process that plane (or subdomain). Stages 2 and 3 (described

below) are independently applied to each subdomain, ensuring high scalability of the overall approach.

B. Stage 2: Guaranteeing PD Error Bounds

As described in Section II-B, MGARD achieves these guarantees through the use of truncated basis function representations. The input domain is recursively divided into subdomains, with each subdomain represented by a fixed basis function. If the fixed basis function fails to meet the required error tolerance, it is discarded, and the subdomain is further subdivided. This iterative process ensures that the error guarantees are met, and a subdivision procedure is executed for each image. The overall computation is memory-bound, meaning that achieving high performance on GPUs requires more than just high parallelism; it also requires the following properties for the parallel compression algorithm: (1) fine-grain parallelism with minimal execution divergence between threads; (2) highly efficient memory access patterns to fully utilize the memory bandwidth; and (3) effective strategies for latency hiding to handle various types of latency in the deep memory hierarchy of GPUs.

The existing MGARD compression pipeline contains three major parts: multilevel decomposition, linear quantization, and entropy encoding. Previously, only multilevel decomposition has been optimized for GPUs [10]. Although multilevel decomposition appears to be highly parallelizable, the interlocked grid data structure makes memory access not GPU friendly. So, an optimized data layout was designed to split and pack coarse grid with coefficients of each level to greatly improve performance. Also, to further improve performance, three customized GPU kernels were designed to use shared memory to build a micropipeline to effectively hide the latency of accessing global memory.

In this work, we extend the work of [10] to build an end-to-end compression pipeline on GPUs by designing quantization and an entropy-encoding kernel on GPUs. Linear quantization works by assigning different quantization bin sizes to different coefficients of different levels. Specifically, for multilevel coefficient $mc[i]$ at level l, the quantization is done by calculating $quantized\_data[i] = copy\_sign\left(0.5 + \left|\frac{mc[i]}{quantizer[l]}\right|, mc[i]\right)$, where $quantizer[i]$ is the quantizer at level l. To optimize the linear quantization process on GPU, we pre-compute the value of the quantizer of each level on the CPU and cache them into shared memory on the GPU before quantization. This eliminates redundant quantizer calculation processes because each level shares the same quantizer value.

For entropy encoding, because quantized values are normally distributed integers, we choose to use Huffman variable-length encoding to achieve compression. Inspired by [41], we use the two-phase Huffman code generation algorithm [1] to achieve fine-grain parallelism on GPUs. To achieve high encoding performance and compression ratio, the key optimization is handling the Huffman codebook for variable-length encoding. We build a cache-memory adaptive strategy to improve the performance of accessing the codebook. Specifically, when the codebook is small enough (smaller dictionary size), we cache the codebook in GPU shared memory. On the other hand, for data that are hard to compress, we need to use a larger dictionary, which may lead to a larger codebook. In this case, we adaptively save the codebook in GPU global memory. So by combining the work of [10] and our GPU quantization and Huffman encoding kernels, we build an end-to-end MGARD compression on GPUs.

To improve the performance for compressing XGC data with MGARD compression and post-processing pipeline, we use two optimizations described as follows.

1) A limitation of existing GPU-accelerated multilevel decomposition [10] is that the maximum achievable parallelism is bounded by the smallest dimension of the input data. This is because the tridiagonal linear solver used [10] is only parallelized by the number of linear systems. Each system is solved by the sequential Thomas algorithm. It can provide decent performance when the input data have a regular shape. However, for tall-and-skinny shaped data it may cause performance degradation. For example, after domain decomposition, the XGC data can be tall-and-skinny. Specifically, the shape is $V_x \times nnodes \times V_y$, where V_x and V_y are the two dimensions of the particle histogram, which are usually around 30-40. $nnodes$ is the number of nodes assigned to the current GPU, which can be on the order of hundred thousands to millions. In this case, the parallelism of decomposition is bounded by V_x or V_y. To optimize, we proposed a dimension decomposition technique to decompose the data along the tall dimension so that the decomposed datasets have similar dimensions. For example, for XGC data, our algorithm will first identify that $nnodes$ is the largest dimension, then it does a domain decomposition along that dimension so that the decomposed dimension size is close to V_x and V_y. Because each decomposed dataset can be processed in parallel, we can effectively improve decomposition performance.

2) We used a new compression pipeline to facilitate post-processing where both compressed data and decompressed data are needed. Because the compression post-processing relies on the decompressed data to measure the error incurred by MGARD compression, the existing compression pipeline invokes MGARD decompression immediately after compression. This brings unnecessary entropy decoding overhead as we can directly compute the decompression data by inverting the compression pipeline before entropy encoding. So, we propose to build a new compression pipeline in MGARD such that decompressed data are produced immediately after the quantization operation in the compression pipeline.

The above optimization results in a factor of 4.6 improvement in total execution time (cf. Section V). The error control capability and compression ratios remain unchanged since the new results are based on the same compression algorithm. While the optimization of the decomposition does introduce a marginal decline in compression ratio, this can be readily offset by employing additional encoders during the entropy encoding stage.

C. Guaranteeing QoI Error Bounds

Convex optimization with linear constraints is a suitable framework for QoI preservation as shown in (5): where the objective function $d : \mathbb{R}^N \to \mathbb{R}$ representing the PD error is minimized subject to equality constraints $h : \mathbb{R}^N \to \mathbb{R}^M$, representing the M different QoI. The loss function $d\left(\boldsymbol{f}, \tilde{\boldsymbol{f}}\right)$ is a measure of distance between $\boldsymbol{f}$ and $\tilde{\boldsymbol{f}}$, where $\boldsymbol{f}$ is the corrected PD obtained from $\tilde{\boldsymbol{f}}$ after QoI post-processing, and $\tilde{\boldsymbol{f}}$ is the PD obtained after decompressing previously compressed data.

$$\begin{aligned} &\text{minimize} \quad d\left(\boldsymbol{f}, \tilde{\boldsymbol{f}}\right) \\ &\text{subject to} \quad h\left(\boldsymbol{f}\right) = 0. \end{aligned} \tag{5}$$

The classical method to solve the constrained optimization problem of (5) uses the Lagrange multiplier theorem [6], which proves that the stationary points (maxima, minima, or saddle points) of a real function d in an N-dimensional domain satisfying M constraints $h_k(f_1, f_2, \ldots, f_N)$ for $k \in \{1, \ldots, M\}$ are such that at these stationary points, the gradient of d is a linear combination of the gradients of the constraint functions. The Lagrange multipliers $\{\lambda_k\}$ are the coefficients of this linear combination:

$$\nabla d(f_1, f_2, \ldots, f_N) = \sum_{k=1}^{M} \lambda_k \nabla h_k(f_1, f_2, \ldots, f_N) \tag{6}$$

where the gradients ∇d and ∇h_k are N-dimensional vectors. We introduce an auxiliary equation—the Lagrangian—to incorporate the condition in (6):

$$\mathcal{L}\left(\boldsymbol{f}, \boldsymbol{\lambda}\right) = d\left(\boldsymbol{f}, \tilde{\boldsymbol{f}}\right) + \boldsymbol{\lambda}^T h\left(\boldsymbol{f}\right), \tag{7}$$

where $\boldsymbol{\lambda} = [\lambda_1, \ldots, \lambda_M]$ is a Lagrange multiplier vector. A solution to (6) determines λ_k, $k \in \{1, \ldots, M\}$, which leads us to a corrected output $\boldsymbol{f}$ that preserves the QoIs while simultaneously minimizing the distance between $\boldsymbol{f}$ and $\tilde{\boldsymbol{f}}$ to ensure the PD errors are as small as possible while satisfying the QoI.

Tha authors in [5] chose the *Bregman divergence* for the $x \log x$ entropy function [7], [8] as the objective in (7). The objective function is convex, but the direct solution for $\boldsymbol{\lambda}$ does not exist. Thus, with the duality theorem, they used Newton's method [15] to find the optimal value of the dual via a standard iterative procedure.

In this work, we enhance the computational efficiency of constraint satisfaction by devising with a direct solution for the variable $\boldsymbol{\lambda}$ as opposed to employing the iterative procedure described in [5]. Furthermore, no tradeoffs are involved. We consider the ℓ_2 distance as the distance measure in (7). Let the original histogram be $\boldsymbol{f}_o$. We first obtain the true QoI by $\boldsymbol{b} = A\boldsymbol{f}_o$. With the set of QoI constraints $A\boldsymbol{f} = \boldsymbol{b}$ in hand, the constraint satisfaction Lagrangian is

$$\mathcal{L}\left(\boldsymbol{f}, \boldsymbol{\lambda}\right) = \frac{1}{2}\left(\boldsymbol{f} - \tilde{\boldsymbol{f}}\right)^T \left(\boldsymbol{f} - \tilde{\boldsymbol{f}}\right) + \boldsymbol{\lambda}^T \left(A\boldsymbol{f} - \boldsymbol{b}\right). \tag{8}$$

The first order Karush–Kuhn–Tucker (KKT) conditions are

$$\boldsymbol{f} - \tilde{\boldsymbol{f}} + A^T\boldsymbol{\lambda} = \boldsymbol{O}, \tag{9}$$

$$A\boldsymbol{f} = \boldsymbol{b}. \tag{10}$$

Because we seek $A\boldsymbol{f} = \boldsymbol{b}$, we multiply by A to get

$$\boldsymbol{b} - A\tilde{\boldsymbol{f}} + AA^T\boldsymbol{\lambda} = \boldsymbol{O} \to \boldsymbol{\lambda} = \left(AA^T\right)^{-1} A\left(\tilde{\boldsymbol{f}} - \boldsymbol{f}_o\right). \tag{11}$$

Substituting $\boldsymbol{\lambda}$ in (9), we get

$$\boldsymbol{f} = \tilde{\boldsymbol{f}} - A^T\boldsymbol{\lambda} = \tilde{\boldsymbol{f}} - P\left(\tilde{\boldsymbol{f}} - \boldsymbol{f}_o\right), \tag{12}$$

where $P = A^T\left(AA^T\right)^{-1} A$. The output $\boldsymbol{f}$ is essentially a linear projection of $\tilde{\boldsymbol{f}}$ onto the subspace spanned by constraints. While this approach does not guarantee pointwise non-negativity of the estimate, the improvement in computational complexity makes it a viable candidate for QoI preservation.

We further improve the computation of the projection in (12). Each node in the ITER-scale XGC dataset represents a 33×37 2D histogram. For every node, we need to compute the projection matrix P that includes an inverse operation and has (1221×1221) dimensionality. Instead of performing the full matrix multiplication and inversion, we simplify the computation using the Singular Value Decomposition (SVD). The SVD of A is a factorization of the form $A = U\Sigma V^T$, where the columns of U are the left singular vectors, Σ is a diagonal matrix containing the singular values, and the rows of V^T are the right singular vectors. The computation of P can be simplified as

$$\begin{aligned} P &= A^T\left(AA^T\right)^{-1} A = V\Sigma U^T\left(U\Sigma V^T V\Sigma U^T\right)^{-1} U\Sigma V^T \\ &= V\Sigma\left(\Sigma^T\Sigma\right)^{-1}\Sigma V^T = VV^T, \end{aligned} \tag{13}$$

and thus the corrected output $\boldsymbol{f}$ is

$$\boldsymbol{f} = \tilde{\boldsymbol{f}} - P\left(\tilde{\boldsymbol{f}} - \boldsymbol{f}_o\right) = \tilde{\boldsymbol{f}} - V\left(V^T\left(\tilde{\boldsymbol{f}} - \boldsymbol{f}_o\right)\right), \tag{14}$$

where the parentheses indicate the order of matrix computation—one that drastically reduces the computational complexity. Because the V depend solely on the metadata, common across all the simulation time steps, we precompute it for every node and add it to the metadata. Whenever we perform post-processing, we load V and compute the corrected output $\boldsymbol{f}$. The size of V is more than 300 times smaller than the size of P. Furthermore, the matrix multiplication complexity of $P\left(\tilde{\boldsymbol{f}} - \boldsymbol{f}_o\right)$ is $\mathcal{O}\left(n^3\right)$ whereas the complexity of $V\left(V^T\left(\tilde{\boldsymbol{f}} - \boldsymbol{f}_o\right)\right)$ is $\mathcal{O}\left(n^2\right)$. This improves the overall computational time because the number of nodes in the ITER-scale XGC dataset is very large, with more than 8 million per time step.

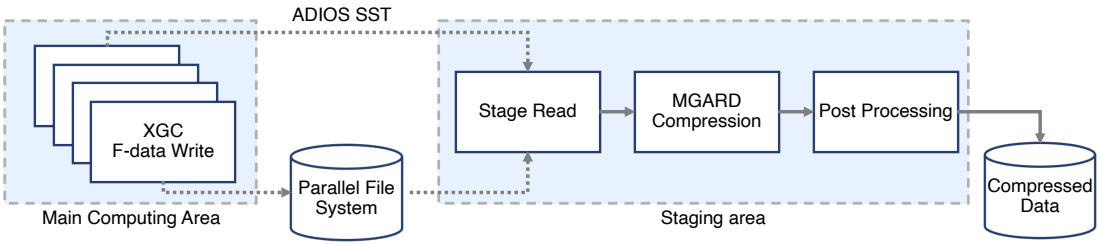


Fig. 3: Online data compression workflow concurrently with the simulation. The simulation is conducted in the main computing area. The values generated from the simulation are sent to a staging area consisting of compression nodes using either the Parallel File System or ADIOS SST Software. The staging area conducts the compression of data from the previous time step while the simulation is proceeding with generation of the data for the current time step. For this approach to work the total time for compression per step should be less than the time required for conducting the simulation per time step. Also, it is desirable that the number of compression nodes should be significantly smaller than the simulation nodes as the former represents the overhead for compression.

IV. Concurrent Simulation and Compression Workflow

Code coupling is a frequently employed approach in scientific research for exploring multiscale and multiphysics simulations, often called coupled simulation or co-simulation [12]. This method entails running multiple applications concurrently in either a loosely coupled or tightly synchronized manner to produce a cohesive outcome. Managing data communication is a crucial aspect of the process, as coupled codes depend on sharing vital information with each other. Coordinating these data exchanges poses a considerable challenge, especially in large-scale, high-performance computing environments.

Integrating online data compression within coupled workflows allows scientific applications to focus on their primary goals without incurring delays due to I/O-intensive operations on supercomputers. To achieve this, we leverage ADIOS [21] as a data management and workflow management tool, which has demonstrated effectiveness in high-performance computing within modern HPC environments. Our specific goal is to perform online data compression in a staging area, utilizing separate extra nodes, while XGC is simultaneously running. This allows us to offload data compression. Two critical performance factors for this integrated workflow are the time required for data movement and the compression time in the staging area. ADIOS is used to manage the data transfer from XGC to the staging area, and the compression time is assumed to be smaller than the time per step of XGC. The overall workflow is shown in Figure 3. We have implemented two ways of performing the data transfer: one through the file system and another one using remote copy based on SST. Additional details are provided in the experimental results section.

To minimize the number of staging nodes needed, it is crucial to optimize both memory usage and performance of the compression process running in the staging area to complete within each XGC iteration. As discussed earlier, we achieve this by utilizing an optimized version of MGARD and making algorithmic improvements to the Quality of Interest (QoI) calculation, which requires less memory as well as less computational overhead.

V. Experiments

We applied our data compression pipeline end-to-end on large-scale XGC data from the ITER simulation. We present the experimental setup in Section V-A, the accuracy results are presented in Section V-B, and performance results in Section V-C.

A. Setup

1) Platform: We perform experiments on Summit, a supercomputer at Oak Ridge National Laboratory [24]. It consists of 4608 IBM Power9 compute nodes, each with 42 cores and 6 Nvidia Volta GPUs. Each node has 1.6 TB of node-local NVME storage. The IBM Spectrum Scale GPFS parallel file system is used for large-scale I/O.

2) ITER dataset: We collect the outputs from an XGC simulation for the ITER geometry using 512 Summit nodes (3072 GPUs) and focus on compressing F data (the data for the particle distribution function) over multiple steps. The XGC ITER simulation employs 87 billion virtual electrons and ions. It creates 78.7 GB of F data for both ions and electrons per step, representing the kinetics of particle motions in five-dimensional physics (three dimensions in real space and two dimensions in velocity space). XGC saves F data in the file system by forming a multidimensional array of shape $(4, 33, 1.1\text{x}10^6, 37)$, which represents the data for 4 poloidal planes, 1.1 million mesh points, and 33-by-37 resolution velocity space.

3) ADIOS: ADIOS-2 [21], the Adaptable Input Output System, is an I/O framework designed to manage scientific data at scale, focusing on providing scalable parallel I/O performance on HPC systems. ADIOS provides a unified application programming interface (API) with a level of abstraction focusing on how data are produced and consumed in scientific applications. In addition, ADIOS provides various data services to reduce the cost of integrating different data-related operations, such as metadata collection, transportation, and compression [21], [34].

ADIOS supports a high-level operator mechanism, known as a callback, to allow a set of operations to perform on the data in memory before it is written or saved in a specific format.

With this operator mechanism, ADIOS can easily support several different lossless and lossy compression methods. In addition, ADIOS provides a systematic way to extend the capability with a user-defined compression operator where users can define their data operations and compression. We focus on using the ADIOS callback mechanism to develop a compression method with post-processing for XGC data.

ADIOS2 comes equipped with the Sustainable Staging Transport (SST) engine, which facilitates direct connection between data producers and consumers via ADIOS2's write/read APIs. The system operates on a streaming data architecture, where data written to ADIOS2 through functions such as Put() becomes instantly available to readers using Get() and similar functions. SST is optimized for using in high-performance computing (HPC) environments, utilizing RDMA network interconnects to expedite data transfer between HPC applications. SST supports full MxN data distribution, enabling the number of reader ranks to differ from that of writer ranks. The engine also allows multiple reader cohorts simultaneous access to a writer's data.

B. Accuracy

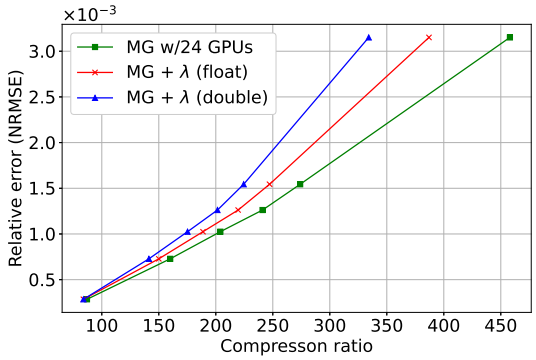
We use the criteria presented in [5] to measure, compare, and report the performance of the various compression schemes. For QoI and PD errors, we use the normalized root mean square error (NRMSE) as the error metric to evaluate the compression quality for both primary data and QoIs. NRMSE is a relative error defined as follows:

$$\text{NRMSE}(u, f) = \frac{\sqrt{\sum_{i=0}^{N}(u_i - f_i)^2/N}}{\max(u) - \min(u)}, \qquad (15)$$

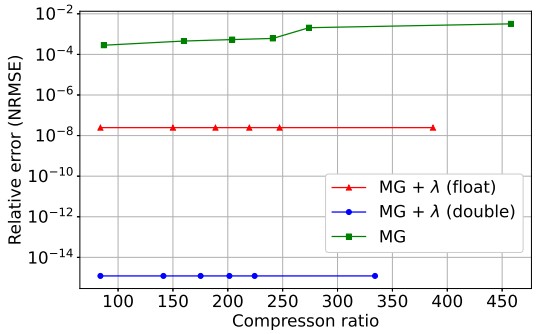
where u is the original data, f is the reconstructed data, and N is the number of degrees of freedom in u. If only a primary data compressor is applied to u, then, $f = \tilde{f}$. If a post-processing step is applied in addition to the primary data compressor, then, $f = \hat{f}$ in Eq. (15). The compression ratio is simply a ratio of the size of the original data to that of the compressed data.

We present accuracy results across multiple time steps. It is worth noting that the approach guarantees errors on QoI for each step separately. So, one would expect that the errors generated are relatively stable as the simulation proceeds.

We first show results for a single time step. Figure 4 shows the NRMSE of PD and the maximum NRMSE measured among four QoIs using the lossy compressed data produced by GPUized and optimized version of MGARD. For both cases, we show the errors when MGARD is run as a standalone application versus when a post-processing step is applied. Just like MGARD, the optimized version of MGARD as described in the earlier section ensures that the PD error is below a user-defined error bound. We also observe that for the same PD error bound, there is a difference of approximately 0.8% in PD error between MGARD and MGARD with post-processing, with the PD error slightly lower when post-processing is applied. On average, using both double and single-precision Lagrange parameters ($\boldsymbol{\lambda}$), the PD error is reduced by 0.8%.



(a) Errors in primary data (PD)



(b) Errors in Quantities of Interest (QoI)

Fig. 4: (a) The primary data are lossy compressed using MGARD then corrected using post-processing. The Lagrange parameters (i.e., $\boldsymbol{\lambda}$) are stored in their original (i.e., double) and a quantized precision (i.e., float). The errors of PD after post-processing are within a similar range as the ones before the post-processing. (b) We show the errors in QoIs computed from the lossy compressed data before and after post-processing. This is the average error measured among four QoIs. Overall, these results show that by applying post-processing, the errors of PD remain relatively constant, while the errors of QoIs are reduced by multiple orders of magnitude.

A careful investigation of Figure 4(a) shows that all the corresponding points for the three approaches a given level of PD error are nearly on the same horizontal line. The level of compression decreases because of need to store additional values for the $\boldsymbol{\lambda}$s.

Figure 4(b) shows that after using post-processing, the errors of QoIs drop to machine epsilon (as low as 10^{-14} when the Lagrange parameters are stored in double precision and as low as 10^{-8} when the Lagrange parameters are stored in single precision). The size of Lagrange parameters limits the maximum compression ratios that the pipeline may achieve (i.e., the curve of MG + λ (double) in Figure 4). By applying the quantization on Lagrange parameters, users can compress data more aggressively at the cost of a slight increase in PD errors. We found that converting the Lagrange parameters from double to single precision improves the compression ratio by an average of 20%.

The work in [5] compared NRMSE of PD and four QoIs using the lossy compressed data produced by MGARD and

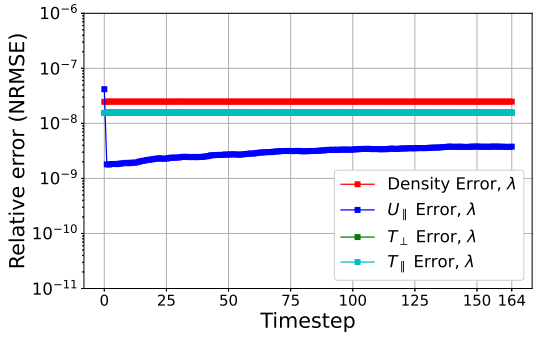


Fig. 5: The post-processing algorithm employed in the analysis of ITER data was successful in reducing errors in the QoIs across all available time steps. This suggests that the algorithm was able to consistently improve the accuracy and reliability of the data throughout the entire dataset. As the errors for $T_\parallel$ and $T_\perp$ are similar, they appear overlapped in this log scaled graph for NRMSE.

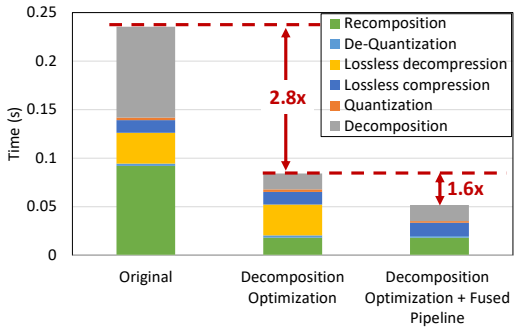


Fig. 6: Performance comparison of MGARD compression pipeline on one NVIDIA V100 GPU. Each stacked bar shows the time breakdown of the end-to-end pipeline of compressing 50,000 mesh points in the XGC data within one plane. The overall approach results in a reduction of total time by a factor of 4.6

compared to SZ [28] and ZFP [31], respectively. Among those target error ranges and by L2 error metric, MGARD delivers larger compression ratios than SZ and ZFP [22]. Therefore, in the rest of this section, we only show the results that pertain to MGARD. Although, our optimized post-processing approach is equally applicable to SZ and ZFP.

Figure 5 shows the NRMSE on ITER snapshots over 165 time steps using MGARD with and without post-processing. For post-processing, we only show results for single precision Lagrange parameters. The flat accuracy profile shows how the QoI preservation step successfully preserves all four QoIs over all ITER snapshots.

C. Performance

We experimented, compressing 50,000 mesh points on the first plane of the XGC simulation data of one of the time steps to compare the new MGARD implementation with the previous version. Figure 6 shows the performance comparison of these two MGARD compression pipelines. We can see that due to limited parallelism, the original decomposition and recomposition take the majority of the time in the pipeline. With our decomposition optimization, we effectively speed up the decomposition and recomposition by 5.7×, leading to total 2.8× speedup for the end-to-end compression pipeline. Introducing the post-processing step could lead to processor load imbalances. However, given that the post-processing task accounts for less than 30% of the overall time, equivalent to just a few seconds, the issue of load imbalance is of little significance. Furthermore, with our fused compression-decompression pipeline, we can eliminate the lossless decompression step (yellow bars) in the pipeline, which brings additional 1.6× speed-up to the end-to-end pipeline. The overall speed-up after using all optimizations is about 4.6×.

We also compared the post-processing algorithm described in Section III-C. We applied it to compressing all the mesh points assigned to a single node. We also implemented the iterative approach presented in [5] The results show that the new approach is about 20 times faster than the iterative approach presented in [5].

Using the optimized MGARD implementation and the new post-processing scheme, we performed a set of experiments where we ran online compression and post-processing in a separate staging area while XGC was running concurrently on 512 Summit nodes. We compared ADIOS's file-based coupling with SST, a network-based coupling method. We varied the number of staging nodes for online compression and post-processing from 6 to 2, which required an extra 1% of XGC node usage, as low as 0.4%. XGC took an average of 35 seconds per step, which serves as our upper limit for online compression. As we decreased the number of staging nodes, we observed an increase in data compression and post-processing time as expected. However, all our runs were completed within the average XGC execution time, demonstrating the feasibility and effectiveness of our approach.

Figure 7 shows the overall performance by varying the number of MPI processes used for compression and post-processing. MGARD compression is completed on multiple GPUs using MPI, whereas the post-processing is done on multiple CPUs using OpenMP and MPI to use all the cores on a Summit node.

These results show that the compression can be achieved by using only 2 nodes while the simulation requires 512 nodes. Further, the time for compression (including communication overhead) is less than the time required for a simulation step (roughly 35 seconds). Additionally, these results show that the SST approach is superior. Overall, including the compression costs, the overall time needed for data compression using an SST method is 7% to 31% more efficient compared to the use of the file-based approach.

VI. Related Work

The exponential growth of scientific data generated by modern high-performance computing (HPC) systems and high-resolution instruments has caused several issues in data storage, transmission, and analysis. While computing resources

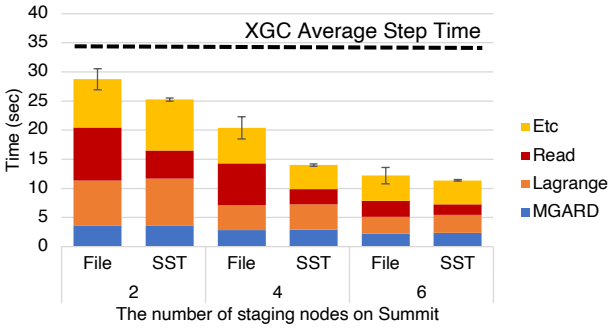


Fig. 7: Executing online compression and post-processing in a staging area with a varying number of staging nodes, ranging from 2 to 6, and comparing two staging methods: file-based and ADIOS SST. XGC runs 512 Summit nodes. Due to the MGARD optimization and memory optimization in the post-processing, we can finish the online compression using 2 nodes (0.4% of XGC simulation nodes) less than the average XGC step time (35 seconds).

have rapidly evolved, input/output (I/O) systems have not kept pace and remain a bottleneck in most scenarios. Data compression is considered a direct solution to address this issue, and various approaches have been presented in the literature. The two most common compression techniques used in the field are lossless and lossy.

Lossless compressors [17], [40], [33], [14], [32] ensure that no information is lost during the compression process. These techniques are preferred when preserving the original data is critical, but they tend to achieve lower compression ratios than lossy techniques. Error-bounded lossy compressors can reduce data volume significantly compared to lossless compression techniques while maintaining high fidelity of the reconstructed data within user-defined error tolerance.

In the literature, the most commonly used error-bounded lossy compression algorithms are based on prediction (such as SZ [18], [39], [28]), block transform (such as ZFP [30]), and multilevel decomposition (such as MGARD [2], [3], [4]). SZ follows a four-step pipeline to perform the compression, namely data prediction, quantization, Huffman encoding, and lossless compression. ZFP decorrelates the data using a near-orthogonal transform and encodes the transformed coefficients using embedded encoding. MGARD leverages wavelet theories and L2 projection for data decorrelation, followed by linear-scaling quantization, variable-length encoding, and lossless compression. While most error-bounded lossy compression algorithms aim to preserve pointwise values and offer guaranteed error control on primary data, they may not necessarily preserve derived quantities that are essential for accurate post-analysis. The work in [25] addressed guarantees on QoIs. However, it formulates the error preservation problem as an error bound derivation problem for each data point and has two constraints: 1) it requires pointwise error bound on PD and that the decompressed data is immediately accessible during compression. This constraint limits its application to lossy compressors that support pointwise error bound and instantaneous data reconstruction. Hence, their method only works with SZ. The technique proposed in this paper (as can the one presented in [5]) can work with any lossy compressor (MGARD, ZFP and SZ), 2) it can only preserve four types of QoI– polynomials, weighted sum, logarithmic mapping, and isoline/isosurface, as an analytic solution must be solved to map constraints of QoI. Several QoI for XGC do not fit this constraint. For example, temperature, which has a form of (g(x)/x) cannot be easily bounded using the techniques described.

In this research, we also significantly improved on our methods presented in [5] for preserving QoIs with various lossy compressors through a post-processing approach. In that study, an objective function was formulated as a Bregman divergence function. In this study, we use a Lagrangian optimization framework with an L2 projection. Using our approach, We eliminated the need for Newton's optimization to determine Lagrange parameters, which helps to reduce the overall time requirements. Moreover, several optimizations for implementing MGARD on GPUs are proposed in this study that lead to a significant improvement in the computational requirements.

VII. Conclusion

This paper presents an end-to-end algorithmic and software pipeline for scientific data compression that guarantees error on primary data while reducing the amount of errors on quantities of interest. We demonstrated our pipeline on an XGC-based simulation of ITER [37]. Our results show that by using the new MGARD GPU implementation and a novel post-processing algorithm, we can achieve a compression level of nearly 150 while ensuring that the error on the primary data is less than 10^{-3} and that of the QoIs is less than 10^{-8}. The size of the dataset was roughly 78.7 GBs for each step of simulation, with a step happening every 35 second. Our results demonstrated that these results hold for multiple time steps of the simulation (this is not surprising because the bounds are guaranteed separately for each time step).

This paper presents the following three novel ideas:

1) Novel approaches for GPU-based optimization of MGARD for XGC. These resulted in an overall improvement by a factor of 4.6 as compared to previous implementations.
2) Novel post-processing algorithm that uses a direct solver as compared to previous iterative approach [5]. This reduces the overall post-processing time by a factor of 20 or more.
3) We develop a software approach for concurrently executing the compression and simulation on a disjoint set of nodes and communication between them. Experimental results demonstrated that the total overhead required for a 512-node simulation is two nodes. This represents less than half a percent overhead in the additional computational resources for compression.

Overall, the data compression pipeline is highly parallel, scalable, and practical for applying on-the-fly compression

while guaranteeing errors on QoIs that are critical to scientists. Furthermore, the improvements described in this paper resulted in a reduction of the additional resources required to less than half a percent. In our previous work [5], 32 Summit nodes were required for compression to support a simulation executing on 1024 nodes. In this work, we were able to reduce the overhead to only 2 compression nodes for a simulation on 512 Summit nodes. This represents a factor of 6-8 lower overall compression overhead. Such a significant reduction in compression overhead makes the approach in this paper considerably more practical.

REFERENCES

[1] B. Accelerated, S. A. Ostadzadeh, Z. Zeinalpour-tabrizi, M. A. Moulavi, and K. Bertels. A two-phase practical parallel algorithm for construction of huffman codes. 2007.

[2] M. Ainsworth, O. Tugluk, B. Whitney, and S. Klasky. Multilevel techniques for compression and reduction of scientific data - the univariate case. *Computing and Visualization in Science*, 19(5-6):65–76, 2018.

[3] M. Ainsworth, O. Tugluk, B. Whitney, and S. Klasky. Multilevel techniques for compression and reduction of scientific data—the multivariate case. *SIAM Journal on Sci. Computing*, 41(2):A1278–A1303, 2019.

[4] M. Ainsworth, O. Tugluk, B. Whitney, and S. Klasky. Multilevel techniques for compression and reduction of scientific data-quantitative control of accuracy in derived quantities. *SIAM Journal on Scientific Computing*, 41(4):A2146–A2171, 2019.

[5] T. Banerjee, J. Choi, J. Lee, Q. Gong, R. Wang, S. Klasky, A. Rangarajan, and S. Ranka. An algorithmic and software pipeline for very large scale scientific data compression with error guarantees. In *HiPC*, 2022.

[6] D. P. Bertsekas. *Constrained Optimization and Lagrange Multiplier Methods*. Academic Press, 2014.

[7] L. M. Bregman. The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. *USSR Computational Mathematics and Mathematical Physics*, 7:200–217, 1967.

[8] Y. Censor and A. Lent. An iterative row-action method for interval convex programming. *Journal of Optimization Theory and Applications*, 34:321–353, 1981.

[9] C.-S. Chang and S.-H. Ku. Spontaneous rotation sources in a quiescent tokamak edge plasma. *Physics of Plasmas (1994-present)*, 15(6):062510, 2008.

[10] J. Chen, L. Wan, X. Liang, B. Whitney, Q. Liu, D. Pugmire, N. Thompson, J. Y. Choi, M. Wolf, T. Munson, et al. Accelerating multigrid-based hierarchical scientific data refactoring on gpus. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 859–868. IEEE, 2021.

[11] J. Choi, M. Churchill, Q. Gong, S.-H. Ku, J. Lee, A. Rangarajan, S. Ranka, D. Pugmire, C. Chang, and S. Klasky. Neural data compression for physics plasma simulation. In *Neural Compression: From Information Theory to Applications–Workshop@ ICLR 2021*, 2021.

[12] J. Y. Choi, C.-S. Chang, J. Dominski, S. Klasky, G. Merlo, E. Suchyta, M. Ainsworth, B. Allen, F. Cappello, M. Churchill, et al. Coupling exascale multiphysics applications: Methods and lessons learned. In *2018 IEEE 14th International Conference on e-Science (e-Science)*, pages 442–452. IEEE, 2018.

[13] Y. Collet and M. Kucherawy. Zstandard compression and the application/zstd media type. Technical report, 2018.

[14] Y. Collet and M. S. Kucherawy. Zstandard Compression and the 'application/zstd' Media Type. *RFC*, 8878:1–45, 2021.

[15] J. Dennis and J. Moré. Quasi-Newton Methods, Motivation and Theory. *SIAM Review*, 19(1):46–89, 1977.

[16] P. Deutsch. Gzip file format specification version 4.3. Technical report, 1996.

[17] P. Deutsch. Gzip file format specification version 4.3. 1952:1–12, 1996.

[18] S. Di and F. Cappello. Fast error-bounded lossy HPC data compression with SZ. In *IPDPS*, pages 730–739, 2016.

[19] J. Dominski, J. Cheng, G. Merlo, V. Carey, R. Hager, L. Ricketson, J. Choi, S. Ethier, K. Germaschewski, S.-H. Ku, et al. Spatial coupling of gyrokinetic simulations, a generalized scheme based on first-principles. *Physics of Plasmas*, 28(2):022301, 2021.

[20] I. Foster, M. Ainsworth, J. Bessac, F. Cappello, J. Choi, S. Di, Z. Di, A. M. Gok, H. Guo, K. A. Huck, et al. Online data analysis and reduction: An important co-design motif for extreme-scale computers. *The International Journal of High Performance Computing Applications*, 35(6):617–635, 2021.

[21] W. F. Godoy, N. Podhorszki, R. Wang, C. Atkins, G. Eisenhauer, J. Gu, P. Davis, J. Choi, K. Germaschewski, K. Huck, et al. ADIOS 2: The adaptable input output system. a framework for high-performance data management. *SoftwareX*, 12:100561, 2020.

[22] 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*, pages 22–39. Springer, 2021.

[23] D. Grois, D. Marpe, A. Mulayoff, B. Itzhaky, and O. Hadar. Performance comparison of H.265/MPEG-HEVC, VP9, and H.264/MPEG-AVC encoders. In *PCS*, pages 394–397. IEEE, 2013.

[24] J. Hines. Stepping up to Summit. *Computing in Science & Engineering*, 20:78–82, 03 2018.

[25] P. Jiao, S. Di, H. Guo, K. Zhao, J. Tian, D. Tao, X. Liang, and F. Cappello. Toward quantity-of-interest preserving lossy compression for scientific data. *Proc. VLDB Endow.*, 16(4):697–710, dec 2022.

[26] S.-H. Ku, C.-S. Chang, and P. H. Diamond. 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, 2009.

[27] J. Lee, Q. Gong, J. Y. Choi, T. Banerjee, S. Klasky, S. Ranka, and A. Rangarajan. Error-bounded learned scientific data compression with preservation of derived quantities. *Applied Sciences*, 12:6718, 07 2022.

[28] X. Liang, S. Di, D. Tao, S. Li, S. Li, H. Guo, Z. Chen, and F. Cappello. Error-controlled lossy compression optimized for high compression ratios of scientific datasets. In *Big Data*, pages 438–447, 2018.

[29] X. Liang, Q. Gong, J. Chen, B. Whitney, L. Wan, Q. Liu, D. Pugmire, R. Archibald, N. Podhorszki, and S. Klasky. Error-controlled, progressive, and adaptable retrieval of scientific data with multilevel decomposition. In *High Performance Computing, Networking, Storage and Analysis*, pages 1–13, 2021.

[30] P. Lindstrom. Fixed-rate compressed floating-point arrays. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2674–2683, 2014.

[31] P. Lindstrom. Fixed-rate compressed floating-point arrays. *IEEE TVCG*, 20, 08 2014.

[32] P. Lindstrom. Error distributions of lossy floating-point compressors. Technical report, Lawrence Livermore National Lab. (LLNL), Livermore, CA (United States), October 2017.

[33] P. Lindstrom and M. Isenburg. Fast and efficient compression of floating-point data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1245–1250, 2006.

[34] J. Logan, M. Ainsworth, C. Atkins, J. Chen, J. Y. Choi, J. Gu, J. M. Kress, G. Eisenhauer, B. Geveci, W. Godoy, et al. Extending the publish/subscribe abstraction for high-performance I/O and data management at extreme scale. *Bulletin of the IEEE Technical Committee on Data Engineering*, 43(1), 2020.

[35] A. Moffat. Huffman coding. *ACM CSUR*, 52(4):1–35, 2019.

[36] NVIDIA. Nvcomp: High speed data compression using nvidia gpus.

[37] P.-H. Rebut. ITER: the first experimental fusion reactor. *Fusion Engineering and Design*, 30(1):85–118, 1995.

[38] 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. *The International Journal of High Performance Computing Applications*, 36(1):106–128, 2022.

[39] D. Tao, S. Di, Z. Chen, and F. Cappello. Significantly improving lossy compression for scientific data sets based on multidimensional prediction and error-controlled quantization. In *IPDPS*, pages 1129–1139, 2017.

[40] D. S. Taubman, M. W. Marcellin, and M. Rabbani. Jpeg2000: Image compression fundamentals, standards and practice. *Journal of Electronic Imaging*, 11(2):286–287, 2002.

[41] J. Tian, C. Rivera, S. Di, J. Chen, X. Liang, D. Tao, and F. Cappello. Revisiting huffman coding: Toward extreme performance on modern gpu architectures. In *IPDPS*, pages 881–891. IEEE, 2021.

[42] K. Zhao, S. Di, M. Dmitriev, T.-L. D. Tonellot, Z. Chen, and F. Cappello. Optimizing error-bounded lossy compression for scientific data by dynamic spline interpolation. In *ICDE*, pages 1643–1654. IEEE, 2021.