AMRIC: A Novel In Situ Lossy Compression Framework for Efficient I/O in Adaptive Mesh Refinement Applications

Daoce Wang
Indiana University
Bloomington, IN, USA
daocwang@iu.edu

Jesus Pulido
Los Alamos National Lab
Los Alamos, NM, USA
pulido@lanl.gov

Pascal Grosset
Los Alamos National Lab
Los Alamos, NM, USA
pascalgrosset@lanl.gov

Jiannan Tian
Indiana University
Bloomington, IN, USA
jti1@iu.edu

Sian Jin
Indiana University
Bloomington, IN, USA
sianjin@iu.edu

Houjun Tang
Lawrence Berkeley National Lab
Berkeley, CA, USA
htang4@lbl.gov

Jean Sexton
Lawrence Berkeley National Lab
Berkeley, CA, USA
jmsexton@lbl.gov

Sheng Di
Argonne National Lab
Lemont, IL, USA
sdi1@anl.gov

Zarija Lukić
Lawrence Berkeley National Lab
Berkeley, CA, USA
zarija@lbl.gov

Kai Zhao
Florida State University
Tallahassee, FL, USA
kzhao@cs.fsu.edu

Bo Fang
Pacific Northwest National Lab
Richland, WA, USA
bo.fang@pnnl.gov

Franck Cappello
Argonne National Lab
Lemont, IL, USA
cappello@mcs.anl.gov

James Ahrens
Los Alamos National Lab
Los Alamos, NM, USA
ahrens@lanl.gov

Dingwen Tao*
Indiana University
Bloomington, IN, USA
ditao@iu.edu

ABSTRACT

As supercomputers advance towards exascale capabilities, computational intensity increases significantly, and the volume of data requiring storage and transmission experiences exponential growth. Adaptive Mesh Refinement (AMR) has emerged as an effective solution to address these two challenges. Concurrently, error-bounded lossy compression is recognized as one of the most efficient approaches to tackle the latter issue. Despite their respective advantages, few attempts have been made to investigate how AMR and error-bounded lossy compression can function together. To this end, this study presents a novel in-situ lossy compression framework that employs the HDF5 filter to improve both I/O costs and boost compression quality for AMR applications. We implement our solution into the AMReX framework and evaluate on two real-world AMR applications, Nyx and WarpX, on the Summit supercomputer. Experiments with 4096 CPU cores demonstrate that AMRIC improves the compression ratio by up to 81× and the I/O performance by up to 39× over AMReX's original compression solution.

CCS CONCEPTS

• **Theory of computation → Data compression**; • **Computing methodologies → Massively parallel and high-performance simulations**.

KEYWORDS

Lossy compression, AMR, I/O, performance.

*Corresponding author: Dingwen Tao, Department of Intelligent Systems Engineering, Luddy School of Computing, Informatics, and Engineering, Indiana University.

SC '23, November 12–17, 2023, Denver, CO, USA

ACM ISBN 979-8-4007-0109-2/23/11.
https://doi.org/10.1145/3581784.3613212

ACM Reference Format:
Daoce Wang, Jesus Pulido, Pascal Grosset, Jiannan Tian, Sian Jin, Houjun Tang, Jean Sexton, Sheng Di, Zarija Lukić, Kai Zhao, Bo Fang, Franck Cappello, James Ahrens, and Dingwen Tao. 2023. AMRIC: A Novel In Situ Lossy Compression Framework for Efficient I/O in Adaptive Mesh Refinement Applications. In *The International Conference for High Performance Computing, Networking, Storage and Analysis (SC '23), November 12–17, 2023, Denver, CO, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3581784.3613212

1 INTRODUCTION

In recent years, scientific simulations have experienced a dramatic increase in both scale and expense. To address this issue, many high-performance computing (HPC) simulation packages, like AMReX [42] and Athena++ [32], have employed Adaptive Mesh Refinement (AMR) as a technique to decrease computational costs while maintaining or even improving the accuracy of simulation results. Unlike traditional uniform mesh methods that apply consistent resolution throughout the entire simulation domain, AMR offers a more efficient approach by dynamically adjusting the resolution and focusing on higher resolution in crucial areas, thereby conserving computational resources and storage needs.

Although AMR data can reduce output data size, the reduction may not be substantial enough for scientific simulations resulting in high I/O and storage costs. For example, an AMR simulation with a resolution of 2048^3 (i.e., 0.5×1024^3 mesh points in the coarse level and 0.5×2048^3 in the fine level) can generate up to 1 TB of data for a single snapshot with all data fields dumped; a total of 1 PB of disk storage is needed, assuming the simulation is run in an ensemble of five times with 200 snapshots dumped per simulation.

To this end, data compression approaches could be utilized in conjunction with AMR techniques to further save I/O and storage costs. However, traditional lossless compression methods have limited effectiveness in reducing the massive amounts of data generated by scientific simulations, typically achieving only up to a compression ratio of 2×. As a solution, a new generation of error-bounded lossy compression techniques, such as SZ [10, 21, 33], ZFP [23], MGARD

[2] and their GPU versions [8, 37, 38], have been widely used in the scientific community [5, 7, 10, 15, 18, 20, 21, 23, 25, 26, 33, 35].

While lossy compression holds the potential to significantly reduce I/O and storage costs associated with AMR simulations, there has been limited research on using lossy compression in AMR simulations. Two recent studies have aimed to devise efficient lossy compression methods for AMR datasets. Luo et al. [27] proposed zMesh, which reorders AMR data across different refinement levels into a 1D array to leverage data redundancy. However, by compressing data in a 1D array, zMesh is unable to exploit higher-dimension compression, leading to a loss of topology information and data locality in higher-dimension data. In contrast, Wang et al. [40] developed TAC to enhance zMesh's compression quality through adaptive 3D compression. While zMesh and TAC offer offline compression solutions for AMR data, they are not suitable for in situ compression of AMR data. We will discuss these works and their limitations for in situ compression in detail in §5.

On the other hand, in situ compression of AMR data could enhance I/O efficiency by compressing data during the application's runtime, allowing for the direct writing of smaller, compressed data to storage systems. This approach would eliminate the need to transfer large amounts of original data between computing nodes and storage systems, further streamlining the process. AMReX currently supports in situ compression for AMR data [3]; however, the current implementation converts the high-dimensional data into a 1D array before compression, which limits the compression performance without the additional spatial information. Additionally, it utilizes a small HDF5 chunk size, leading to lower compression ratios and reduced I/O performance. These limitations will be discussed in more detail in Sections 2.1 and 3.3.

To address these issues, we propose an effective in situ lossy compression framework for AMR simulations, called AMRIC, that enhances I/O performance and compression quality. Different from AMReX's naïve in situ compression approach, with a customized pre-processing process, AMRIC can perform 3D compression and leverage its high compression ratio. Additionally, we incorporate the HDF5 compression filter to further improve I/O performance and usability. Our primary contributions are outlined below:

- We propose a first-of-its-kind 3D in situ AMR data compression framework through HDF5 (called AMRIC[1]).
- We design a compression-oriented pre-processing workflow for AMR data, which involves removing redundant data, uniformly truncating the remaining data into 3D blocks, and reorganizing the blocks based on different compressors.
- We employ the state-of-the-art lossy compressor SZ (with two different algorithms/variants) and further optimize it to improve the compression quality for AMR data. This involves utilizing Shared Lossless Encoding (SLE) and adaptive block size in the SZ compressor to enhance prediction quality and hence improve the compression quality.
- To efficiently utilize the HDF5 compression filter on AMR data, we modify the data layout and the original compression filter to adopt a larger HDF5 chunk size without introducing extra storage overhead. This enables higher compression throughput, improving both I/O and compression quality.
- We integrate AMRIC into the AMReX framework and evaluate it on two real-world AMReX applications, WarpX and Nyx, using the Summit supercomputer.
- Experimental results demonstrate that AMRIC can significantly outperform non-compression solution and AMReX's original compression solution in terms of I/O performance and compression quality.

The remainder of this paper is organized as follows. In §2, we provide an overview of error-bounded lossy compression for scientific data, the HDF5 file format, AMR approaches, and data structures, as well as a review of related work in the field of AMR data compression. In §3, we describe our proposed 3D in situ compression strategies in detail. In §4, we present the experimental results of AMRIC and comparisons with existing approaches. In §5, we discuss related work and their limitations. In §6, we conclude this work and outline potential future research.

2 BACKGROUND AND MOTIVATION

In this section, we introduce some background information on the HDF5 format and its filter mechanism, lossy compression for scientific data, and AMR methods and AMR data.

2.1 HDF5 Format and HDF5 Filter

An essential technique for minimizing I/O time associated with the vast data generated by large-scale HPC applications is parallel I/O. Numerous parallel I/O libraries exist, including HDF5 [14] and NetCDF [31]. In this study, we focus on HDF5, as it is widely embraced by the HPC community [12, 20] and hydrodynamic simulations such as Nyx [28], a simulation to model astrophysical reacting flows on HPC systems, and VPIC [6], a large-scale plasma physics simulation. Moreover, HDF5 natively supports data compression filters [1] such as H5Z-SZ [9] and H5Z-ZFP [24]. HDF5 allows chunked data to pass through user-defined compression filters on the way to or from the storage system [36]. This means that the data can be compressed/decompressed using a compression filter during the read/write operation.

Selecting the optimal chunk size when using compression filters in parallel scenarios is often challenging due to dataset chunking being necessary to enable the use of I/O filters. On the one hand, choosing a chunk size too small may lead to an excessive number of data blocks, resulting in a lower compression ratio caused by the reduction of encoding efficiency and data locality. Additionally, smaller chunks can hamper I/O performance as a consequence of the start-up costs associated with HDF5 and the compressor. On the other hand, larger chunks may enhance writing efficiency but can also create overhead if the chunk size exceeds the data size on certain processors. This occurs because the chunking size must remain consistent across the entire dataset for each process. Striking a balance between these two factors is essential for achieving optimal I/O performance and compression efficiency in parallel environments. Moreover, a larger chunking size can result in the compression of distinct fields of data (e.g., density and velocity) together. To tackle these challenges, we propose an optimized approach to modify the AMR data layout and adaptively determine the maximum chunk size without introducing size overhead issues. We will detail these methods in §3.3.

[1]The code is available at https://github.com/SC23-AMRIC/SC23-AMRIC.

2.2 Lossy Compression for Scientific Data

Lossy compression is a common data reduction method that can achieve high compression ratios by sacrificing some non-critical information in the reconstructed data. Compared to lossless compression, lossy compression often provides much higher compression ratios, especially for continuous floating-point data. The performance of lossy compression is typically measured by three key metrics: compression ratio, data distortion, and compression throughput. The compression ratio refers to the ratio between the original data size and the compressed data size. Data distortion measures the quality of the reconstructed data compared to the original data using metrics such as peak signal-to-noise ratio (PSNR). Compression throughput represents the size of data that the compressor can compress within a certain time.

In recent years, several high-accuracy lossy compressors for scientific floating-point data have been proposed and developed, such as SZ [10, 21, 33] and ZFP [23]. SZ is a prediction-based lossy compressor, whereas ZFP is a transform-based lossy compressor. Both SZ and ZFP are specifically designed to compress scientific floating-point data and provide a precise error-controlling scheme based on user requirements. For example, the error-bounded mode requires users to set a type of error-bound, such as absolute error bound, and a bound value. The compressor then ensures that the differences between the original and reconstructed data do not exceed the error bound.

In this work, we adopt the SZ lossy compressor in our framework due to its high compression ratio and its modular design that facilitates the integration with our framework, AMRIC. Additionally, the SZ framework includes various algorithms to satisfy different user needs. For example, SZ with Lorenzo predictor [34] provides high compression throughput, while SZ with spline interpolation [22] provides high compression ratio, particularly for large error bounds. Generally, there are three main steps in prediction-based lossy compression, such as SZ. The first step is to predict each data point's value based on its neighboring points using a best-fit prediction method. The second step is to quantize the difference between the real value and the predicted value based on the user-set error bound. Finally, customized Huffman coding and lossless compression are applied to achieve high compression ratios.

Several prior works have studied the impact of lossy compression on reconstructed data quality and post-hoc analysis, providing guidelines on how to set the compression configurations for certain applications [11, 18–22, 34]. For instance, a comprehensive framework was established to dynamically adjust the best-fit compression configuration for different data partitions of a simulation based on its data characteristics [19]. However, no prior study has established an in situ lossy method for AMR simulations by efficiently leveraging these existing lossy compressors. Therefore, this paper proposes an efficient in situ data compression framework for AMR simulations to effectively utilize the high compression ratio from lossy compression.

2.3 AMR Methods and AMR Data

AMR is a technique that tailors the accuracy of a solution by employing a non-uniform grid, which allows for computational and storage savings without compromising the desired accuracy. In

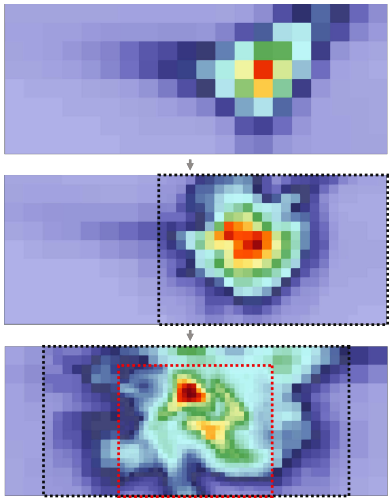


Figure 1: Visualization of an up-close 2D slice of three pivotal timesteps generated by an AMR-based cosmology simulation, Nyx. As the universe evolves, the grid structure adapts accordingly. The dashed black and red boxes highlight areas of finer and finest refinement, respectively.

AMR applications, the mesh or spatial resolution is adjusted based on the level of refinement required for the simulation. This involves using a finer mesh in regions of greater importance or interest and a coarser mesh in areas of lesser significance. Throughout an AMR simulation, meshes are refined according to specific refinement criteria, such as refining a mesh block when its maximum value surpasses a predetermined threshold (e.g., the average value of the entire field), as illustrated in Figure 1.

Figure 1 shows that during an AMR run, the mesh will be refined when the value meets the refinement criteria, e.g., refining a block when its norm of the gradients or maximum value is larger than a threshold. By dynamically adapting the mesh resolution in response to the simulation's requirements, AMR effectively balances computational efficiency and solution accuracy, making it a powerful approach for various scientific simulations.

Data generated by an AMR application is inherently hierarchical, with each AMR level featuring different resolutions. Typically, the data for each level is stored separately, such as in distinct HDF5 datasets(groups). For example, Figure 3 (left) presents a simple two-level patch-based AMR dataset in an HDF5 structure, where "0" indicates the coarse level (low resolution), and "1" signifies the fine level (high resolution). When users need the AMR data for post-analysis , they usually convert the data from different levels into a uniform resolution. In the given example, the data at the coarse level would be up-sampled and combined with the data at the fine level, excluding the redundant coarse data point "0D", as illustrated in Figure 3 (right). This method could also serve to visualize AMR data without the need for specific AMR visualization tool kits.

There are two primary techniques for representing AMR data: patch-based AMR and tree-based AMR [41]. The key distinction between these approaches lies in how they handle data redundancy across different levels of refinement. Patch-based AMR maintains redundant data in the coarse level, as it stores data blocks to be refined at the next level within the current level, simplifying the

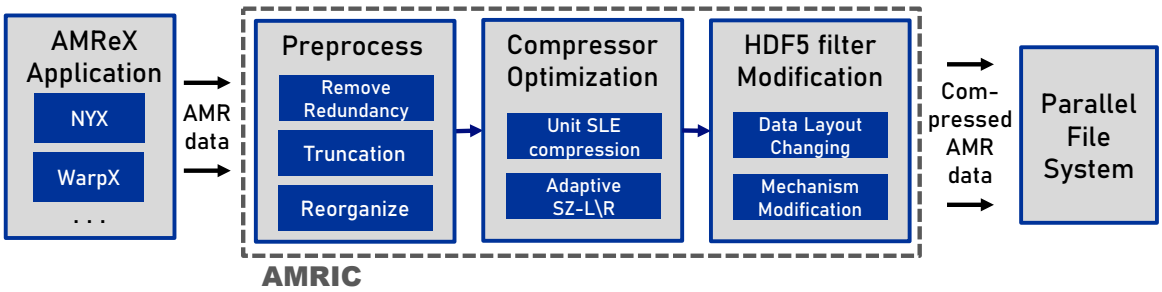


Figure 2: Overview of our proposed AMRIC.

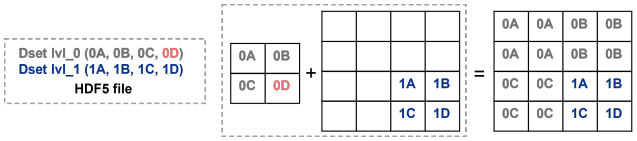


Figure 3: A typical example of AMR data storage and usage.

computation involved in the refinement process. Conversely, tree-based AMR organizes grids on tree leaves, eliminating redundant data across levels. However, tree-based AMR data can be more complex for post-analysis and visualization when compared to patch-based AMR data [16].

In this work, we focus on the state-of-the-art patch-based AMR framework, AMReX, which supports the HDF5 format and compression filter [3]. However, AMReX currently only supports 1D compression, which restricts its ability to leverage higher-dimension compression. Furthermore, the original compression of AMReX cannot effectively utilize the HDF5 filter, resulting in low compression quality and I/O performance (will be described in detail in §3.3 and Section 5). This limitation serves as motivation for our proposal of a 3D in situ compression method for AMReX, aimed at enhancing compression quality and I/O performance.

It is worth noting that the redundant coarser-level data in AMReX (patch-based AMR) is often not utilized during post-analysis and visualization, as demonstrated in Figure 3 (the coarse point "0D" will not be used). Therefore, we discard the redundant data during compression to improve the compression ratio.

3 DESIGN METHODOLOGY

In this section, we introduce our proposed in situ 3D AMR compression framework, AMRIC, using the HDF5 filter, as shown in Figure 2, with an outline detailed below.

In §3.1, we first propose a pre-processing approach for AMR data, which includes the elimination of data redundancy, uniform truncation of data, and reorganization of truncated data blocks tailored to the requirements of different compressors including different SZ compression algorithms. In §3.2, we further optimize the SZ compressor's efficiency for compressing AMR data by employing Shared Lossless Encoding (SLE) and dynamically determining the ideal block sizes for the SZ compressor, taking into account the specific characteristics of AMR data. In §3.3, we present strategies to overcome the obstacles between the HDF5 and AMR applications by modifying the AMR data layout as well as the HDF5 compression filter mechanism, which subsequently results in a significant improvement in both compression ratio and I/O performance, as discussed in §2.1.

3.1 Pre-processing of AMR Data

As mentioned in §2.3, in the patch-based AMR dataset generated by AMReX, the data in the coarse level could be removed to easily improve the compression ratio and I/O performance because there would be fewer data to be processed. Patch-based AMR divides each AMR level's domain into a set of rectangular boxes. Figure 4 (right) illustrates an example of an AMR dataset with three total levels. In the AMReX numbering convention, the coarsest level is designated as level 0. There are 4, 3, and 3 boxes on levels 0, 1, and 2, respectively. Bold lines signify box boundaries. The four coarsest boxes (black) cover the whole domain. There are three intermediate-resolution boxes (blue) at level 1 with cells that are two times finer than those at level 0. The three finest grids (red) at level 2 have cells that are twice as fine as those in level 1.

Clearly, there are overlapping areas between the different AMR levels: the coarsest level 0 overlaps with the finer level 1, and level 1 also has overlapping regions with the finest level 2. Taking level 1 as an example, we can eliminate the redundant coarse regions that overlap with level 2. It is worth noting that AMReX offers efficient functions for box intersection, which can be employed to identify these overlapping areas. These functions are significantly faster than a naive implementation, resulting in reduced time costs [4]. Furthermore, there is no need to record the position of the empty regions in the compressed data, as the position of the empty regions in level 1 can be inferred using the box position of level 2, which introduces minimal overhead to the compressed data size.

The challenge in compressing 3D data is that the boxes will have varying shapes, particularly after redundancy removal, as shown in Figure 4, especially in larger datasets. To tackle this irregularity, we propose a uniform truncation method that partitions the data into a collection of unit blocks. This approach facilitates the collective compression of boxes, irrespective of their varying and irregular shapes. This method not only boosts encoding efficiency but also reduces the compressor's launch time by eliminating the need to call the compressor separately for each unique box shape.

Subsequently, the generated unit blocks after truncation, as mentioned, can be rearranged based on the needs of a specific compressor to improve compression performance. In this work, we focus on the SZ2 compression algorithm with the Lorenzo and linear regression predictors (denoted by "SZ_L/R") [21], and the SZ compression algorithm with the spline interpolation approach (denoted by "SZ_Interp") [43]. Specifically, the SZ_L/R will first truncate the whole input data to blocks with the size of 6×6×6, and then perform Lorenzo predictor or high-dimensional linear regression on each block separately. For the SZ_L/R, we will linearize the truncated

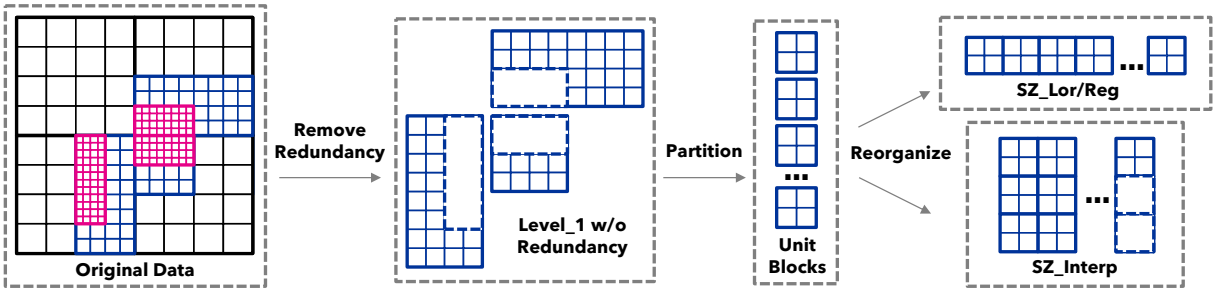
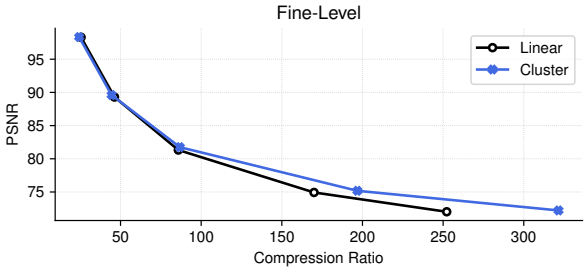
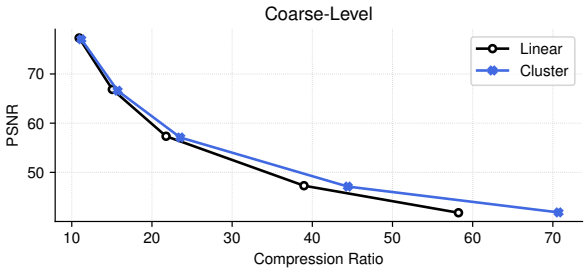


Figure 4: An example of our proposed 3D pre-processing workflow (in a top-down 2D view).



(a) Fine level (density = 17.4%)



(b) Coarse level (density = 82.3%)

Figure 5: Rate-distortion comparison between linear and cluster arrangements across different levels for Nyx's "baryon density" field. The considered relative error bounds range from 2×10^{-2} to 3×10^{-4}.

unit blocks as shown in the top right of Figure 4 (put the unit blocks along the z-axis for 3D data) because this will offer the minimum operation during the organization, thus saving time.

The SZ_Interp will perform interpolation across all three dimensions of the entire dataset. Given that interpolation is a global operation, one potential solution to improve interpolation accuracy is to cluster the truncated unit blocks more closely into a cube-like formation, as depicted in the bottom right part of Figure 4. This configuration helps balance the interpolation process across multiple dimensions, thus significantly improving the compression performance. As depicted in Figure 5, organizing unit blocks in a more compact cluster arrangement leads to enhanced overall compression performance concerning rate-distortion (PSNR[2] versus compression ratio) when compared to a linear arrangement of the blocks. This improvement is particularly noticeable when the compression ratio is relatively high. The test data is generated from a Nyx Run featuring two refinement levels: a coarse level with 256^3 grids and a fine level containing 512^3 grids. After eliminating

[2]PSNR is calculated as $20 \cdot \log_{10} R - 10 \cdot \log_{10} \left(\sum_{i=1}^{N} e_i^2 / N \right)$, where e_i is the absolute error for the point i, N is the number of points, and R is the value range of the dataset.

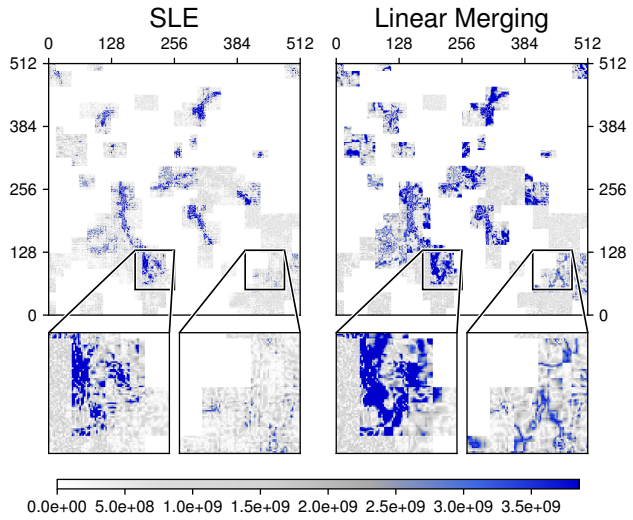


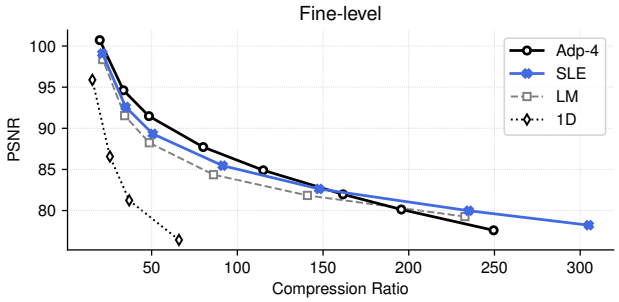
Figure 6: Visualization comparison (one slice) of absolute compression errors of unit SLE (left, CR = 91.4) and original linear merging (right, CR = 86.1) on Nyx "baryon density" field (i.e., fine level, 18% density, unit block size = 16). Bluer means higher compression error.

redundant coarse data, the coarse level has a data density of 82.3%, while the fine level has a data density of 17.4%. Here, data density refers to the proportion of data saturation within the entire domain.

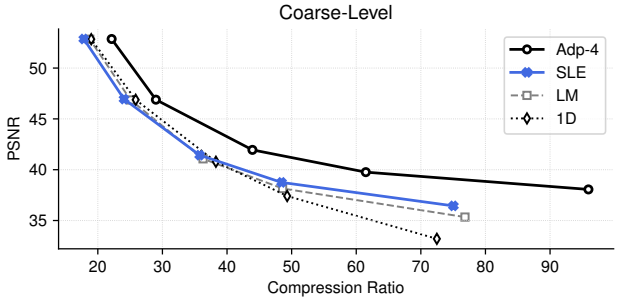
3.2 Optimization of SZ_L/R Compression

The pre-processed AMR data, however, faces two significant challenges that prevent it from achieving optimized compression quality when using the original SZ_L/R compressor. To overcome these challenges and further improve compression performance, we propose optimizing the SZ_L/R compressor. In the following paragraphs, we will outline the two challenges and describe our proposed solutions to address them effectively.

***Challenge 1**: Low prediction accuracy of SZ_L/R on AMR data.* As discussed in §3.1, the truncated unit blocks are linearized and sent to the SZ_L/R compressor. However, some merged small blocks may not be adjacent in the original dataset, resulting in poor data locality/smoothness between these non-neighboring blocks. This negatively affects the accuracy of SZ_L/R's predictor. An intuitive solution would be to compress each box individually. But, truncation can produce a large number of small data blocks (e.g., 5,000+), causing the SZ_L/R to struggle on small datasets due to low encoding efficiency, as mentioned in §2.1. This is because the SZ compressor utilizes thousands of Huffman trees to encode these



(a) Fine level (unit block size=16, density = 17.4%)



(b) Coarse level (unit block size=8, density = 82.3%)

Figure 7: Rate-distortion comparison between LM, SLE, adaptive SZ_L/R, and 1D compression, across different levels for Nyx's "baryon density" field. The relative error bound ranges from 2×10^{-2} to 3×10^{-4}.

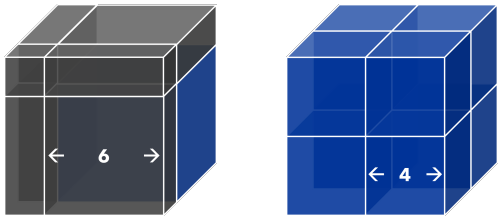
small blocks separately, leading to decreased encoding efficiency. In conclusion, the original SZ_L/R faces a dilemma: either predict and encode small blocks collectively (by merging them), which compromises prediction accuracy, or predict and encode each small block individually, incurring high Huffman encoding overhead.

***Solution 1**: Improve prediction using unit SLE.* To address Challenge 1, we propose using the Shared Lossless Encoding (SLE) technique in SZ_L/R. This method allows for separate prediction of unit data blocks while encoding them together with a single shared Huffman tree. Specifically, each unit block is initially predicted and quantized individually. Afterward, the quantization codes and regression coefficients from each unit block are combined to create a shared Huffman tree and then encoded. This approach improves the prediction performance of SZ_L/R without significantly increasing the time overhead during the encoding process.

As shown in Figure 6, the unit SLE notably reduces overall compression error in comparison to the original linear merging (LM), especially for data located at the boundaries of data blocks. As a result, this leads to a substantial improvement in rate distortion, as illustrated in Figure 7a. Note that the data used for testing in this section is the same as in §3.1. Specifically, the unit block size for the fine level is 16, while the unit block size for the coarse level is 8.

***Challenge 2**: Unit SLE may produce undesirable residues.* As previously mentioned, the input data is truncated into 6×6×6 blocks by the SZ_L/R compressor for separate processing. This block size was chosen to balance between prediction accuracy and metadata overhead, achieving optimal overall compression quality.

When using unit SLE, the compressor will further partition each of these unit blocks. The issue is that the unit block size of data produced by AMReX is typically a power of two (i.e., 2^n), which is



(a) Original partition (b) Adaptive partition

Figure 8: Example of the original partition and adaptive partition of SZ_L/R on a unit block with the size of 8×8×8; the gray boxes represent data that are difficult to compress.

not evenly divisible by 6. As a result, using a 6×6×6 cube to truncate unit blocks with specific sizes may leave undesirable residues that impact compression quality. For example, if the unit block is 8×8×8, as shown in Figure 8, SZ_L/R with unit SLE will further divide it into smaller blocks with sizes of 6×6×6 (one block), 6×6×2 (three "flat" blocks), 6×2×2 (three "slim" blocks), and 2×2×2 (one "tiny" block) as shown in Figure 8a. While the data in the 6×6×2, 6×2×2, and 2×2×2 blocks is almost flattened/collapsed to 2D data, 1D data, and a single point, respectively, rather than preserving 3D data features. These "low-dimension" data blocks can greatly affect the prediction accuracy of SZ_L/R, as they cannot leverage high-dimensional topological information. As shown in Figure 7b, when the unit block size is 8, the unit SLE approach does not appear to significantly improve the performance over the original LM method.

***Solution 2**: SZ_L/R with adaptive block size.* To address the issue of residue blocks that are difficult to compress, we propose an adaptive approach for selecting the block size used by the SZ_L/R compressor based on the unit block size of the AMR data. Equation 1 describes the adaptive block size selection method.

$$\text{SZ\_BlkSize} = \begin{cases} 4\times4\times4, & \text{if unitBlkSize} \mod 6 \le 2; \\ 6\times6\times6, & \text{if unitBlkSize} \mod 6 > 2; \\ 6\times6\times6, & \text{if unitBlkSize} \ge 64; \end{cases} \quad (1)$$

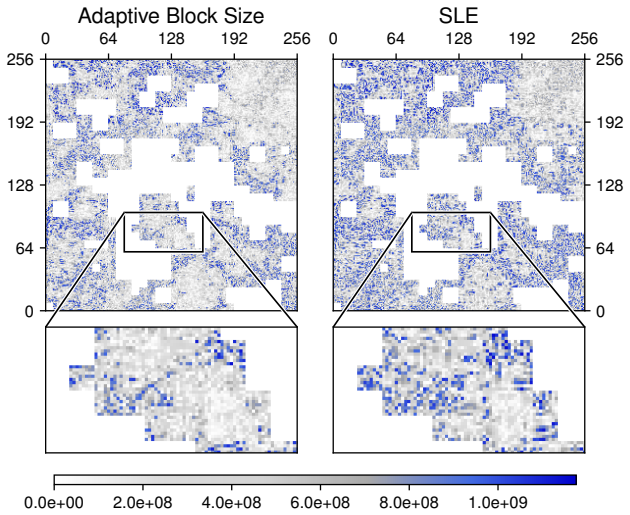
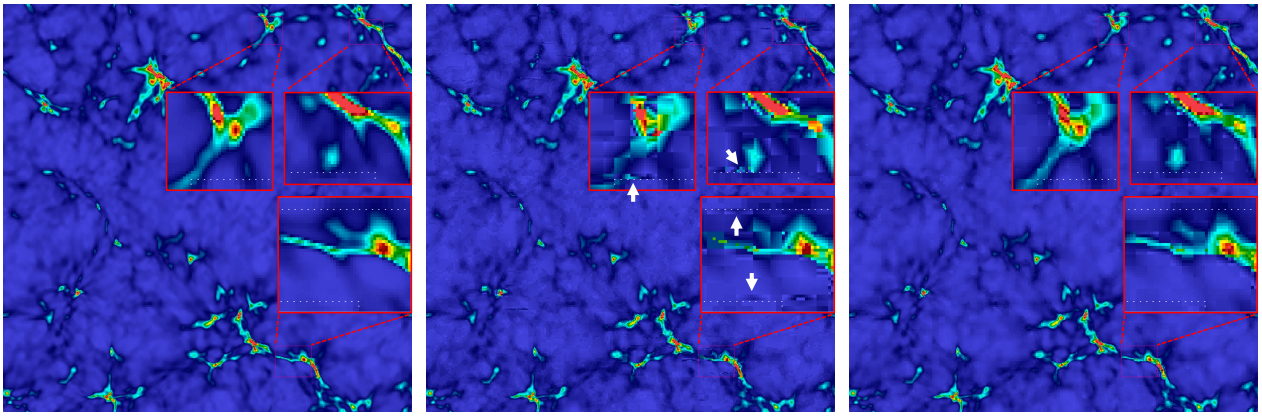


Figure 9: Visualization comparison (one slice) of compression errors of the adaptive block size (left, CR=39.8) and unit SLE (right, CR=38.8) and on Nyx "baryon density" field (i.e., coarse level, 82% density, unit block size = 8). Bluer means higher compression error.



(a) Original data (b) Original SZ_L/R, CR =51.7 (c) AMRIC SZ_L/R, CR =53.2

Figure 10: Visualization comparison (one slice) of original (uncompressed) data and decompressed data produced by original SZ_L/R and AMRIC's optimized SZ_L/R on Nyx's "baryon density" field (with 2 levels, 18% density for fine level). Warmer colors indicate higher values. The red dotted lines denote the boundaries between AMR levels, and the white arrows in Figure 10b highlight the artifacts between two AMR levels.

Specifically, if the remainder of the unit block size divided by the original SZ_L/R block size is less than or equal to 2, there will be undesirable residue blocks. In such cases, we adjust the SZ_L/R block size to be $4 \times 4 \times 4$ to avoid compressing these blocks with low compressibility and to improve prediction quality. Conversely, if the remainder is greater than 2, we use the original block size of $6 \times 6 \times 6$. For example, as shown in Figure 8b, for the 8×8×8 unit block, we have $8 \ mod \ 6 = 2$, and we will select the SZ_L/R block to be 4^3. Although using an SZ_L/R block size of 4^3 results in higher metadata overhead, the increased prediction accuracy compensates for it, achieving compression performance comparable to that of 6^3 while avoiding the undesirable residue issue. Figure 9 illustrates that the adaptive block size approach (i.e., Adp-4) can significantly reduce compression errors compared to the SLE approach, leading to a considerable enhancement in rate-distortion, as shown in Figure 7b. On the other hand, for example, when the unit block size is 16, we do not have the undesirable residues issue and there is no need to use an adaptive block size approach since it does not have an obvious advantage over the SLE approach, as shown in Figure 7a. Furthermore, note that when the unit block size is relatively large (i.e., larger than 64, which is not common for AMR data), we retain the original SZ_L/R block size. This is because even if there are undesirable residue blocks, they only occupy a small portion of the dataset, while the compression performance using 6^3 is slightly better than using 4^3, offsetting the negative effect of the few residue blocks. Note that we did not select the SZ_L/R block size to be 8^3 to eliminate the undesirable residue because it will significantly reduce compression quality.

Improvement in Visualization Quality: It is worth noting that compared to the original SZ_L/R, AMRIC's optimized SZ_L/R notably enhances the visualization quality of AMR data, particularly in areas with intense data fluctuations. For example, as shown in Figure 10, our optimized SZ_L/R effectively reduces the artifacts at the boundaries between different AMR levels (denoted by the white dotted lines), which were previously caused by the original SZ_L/R (as indicated by the white arrows in Figure 10b).

3.3 Modification of HDF5 Compression Filter

In this work, we use the HDF5 compression filter to enhance I/O performance and increase usability. However, as mentioned in §2.1, there are barriers between the HDF5 filter and the AMR application. Specifically, when employing the compression filter, HDF5 splits the in-memory data into multiple chunks, with filters being applied to each chunk individually. This makes it challenging to determine an appropriate large chunk size in order to improve the compression ratio and I/O performance. We face two primary obstacles when attempting to use a larger chunk size, and we will discuss each of these challenges along with their solutions.

Challenge 1*: AMR data layout issue for multiple fields.* As discussed in §3.1, AMReX (Patch-based AMR) divides each AMR level's domain into a collection of rectangular boxes/patches, with each box typically containing data from multiple fields. Consequently, in the AMReX framework, data corresponding to various fields within each box are stored continuously, rather than being stored in a separate manner. For instance, as illustrated in the upper portion of Figure 11, we have three boxes and two fields (i.e., Temp for temperature and Vx for velocities in the x direction), and the data for Temp and Vx in different boxes are placed together. In this situation, when determining the HDF5 chunk size, it cannot exceed the size of the smallest box (i.e., Box-1).

This limitation arises because we want to avoid compressing different fields together using lossy compression, as it would lead to the usage of identical error bounds across various fields, despite their potentially significant differences in value ranges. Additionally, combining data from distinct fields can compromise data smoothness and negatively impact the compressor's performance.

As a result, the original AMReX could only utilize a small chunk size (i.e., 1024), which significantly increased the encoding overhead of compressing each small chunk separately and led to a lower compression ratio. Moreover, the compressor had to be called for each small chunk, substantially raising the overall startup cost of the compressor and adversely affecting I/O performance.

Solution 1*: Change data layout.* A potential solution to this issue is to separate data from different fields into distinct buffers for compression. However, this approach requires compressing and

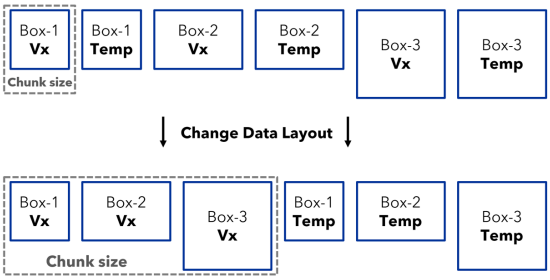


Figure 11: An example of an AMR dataset with two fields and three boxes, illustrating our data layout modification that applies a larger chunk size (indicated by the grey dashed line).

writing multiple buffers into multiple HDF5 datasets simultaneously, resulting in reduced performance for HDF5 collective writes. Based on our observations, compressing and writing AMR data into multiple HDF5 datasets can be up to 5× slower than processing them collectively.

To address this problem, we propose continuing to compress and write data into a single HDF5 dataset, while modifying the data layout to group data from the same field of each box together, as depicted in the lower portion of Figure 11. This method allows us to increase the chunk size and compress the entire field as a whole. It is important to note that we achieved this by altering the loop access order when reading the data into the buffer, which adds minimal time overhead, rather than reorganizing the buffer itself. By increasing the chunk size, we can significantly enhance both the compression and I/O performance (will be shown in §4).

***Challenge 2**: Load imbalance for AMR data.* Another challenge that prevents the adoption of a large chunk size is the load imbalance issue for AMR data across multiple processes. Given that the entire HDF5 dataset has to use the same chunk size, selecting an optimal global chunk size in a parallel scenario becomes difficult, as the data size on each MPI rank may vary.

For example, as shown in Figure 12, we have 4 ranks that hold different data in the memory. Without loss of generality and for clearer demonstration, we suppose there is only one field. If we set the chunk size to be the largest data size in all the ranks (i.e., rank 1), there would be overhead in the other 3 ranks (i.e., we have to pad useless data onto the other 3 ranks). Clearly, This will make the compressor handle extra data and impact the compression ratio as well as the I/O time.

Another intuitive solution to this is to let each rank write its data to its own dataset. In this way, each rank does not have to use the same global chunk size and can select its own chunk size based on the data size. The problem is that due to the usage of the filter, HDF5 has to perform the collective write, which means all processes need to participate in creating and writing each dataset. For example, when rank 0 is writing its data to dataset 0, the other 3 ranks also need to participate even if they have no data to be written to dataset 0. As a result, the other 3 ranks will be idle and wait for the current write (to dataset 0) to finish before proceeding with its own write, causing a serial write with poor performance.

***Solution 2**: Modify the HDF5 filter mechanism.* To address the above issue, we propose to still use the global chunk size, which is equal to the largest data size across all ranks. However, we modify the compression filter and provide the actual data size of each

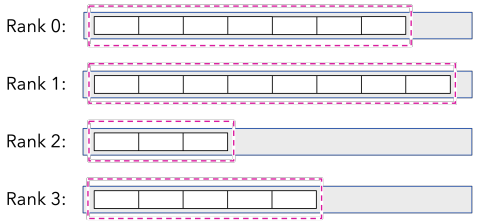


Figure 12: An example with four MPI ranks that hold a different amount of AMR data to demonstrate our proposed chunk size selection strategy. We select the chunk size to be the data size of rank 1 (outer blue box) while passing the actual data size to the compression filter (enclosing magenta dashed box).

rank (as shown in the magenta dashed box in Figure 12) to the compression filter before the compression process.

It should be noted that we also need to store metadata such as the value of the original data size for each rank for decompression purposes. The metadata overhead is minimal since the data component far outweighs the metadata to be written. This results in nearly no size overhead while improving the overall compression ratio and I/O time by adapting the biggest possible chunk size.

4 EXPERIMENTAL EVALUATION

4.1 Experimental Setup

AMR applications. Our evaluation primarily focuses on two AMR applications developed by the AMReX framework [42]: Nyx cosmology simulation [28] and the WarpX [13] electromagnetic and electrostatic Particle-In-Cell (PIC) simulation. Nyx, as shown in Figure 13, is a cutting-edge cosmology code that employs AMReX and combines compressible hydrodynamic equations on a grid with a particle representation of dark matter. Nyx generates six fields, including baryon density, dark matter density, temperature, and velocities in the x, y, and z directions. WarpX, as shown in Figure 14, is a highly-parallel and highly-optimized code that utilizes AMReX, runs on GPUs and multi-core CPUs, and features load-balancing capabilities. WarpX can scale up to the world's largest supercomputer and was the recipient of the 2022 ACM Gordon Bell Prize [30].

Test platform. Our test platform is the Summit supercomputer [29] at Oak Ridge National Laboratory, each node of which is equipped with two IBM POWER9 processors with 42 physical cores and 512 GB DDR4 memory. It is connected to an IBM Spectrum Scale filesystem [39]. We use up to 128 nodes and 4096 CPU cores.

Comparison baseline. We compare our solution with AMReX's 1D SZ_L/R compression solution [3] (denoted by "AMReX"). We exclude zMesh and TAC because they are not in situ compression solutions, as mentioned in §1. Note that we evaluate AMRIC with both SZ_L/R (SZ with Lorenzo and linear regression predictors) and SZ_Interp (SZ with spline interpolation predictor).

Test runs. As shown in Table 1, we have conducted six simulation runs in total, with three runs for each of the scientific applications, WarpX (WarpX_1, WarpX_2, and WarpX_3) and Nyx (Nyx_1, Nyx_2, and Nyx_3). Each simulation run consists of two levels, and the number of nodes (ranks) varying from 2 (64), to 16 (512), up to 128 (4096). In WarpX_1, the grid sizes of the levels progress from coarse to fine, with the dimensions of $256\times256\times2048$ and $512\times512\times4096$, and the data densities of 98.04% and 1.96%, respectively. The data size for one timestep in this run is 12.4 GB. For

Table 1: Detailed information about our tested AMR runs.

Runs	#AMR Levels	#Nodes (#MPI ranks)	Grid size of each level (coarse to fine)	Density of each level (coarse to fine)	Data size (each timestep)	Error bound (AMRIC and AMReX)
Warpx_1	2	2 (64)	256×256×2048, 512×512×4096	98.04%, 1.96%	12.4 GB	1E-3, 5E-3
Warpx_2	2	16 (512)	512×512×4096, 1024×1024×8192	98.05%, 1.96%	99.3 GB	1E-3, 5E-3
Warpx_3	2	128 (4096)	1024×1024×8192, 2048×2048×16384	98.96%, 1.04%	624 GB	1E-4, 5E-4
Nyx_1	2	2 (64)	256×256×256, 512×512×512	98.6%, 1.4%	1.6 GB	1E-3, 1E-2
Nyx_2	2	16 (512)	512×512×512, 1024×1024×1024	96.67%, 3.23%	12 GB	1E-3, 1E-2
Nyx_3	2	128 (4096)	1024×1024×1024, 2048×2048×2048	98.3%, 1.7%	97.5 GB	1E-3, 1E-2

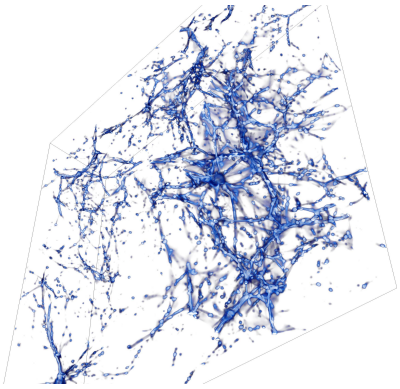


Figure 13: Visualization of baryon density field of Nyx.

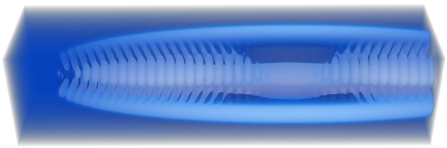


Figure 14: Visualization of the electric field (x-direction) of WarpX.

WarpX_2, the grid sizes are $512 \times 512 \times 4096$ and $1024 \times 1024 \times 8192$, accompanied by the data densities of 98.05% and 1.96%. The data size for one timestep is 99.3 GB. For WarpX_3, the grid sizes are $1024 \times 1024 \times 8192$ and $2048 \times 2048 \times 16384$, accompanied by the data densities of 98.96% and 1.04%. The data size for one timestep is 624 GB. Regarding Nyx_1, the grid sizes of the levels, from coarse to fine, are $256 \times 256 \times 256$ and $512 \times 512 \times 512$, with data densities of 98.6% and 1.4%, respectively. The data size for one timestep in this run is 1.6 GB. For Nyx_2, the grid sizes are $512 \times 512 \times 512$ and $1024 \times 1024 \times 1024$, and the data densities are 96.67% and 3.23%. The data size for one timestep is 12 GB. For Nyx_3, the grid sizes are $1024 \times 1024 \times 1024$ and $2048 \times 2048 \times 2048$, and the data densities are 98.3% and 1.7%. The data size for one timestep is 97.5 GB. Finally, as Summit uses a shared parallel filesystem that fluctuates in performance depending on the I/O load across the overall user-base, we run each set of simulation runs multiple times and discard the results with abnormal performance (extremely slow).

4.2 Evaluation on Compression Ratio

As demonstrated in Table 2, our method, which includes optimized SZL/R and optimized SZ_Interp, outperforms the original 1D baseline for all three runs of the two applications, with a particularly notable improvement in WarpX. This superior performance is primarily due to our 3D compression's ability to leverage spatial and topological information, thereby enhancing the compression process. Moreover, the optimizations for SZ_L/R and SZ_Interp outlined in §3 further improve their respective compression performances. Furthermore, as mentioned in §3.3, the small chunk size of the original AMReX's compression leads to substantial encoding overhead, which ultimately results in a lower compression ratio.

Upon further analysis, we observe that WarpX exhibits a notably high compression ratio. This is mainly due to the smooth nature of the data generated by WarpX, as depicted in Figure 14, which results in excellent compressibility. In contrast, the data produced by Nyx appears irregular, as illustrated in Figure 13, making it more challenging to compress. Consequently, our AMRIC method has the potential to significantly reduce I/O time for WarpX by achieving greater data size reduction. Importantly, despite the difficulty in compressing Nyx data, our AMRIC will not introduce significant overhead to Nyx simulation, as will be demonstrated in §4.4.

In terms of specific performance, SZL/R proves more effective in handling Nyx data, while SZ_Interp delivers superior compression ratios for WarpX. This can be attributed to SZL/R's block-based predictor, which is better suited to capturing local patterns within Nyx data, whereas SZ_Interp's global interpolation predictor excels when applied to the overall smoother data produced by WarpX.

Table 2: Comparison of compression ratio (averaged across all fields/timesteps) with AMReX's original compression and AMRIC.

Run	AMReX(1D)	AMRIC(SZ_L/R)	AMRIC(SZ_Interp)
WarpX_1	16.4	267.3	**482.1**
WarpX_2	117.5	461.2	**2406.0**
WarpX_3	29.6	949.0	**4753.7**
Nyx_1	8.8	**15.0**	14.0
Nyx_2	8.8	**16.6**	14.2
Nyx_3	8.7	**16.3**	13.6

4.3 Evaluation on Reconstruction Data Quality

As shown in Table 3, the reconstruction data quality of AMRIC (including both SZ_L/R and SZ_Interp) is greatly higher than that of AMReX due to our optimization as well as the benefit of the 3D compression. As shown in Figure 15, the error generated by AMRIC is considerably lower than that of AMReX. Note that the existing block-like pattern of the error is from the parallel compression done by 512 processes, each assigned with an error bound relative to the data range in the corresponding block. Therefore, the absolute error of each block is independent of those of the other blocks.

Comparison with offline solution TAC [40]: To further demonstrate the effectiveness of AMRIC's optimized SZ_L/R, we conduct

Table 3: Comparison of reconstruction data quality (in PSNR) with AMReX's original compression and AMRIC for different runs.

Run	AMReX(1D)	AMRIC(SZ_L/R)	AMRIC(SZ_Interp)
Nyx_1	52.5	**66.8**	66.5
Nyx_2	56.7	**69.1**	68.9
Nyx_3	54.9	**68.3**	68.0
WarpX_1	73.6	**80.3**	79.9
WarpX_2	78.5	83.8	**88.7**
WarpX_3	82.5	97.9	**103.1**

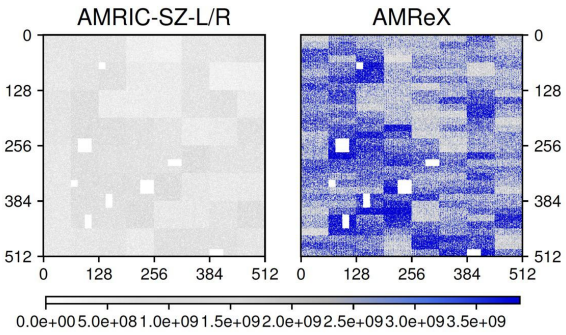


Figure 15: Visualization comparison (one slice) of compression errors of our AMRIC (left) and AMReX's compression (right) on Nyx_2 "baryon density" field (i.e., coarse level, 96% density, unit block size=32). Bluer/darker means higher compression error.

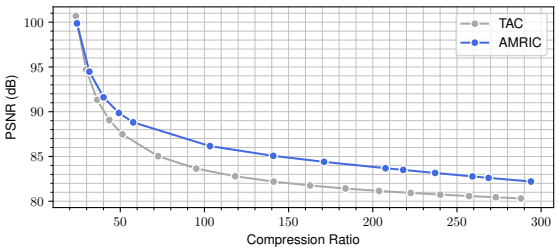


Figure 16: Rate-distortion comparison of TAC and AMRIC using TAC's dataset [17] (i.e., Run1_Z10).

a comparison with a state-of-the-art offline compression approach for 3D AMR data, called TAC (will be introduced in §5), using the dataset from TAC's work. As depicted in Figure 16, AMRIC outperforms TAC in terms of compression quality, achieving up to a 2.2× higher compression ratio while maintaining the same PSNR. This superior performance can be attributed to the fact that, unlike TAC, which only focuses on pre-processing and uses SZ_L/R as a black box, AMRIC optimizes both the pre-processing and SZ_L/R.

Insight of different compressors' performance on AMR simulations. One takeaway is that, compared with SZ_Interp, SZ_L/R is more suitable for compressing AMR data because both AMR data and SZ_L/R are block-based, while SZ_Interp is global. Specifically, AMR simulations divide the data into boxes, which can negatively impact data locality and smoothness. SZ_L/R, on the other hand, also truncates data into blocks, aligning with AMR simulations. By applying our optimization approach outlined in §3.2, SZ_L/R with SLE and adaptive block size can effectively mitigate the impact on data locality/smoothness caused by AMR applications, resulting in ideal compatibility with AMR applications. On the other hand, since SZ_Interp applies global interpolation to the unstructured block-based AMR data, it is challenging to achieve perfect compatibility between SZ_Interp and AMR applications.

4.4 Evaluation on I/O Time

The overall writing time consists of: (1) pre-processing (including copying data to the HDF5 buffer, handling metadata, calculating the offset for each process) and I/O time without compression (directly writing the data to the disk); or (2) pre-processing and I/O time with compression (including compression computation cost and writing the compressed data to the file system).

Figure 17: Writing time of WarpX runs with different scales (in a weak scaling study). Log scale is used here for better comparison.

Figure 18: Writing time of Nyx runs with different scales. Log scale is used for better comparison.

Figure 17 shows that our in situ compression significantly reduces the overall writing time for HDF5 when compared to writing data without compression. This reduction reaches up to 90% for the largest-scale WarpX run3, and 64% for the larger-scale WarpX run2, without introducing any noticeable overhead for the smaller-scale WarpX run1. This is because, for the relatively large-scale WarpX run3 and run2, the total data to be written amounts to 624 GB and 99 GB respectively. Due to the high compression ratio, we can significantly save both writing time and overall processing time.

Our approach also considerably reduces the total writing time by 97% compared to the original AMReX's compression for WarpX run1, 93% for WarpX run2, and 89% for WarpX run3. We can find out that AMReX's compression is extremely slow on WarpX. This is because, first, as discussed in §3.3 (Challenge 1), the original compression of AMReX is constrained by the existing data layout, which necessitates the use of a small HDF5 chunk size. This constraint negatively impacts compression time and ratio, causing suboptimal I/O performance. Moreover, this negative impact becomes more severe when there are relatively larger amounts of data in each process. Specifically, for all three WarpX runs, each process contains at least 128^3 data points (considering only the coarse level). Consequently, given that the HDF5 chunk size for the original AMReX compression is 1024, each process will call the compressor 2048 times, generating a substantial startup cost and leading to extremely slow I/O. For WarpX run3, an HDF5 chunk size of 1024 causes issues, so we instead use 4096 as the chunk size. This issue further emphasizes the significance of our modifications presented in §3.3, which effectively utilize the HDF5 compression filter. It is worth noting that the impact of this small chunk issue will be relatively mitigated when there are fewer data points in each process (as demonstrated in the subsequent evaluation of Nyx).

Also, note that our proposed method does not introduce any significant overhead to the pre-processing time. This is primarily because our pre-processing strategy is lightweight, employing AMReX's built-in functions to identify redundant coarse data. Furthermore, the truncation, block ordering in the pre-processing stage, and data layout changes in §3.3 are performed simultaneously when loading data into the compression buffer, eliminating the need for extra rearrangement operations. In addition, the elimination of redundant data using the pre-processing workflow reduces the amount of data to be processed, lowering the pre-processing time.

To further demonstrate the performance of our method on data with low compressibility as well as when each process owns fewer data, we conduct our test using Nyx with a smaller scale but the same number of nodes (ranks). In our Nyx runs, each process owns $64 \times 64 \times 64$ data points in the coarse level, which is 8 times less than that of WarpX runs (i.e., $128 \times 128 \times 128$). Note that a smaller data size in each process will lower the overall lossless encoding efficiency of the data, thus affecting the compression ratio.

As shown in Figure 18, even in a challenging setup (i.e., low data compressibility and low encoding efficiency), AMRIC is still able to achieve writing speeds comparable to those with no compression for all three Nyx runs. Furthermore, AMRIC significantly reduces the total writing time by 79% compared to the original AMReX's compression for Nyx run1, by 53% for Nyx run2, and by 64% for Nyx run3. It is worth noting that the small chunk issue in the Nyx run is relatively mitigated compared to WarpX. Specifically, we observe that the writing time using AMReX's compression for both 2-node and 16-node runs on Nyx is reduced by approximately 55 seconds, while the reduction for the 128-node run is approximately 10 seconds. Our interpretation is that the time taken to launch the compression once remains constant (e.g., 0.03 seconds). In the Nyx run, each process needs to call the compressor only 256 times, as opposed to the 2048 calls required in the WarpX run1 and run2 (due to the increased HDF5 chunk size as aforementioned, WarpX run3 only requires 512 calls.). This difference results in a time reduction of $(2048 - 128) * 0.03 \approx 55$ seconds for the writing process for WarpX run1 and run2, and $(512 - 128) * 0.03 \approx 10$ for WarpX run3.

5 RELATED WORK

There are two recent works focusing on designing efficient lossy compression methods for AMR datasets. Specifically, zMesh, proposed by Luo *et al.* [27], aims to leverage the data redundancy across different AMR levels. It reorders the AMR data across different refinement levels in a 1D array to improve the smoothness of the data. To achieve this, zMesh arranges adjacent data points together in the 1D array based on their physical coordinates in the original 2D/3D dataset. However, by compressing the data in a 1D array, zMesh cannot leverage higher-dimension compression, leading to a loss of topology information and data locality in higher-dimension data. On the other hand, TAC, proposed by Wang *et al.* [40], was designed to improve zMesh's compression quality through adaptive 3D compression. Specifically, TAC pre-processes AMR data before compression by adaptively partitioning and padding based on the data characteristics of different AMR levels.

While zMesh and TAC provide offline compression solutions for AMR data, they are not designed for in situ compression of AMR data. In particular, zMesh requires extra communication to perform reordering in parallel scenarios. Specifically, zMesh must arrange neighboring coarse and fine data more closely. However, the neighboring fine and coarse data might not be owned by the same MPI rank. As a result, data from different levels must be transferred to the appropriate processes, leading to high communication overhead. TAC requires the reconstruction of the entire physical domain's hierarchy to execute its pre-processing approach. This relatively complex process results in significant overhead for in situ data compression.

Although the AMReX framework [42] supports in situ AMR data compression through HDF5 compression filters [3], it has two main drawbacks: (1) The original AMReX compression only compresses the data in 1D, limiting its capacity to benefit from higher-dimension compression and resulting in sub-optimal compression performance, particularly in terms of compression quality. (2) The original AMReX compression cannot effectively utilize the HDF5 filter. Specifically, due to the limitation of the data layout for multiple fields, AMReX can only adopt a very small chunk size to prevent compressing different physical fields together. As a result, AMReX needs to apply the compressor separately for each small chunk, resulting in low I/O and compression performance as demonstrated in §4.4, §4.2, and §4.3.

6 CONCLUSION AND FUTURE WORK

In conclusion, we have presented AMRIC, an effective in situ lossy compression framework for AMR simulations that significantly enhances I/O performance and compression quality. Our primary contributions include designing a compression-oriented in situ pre-processing workflow for AMR data, optimizing the state-of-the-art SZ lossy compressor for AMR data, efficiently utilizing the HDF5 compression filter on AMR data, and integrating AMRIC into the AMReX framework. We evaluated AMRIC on two real-world AMReX applications, WarpX and Nyx, using 4096 CPU cores from the Summit supercomputer. The experimental results demonstrate that AMRIC achieves up to 10.5× I/O performance improvement over the non-compression solution and up to 39× I/O performance improvement and up to 81× compression ratio improvement with better data quality over the original AMReX's compression solution.

In future work, we plan to evaluate AMRIC on additional AMReX applications accelerated by GPUs. Furthermore, we will assess AMRIC on a wider range of HPC systems and at different scales. Additionally, we will incorporate our in situ compression solution into other AMR frameworks.

ACKNOWLEDGEMENT

This work (LA-UR-23-24096) has been authored by employees of Triad National Security, LLC, which operates Los Alamos National Laboratory under Contract No. 89233218CNA000001 with the U.S. Department of Energy (DOE) and National Nuclear Security Administration (NNSA). The material was supported by the U.S. DOE Office of Science (SC), Office of Advanced Scientific Computing Research (ASCR), under contracts DE-AC02-06CH11357 and DE-AC02-05CH11231, and under award 66150: "CENATE - Center for Advanced Architecture Evaluation". The Pacific Northwest National Laboratory is operated by Battelle for the U.S. Department of Energy under contract DE-AC05-76RL01830. This research was supported by the Exascale Computing Project (ECP), Project Number: 17-SC-20-SC, a collaborative effort of the DOE SC and NNSA. This work was also supported by NSF awards 2003709, 2303064, 2104023, 2247080, 2311875, 2311876, 2312673. This research used resources of the National Energy Research Scientific Computing Center, a DOE SC User Facility located at Lawrence Berkeley National Laboratory, operated under Contract No. DE-AC02-05CH11231.

REFERENCES

[1] 2023. *HDF5 Filters.* https://docs.hdfgroup.org/hdf5/develop/_f_i_l_t_e_r.html Online.

[2] Mark Ainsworth, Ozan Tugluk, Ben Whitney, and Scott Klasky. 2018. Multilevel techniques for compression and reduction of scientific data—the univariate case. *Computing and Visualization in Science* 19, 5–6 (2018), 65–76.

[3] AMReX - HDF5 Plotfile Compression. 2023. https://amrex-codes.github.io/amrex/docs_html/IO.html#hdf5-plotfile-compression. Online.

[4] AMReX's documentation. 2023. https://amrex-codes.github.io/amrex/docs_html/Basics.html#boxarray. Online.

[5] Allison H Baker, Dorit M Hammerling, and Terece L Turton. 2019. Evaluating image quality measures to assess the impact of lossy data compression applied to climate simulation data. In *Computer Graphics Forum*, Vol. 38. Wiley Online Library, 517–528.

[6] Kevin J Bowers, BJ Albright, L Yin, B Bergen, and TJT Kwan. 2008. Ultrahigh performance three-dimensional electromagnetic relativistic kinetic plasma simulation. *Physics of Plasmas* 15, 5 (2008), 055703.

[7] Franck Cappello, Sheng Di, Sihuan Li, Xin Liang, Ali Murat Gok, Dingwen Tao, Chun Hong Yoon, Xin-Chuan Wu, Yuri Alexeev, and Frederic T Chong. 2019. Use cases of lossy compression for floating-point data in scientific data sets. *The International Journal of High Performance Computing Applications* (2019).

[8] cuZFP. 2023. https://github.com/LLNL/zfp/tree/develop/src/cuda_zfp. Online.

[9] Sheng Di. 2023. *H5Z-SZ.* https://github.com/disheng222/H5Z-SZ Online.

[10] Sheng Di and Franck Cappello. 2016. Fast error-bounded lossy HPC data compression with SZ. In *2016 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 730–739.

[11] Sheng Di and Franck Cappello. 2016. Fast error-bounded lossy HPC data compression with SZ. In *2016 IEEE International Parallel and Distributed Processing Symposium*. IEEE, IEEE, Chicago, IL, USA, 730–739.

[12] Bo Fang, Daoce Wang, Sian Jin, Quincey Koziol, Zhao Zhang, Qiang Guan, Surendra Byna, Sriram Krishnamoorthy, and Dingwen Tao. 2021. Characterizing Impacts of Storage Faults on HPC Applications: A Methodology and Insights. 409–420. https://doi.org/10.1109/Cluster48925.2021.00048

[13] L. Fedeli, A. Huebl, F. Boillod-Cerneux, T. Clark, K. Gott, C. Hillairet, S. Jaure, A. Leblanc, R. Lehe, A. Myers, C. Piechurski, M. Sato, N. Zaim, W. Zhang, J. Vay, and H. Vincenti. 2022. Pushing the Frontier in the Design of Laser-Based Electron Accelerators with Groundbreaking Mesh-Refined Particle-In-Cell Simulations on Exascale-Class Supercomputers. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society, Los Alamitos, CA, USA, 1–12. https://doi.org/10.1109/SC41404.2022.00008

[14] Mike Folk, Gerd Heber, Quincey Koziol, Elena Pourmal, and Dana Robinson. 2011. An overview of the HDF5 technology suite and its applications. In *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*. 36–47.

[15] Pascal Grosset, Christopher Biwer, Jesus Pulido, Arvind Mohan, Ayan Biswas, John Patchett, Terece Turton, David Rogers, Daniel Livescu, and James Ahrens. 2020. Foresight: analysis that matters for data reduction. In *2020 SC20: International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. IEEE Computer Society, 1171–1185.

[16] Guénolé Harel, Jacques-Bernard Lekien, and Philippe P Pébaÿ. 2017. Two new contributions to the visualization of AMR grids: I. interactive rendering of extreme-scale 2-dimensional grids ii. novel selection filters in arbitrary dimension. *arXiv preprint arXiv:1703.00212* (2017).

[17] hipdac tac. 2023. https://github.com/hipdac-lab/HPDC22-TAC. Online.

[18] Sian Jin, Pascal Grosset, Christopher M Biwer, Jesus Pulido, Jiannan Tian, Dingwen Tao, and James Ahrens. 2020. Understanding GPU-Based Lossy Compression for Extreme-Scale Cosmological Simulations. *arXiv preprint arXiv:2004.00224* (2020).

[19] Sian Jin, Jesus Pulido, Pascal Grosset, Jiannan Tian, Dingwen Tao, and James Ahrens. 2021. Adaptive Configuration of In Situ Lossy Compression for Cosmology Simulations via Fine-Grained Rate-Quality Modeling. *arXiv preprint arXiv:2104.00178* (2021).

[20] Sian Jin, Dingwen Tao, Houjun Tang, Sheng Di, Suren Byna, Zarija Lukic, and Franck Cappello. 2022. Accelerating parallel write via deeply integrating predictive lossy compression with HDF5. *arXiv preprint arXiv:2206.14761* (2022).

[21] Xin Liang, Sheng Di, Dingwen Tao, Sihuan Li, Shaomeng Li, Hanqi Guo, Zizhong Chen, and Franck Cappello. 2018. Error-controlled lossy compression optimized for high compression ratios of scientific datasets. In *2018 IEEE International Conference on Big Data*. IEEE, 438–447.

[22] Xin Liang, Kai Zhao, Sheng Di, Sihuan Li, Robert Underwood, Ali M. Gok, Jiannan Tian, Junjing Deng, Jon C. Calhoun, Dingwen Tao, Zizhong Chen, and Franck Cappello. 2022. SZ3: A Modular Framework for Composing Prediction-Based Error-Bounded Lossy Compressors. *IEEE Transactions on Big Data* (2022), 1–14. https://doi.org/10.1109/TBDATA.2022.3201176

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

[24] Peter Lindstrom. 2023. *H5Z-ZFP.* https://github.com/LLNL/H5Z-ZFP Online.

[25] Tao Lu, Qing Liu, Xubin He, Huizhang Luo, Eric Suchyta, Jong Choi, Norbert Podhorszki, Scott Klasky, Mathew Wolf, Tong Liu, et al. 2018. Understanding and modeling lossy compression schemes on HPC scientific data. In *2018 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 348–357.

[26] Huizhang Luo, Dan Huang, Qing Liu, Zhenbo Qiao, Hong Jiang, Jing Bi, Haitao Yuan, Mengchu Zhou, Jinzhen Wang, and Zhenlu Qin. 2019. Identifying Latent Reduced Models to Precondition Lossy Compression. In *2019 IEEE International Parallel and Distributed Processing Symposium*. IEEE.

[27] Huizhang Luo, Junqi Wang, Qing Liu, Jieyang Chen, Scott Klasky, and Norbert Podhorszki. 2021. zMesh: Exploring Application Characteristics to Improve Lossy Compression Ratio for Adaptive Mesh Refinement. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 402–411.

[28] NYX simulation. 2019. https://amrex-astro.github.io/Nyx/. Online.

[29] Oak Ridge Leadership Computing Facility. [n.d.]. *Summit Supercomputer.* https://www.olcf.ornl.gov/summit/

[30] Oak Ridge Leadership Computing Facility. 2023. *WarpX, granted early access to the exascale supercomputer Frontier, receives the high-performance computing world's highest honor.* https://www.olcf.ornl.gov/2022/11/17/plasma-simulation-code-wins-2022-acm-gordon-bell-prize/ Online.

[31] Russ Rew and Glenn Davis. 1990. NetCDF: an interface for scientific data access. *IEEE computer graphics and applications* 10, 4 (1990), 76–82.

[32] James M Stone, Kengo Tomida, Christopher J White, and Kyle G Felker. 2020. The Athena++ adaptive mesh Refinement framework: Design and magnetohydrodynamic solvers. *The Astrophysical Journal Supplement Series* 249, 1 (2020), 4.

[33] Dingwen Tao, Sheng Di, Zizhong Chen, and Franck Cappello. 2017. Significantly improving lossy compression for scientific data sets based on multidimensional prediction and error-controlled quantization. In *2017 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 1129–1139.

[34] Dingwen Tao, Sheng Di, Zizhong Chen, and Franck Cappello. 2017. Significantly improving lossy compression for scientific data sets based on multidimensional prediction and error-controlled quantization. In *2017 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 1129–1139.

[35] Dingwen Tao, Sheng Di, Xin Liang, Zizhong Chen, and Franck Cappello. 2019. Optimizing lossy compression rate-distortion from automatic online selection between SZ and ZFP. *IEEE Transactions on Parallel and Distributed Systems* 30, 8 (2019), 1857–1871.

[36] The HDF Group. 2023. *Hierarchical data format version 5.* http://www.hdfgroup.org/HDF5 Online.

[37] Jiannan Tian, Sheng Di, Xiaodong Yu, Cody Rivera, Kai Zhao, Sian Jin, Yunhe Feng, Xin Liang, Dingwen Tao, and Franck Cappello. 2021. Optimizing error-bounded lossy compression for scientific data on GPUs. In *2021 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 283–293.

[38] Jiannan Tian, Sheng Di, Kai Zhao, Cody Rivera, Megan Hickman Fulp, Robert Underwood, Sian Jin, Xin Liang, Jon Calhoun, Dingwen Tao, and Franck Cappello. 2020. cuSZ: An Efficient GPU-Based Error-Bounded Lossy Compression Framework for Scientific Data. (2020), 3–15.

[39] Marc-André Vef. 2016. Analyzing file create performance in IBM spectrum scale. *Master's thesis, Johannes Gutenberg University Mainz* (2016).

[40] Daoce Wang, Jesus Pulido, Pascal Grosset, Sian Jin, Jiannan Tian, James Ahrens, and Dingwen Tao. 2022. TAC: Optimizing Error-Bounded Lossy Compression for Three-Dimensional Adaptive Mesh Refinement Simulations. In *Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing*. 135–147.

[41] Feng Wang, Nathan Marshak, Will Usher, Carsten Burstedde, Aaron Knoll, Timo Heister, and Chris R. Johnson. 2020. CPU Ray Tracing of Tree-Based Adaptive Mesh Refinement Data. *Computer Graphics Forum* 39, 3 (2020), 1–12.

[42] Weiqun Zhang, Ann Almgren, Vince Beckner, John Bell, Johannes Blaschke, Cy Chan, Marcus Day, Brian Friesen, Kevin Gott, Daniel Graves, et al. 2019. AMReX: a framework for block-structured adaptive mesh refinement. *Journal of Open Source Software* 4, 37 (2019), 1370–1370.

[43] Kai Zhao, Sheng Di, Maxim Dmitriev, Thierry-Laurent D Tonellot, Zizhong Chen, and Franck Cappello. 2021. Optimizing error-bounded lossy compression for scientific data by dynamic spline interpolation. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 1643–1654.

Appendix: Artifact Description

ARTIFACT DOI

10.5281/zenodo.8080406

ARTIFACT IDENTIFICATION

A.1 Abstract of Artifacts

AMRIC is a novel in-situ lossy compression framework that leverages the HDF5 filter to enhance both I/O efficiency and compression quality for Adaptive Mesh Refinement (AMR) applications. In this report, we utilize the computational artifact to demonstrate and reproduce the performance improvements (covering I/O costs and compression quality) of our proposed AMRIC (including SZ-L/R and SZ-Interp) in comparison to the original AMReX compression approach. Furthermore, we present the I/O cost comparison between AMRIC and the non-compression solution, using the artifact. The reproduced results closely align with those presented in the paper, and the computational artifact can be easily deployed and executed using container technology.

A.2 Description of Artifacts

The primary components of our computational artifact's source code encompass the preprocessing of AMR data, optimization of SZ compression, and modifications to the HDF5 compression filter, as detailed in Sections 3.1, 3.2, and 3.3 of the paper. We have integrated our source code within the AMReX framework, SZ compressor, and HDF5 SZ compression filter (i.e., H5Z-SZ) respectively.

Moreover, this computational artifact incorporates the AMReX framework as well as two real-world AMR applications, Nyx and WarpX. It also includes two SZ compression algorithms (i.e., SZ-L/R and SZ-Interp) and H5Z-SZ3. Here SZ-L/R refers to the SZ compressor with Lorenzo predictor or regression, while SZ-Interp refers to the SZ compressor with spline interpolation. The artifact further contains a dataset (i.e., checkpoint files from later timesteps) of Nyx and WarpX applications.

REPRODUCIBILITY OF EXPERIMENTS

B.1 Description of Experiment Workflow

We provide two experimental workflows. This section primarily focuses on the first workflow: utilizing the Singularity image. The second workflow, i.e., building from source, is similar but requires more time due to the installation of software libraries. For detailed commands of the second workflow as well as the dependencies and hardware requirements, please refer to the DOI link or https://github.com/hipdac-lab/SC23-AMRIC.

While preparing the artifacts, we executed them on a single node from the Chameleon Cloud (https://www.chameleoncloud.org/), equipped with two Intel Xeon Gold 6242 CPUs and 192 GB of memory (specifically, the `compute_skylake` instance). We recommend that reviewers also use the Chameleon Cloud for artifact evaluation.

Specifically, the entire workflow takes about 10 minutes, including downloading container image and preparing the environment (3 mins), running WarpX simulation (3 mins), running Nyx simulation (3 mins), and evaluating compression performance (1 min).

B.2 Systems Requirements

```
Processor: >= 8 cores
Memory: >= 32 GB RAM
Storage: >= 32 GBs
OS Systems: Ubuntu (20.04 is recommended)
```

B.3 Artifact Installation Deployment Process

(1) Install Singularity (https://singularity-tutorial.github.io/01-installation/).

(2) Download the pre-built Singularity image file.

```
git clone https://github.com/hipdac-lab\
/SC23-AMRIC-Image.git
cat SC23-AMRIC-Image/img/amric.sif-* > amric.sif
```

(3) Build and run the image file (need root privilege).

```
sudo singularity build --sandbox artiAmr amric.sif
sudo singularity shell --writable artiAmr
```

(4) Set up environmental variables.

```
export OMPI_DIR=/opt/ompi
export OMPI_VERSION=4.1.1
export PATH=$OMPI_DIR/bin:$PATH
export LD_LIBRARY_PATH=$OMPI_DIR/\
lib:$LD_LIBRARY_PATH
export MANPATH=$OMPI_DIR/share/man:$MANPATH
export C_INCLUDE_PATH=/opt/ompi/include\
:$C_INCLUDE_PATH
export CPLUS_INCLUDE_PATH=/opt/ompi/\
include:$CPLUS_INCLUDE_PATH
export OMPI_ALLOW_RUN_AS_ROOT=1
export OMPI_ALLOW_RUN_AS_ROOT_CONFIRM=1
```

(5) Run WarpX simulation with no compression, AMReX's original compression, and AMRIC (takes several minutes for running 3 iterations).

```
cd /home/wpx256/ && . bash.sh
```

(6) Run NYX simulation with no compression, AMReX's original compression, and AMRIC (takes several minutes for running 3 iterations).

```
cd /home/nyx128/ && . bash.sh
```

(7) Evaluate WarpX's data quality and compression ratio for original AMReX compression and our AMRIC.

```
cd /home/wpx256/diags/
. decomp.sh > temp.txt
. qualityCR.sh
```

(8) Evaluate NYX's data quality and compression ratio for original AMReX compression and our AMRIC.

```
cd /home/nyx128/run/
. decomp.sh > temp.txt
. qualityCR.sh
```

(9) Compare I/O performance between baselines (i.e., no compression and ori AMReX compression) and AMRIC in WarpX.

```
cd /home/wpx256/otfile/ && . io.sh
```

(10) Compare I/O performance between baselines (i.e., no compression and ori AMReX compression) and AMRIC in NYX.

```
cd /home/nyx128/otfile/ && . io.sh
```

B.4 Expected Evaluation Results

We performed two runs each on WarpX and Nyx applications (e.g., steps 5 and 6), with each run representing a weak scaling study corresponding to the respective application presented in the main paper (i.e., 8/64 times reduced scale and process number). Due to space constraints, we will only display the key portion of the expected results.

B.4.1 Evaluation of Data Quality and Compression Ratio. Steps 7 and 8 present the expected data quality and compression ratio for our AMRIC (including SZ-L/R and SZ-Interp) and the original AMReX compression for WarpX and Nyx, respectively. Our AMRIC demonstrates improved data quality and compression ratio in comparison to the original AMReX

compression for both WarpX and Nyx, which is consistent with Tables 2 and 3 in the main paper. It is important to note that the advantage of AMRIC in the expected results is slightly lower than that reported in the main paper. This can be attributed to the reduced data locality/compressibility of smaller-sized AMR data, which makes it more challenging for AMRIC to fully leverage its strengths.

The expected results for step 7 are:

```
----- Data Quality for original AMReX Compression -----
PSNR = 58.600102
---------- Data Quality for AMRIC-SZ-L/R ----------
PSNR = 61.749515
---------- Data Quality for AMRIC-SZInterp ----------
PSNR = 59.146966
---------- CR for original AMReX Compression ----------
CR is: 14.62
---------- CR for AMRIC-SZ-L/R ----------
CR is: 108.94
---------- CR for AMRIC-SZInterp ----------
CR is: 131.41
```

The expected results for Step 8 are:

```
----- Data Quality for original AMReX Compression -----
PSNR = 61.977332
---------- Data Quality for AMRIC-SZ_L/R ----------
PSNR = 66.650492
---------- Data Quality for AMRIC-SZInterp ----------
PSNR = 66.566370
---------- CR for original AMReX Compression ----------
CR is: 6.53
---------- CR for AMRIC-SZ_L/R ----------
CR is: 13.08
---------- CR for AMRIC-SZInterp ----------
CR is: 11.25
```

B.4.2 Evaluation of I/O Time. Steps 9 and 10 present the expected I/O time for the no compression method, the original AMReX compression, and our AMRIC (including SZ-L/R and SZ-Interp) for WarpX and Nyx, respectively. For Step 9 (WarpX), it is evident that AMRIC (SZ-L/R) significantly reduces the total writing time by up to 76.4% compared to the original AMReX compression. Moreover, AMRIC reduces the total writing time by up to 26.3% compared to the no-compression solution. This is consistent with Figure 15 in the main paper. The advantage of AMRIC in the expected I/O time is slightly lower than that reported in the main paper due to two factors: first, the use of fewer processes compared to the main paper, resulting in a lower I/O bottleneck and limiting AMRIC's improvement potential; second, the relatively low compression ratio caused by the smaller-sized AMR data (as mentioned in the previous paragraph), which requires AMRIC to spend more time writing compressed data.

For Step 10 (Nyx), it is clear that AMRIC (SZ-L/R) substantially reduces the total writing time by up to 73.0% compared to the original AMReX compression. Furthermore, AMRIC maintains writing speeds comparable to those with no compression. This is in line with Figure 16 in the main paper. It is also worth noting that, in the expected result, the I/O performance of the SZ-Interp method is relatively worse due to the considerable preprocessing time in the small-sized dataset used for artifact evaluation.

The expected results for Step 9 are:

```
---------- Writing Time for No Compression ----------
***** run 0 *****
No Compression Total time = 1.514 seconds
No Compression Preprocess time = 0.216 seconds
No Compression writing time = 1.322 seconds
...
------------------------ END ------------------------

------ Writing Time for original AMReX Compression ------
***** run 0 *****
original AMReX Total time = 4.734 seconds
original AMReX Preprocess time = 0.189 seconds
original AMReX Writing+Compression time = 4.493 seconds
...
------------------------ END ------------------------

---------- Writing Time for AMRIC-SZ_L/R ----------
***** run 0 *****
AMRIC-SZ_L/R Total time = 1.115 seconds
AMRIC-SZ_L/R Preprocess time = 0.223 seconds
AMRIC-SZ_L/R Writing+Compression time = 0.906 seconds
...
------------------------ END ------------------------

---------- Writing Time for AMRIC-SZInterp ----------
***** run 0 *****
AMRIC-SZ_Interp Total time = 1.878 seconds
AMRIC-SZ_Interp Preprocess time = 0.950 seconds
AMRIC-SZ_Interp Writing+Compression time = 0.937 seconds
...
------------------------ END ------------------------
```

The expected results for Step 10 are:

```
---------- Writing Time for No Compression ----------
***** run 0 *****
No Compression Total time = 0.195 seconds
No Compression Preprocess time = 0.016 seconds
No Compression writing time = 0.177 seconds
...
------------------------ END ------------------------

----- Writing Time for original AMReX Compression -----
***** run 0 *****
original AMReX Total time = 0.674 seconds
original AMReX Preprocess time = 0.020 seconds
original AMReX Writing+Compression time = 0.649 seconds
...
------------------------ END ------------------------

---------- Writing Time for AMRIC-SZ_L/R ----------
***** run 0 *****
AMRIC-SZ_L/R Total time = 0.182 seconds
AMRIC-SZ_L/R Preprocess time = 0.018 seconds
AMRIC-SZ_L/R Writing+Compression time = 0.155 seconds
...
------------------------ END ------------------------

---------- Writing Time for AMRIC-SZInterp ----------
***** run 0 *****
AMRIC-SZ_Interp Total time = 0.230 seconds
AMRIC-SZ_Interp Preprocess time = 0.102 seconds
AMRIC-SZ_Interp Writing+Compression time = 0.122 seconds
...
------------------------ END ------------------------
```