The Journal of Supercomputing (2025) 81:1631
https://doi.org/10.1007/s11227-025-08106-9

An efficient lossy compression framework for density partitioning in AMR applications

Yida Li[1] · Huizhang Luo[1] · Yufeng Zhang[1] · Keqin Li[1] · Kenli Li[1]

Received: 10 January 2025 / Accepted: 21 November 2025

Abstract
Adaptive mesh refinement (AMR) has become an indispensable tool in high-performance computing (HPC), enabling exascale simulations by dynamically allocating computational resources and significantly reducing memory footprint. Meanwhile, lossy compression is widely adopted in HPC environments to alleviate critical storage capacity and I/O bottlenecks, provided that reconstruction errors remain within acceptable bounds. However, the hierarchical structure, multi-resolution nature, and inherent spatio-temporal irregularity of AMR data pose unique challenges that render general-purpose compressors inefficient. Despite their respective advantages, existing data compressors still have an insufficient compression ratio and low throughput for data reduction in AMR applications. This paper mainly explores how to improve the performance of state-of-the-art lossy compression algorithms from the perspective of applications. To this end, we propose a density-partitioned AMR data lossy compression framework called AMRDPC, improving AMR applications' storage efficiency. The main ideas are twofold. First, to address the high computational overhead of using the k-d tree to process medium-density AMR data, we propose a fast k-d tree backfilling density grid (FBKDTree) strategy to improve compression speed. Second, to address the problem of the low compression ratio of high-density AMR data, we propose an efficient loop reversal patching (ELRP) strategy based on the design characteristics of existing prediction-based compressors. It can significantly improve the data compression performance while controlling errors. To verify the effectiveness of AMRDPC, we introduce multiple evaluation metrics for experimental analysis in seven real AMReX application datasets. Compared to state-of-the-art methods, AMRDPC achieves significant performance gains, with up to a 5.73× higher compression ratio and an 18.83% increase in throughput, providing a powerful data reduction solution for supercomputing environments.

Keywords Data compression · High-performance computing (HPC) · Adaptive mesh refinement (AMR) · Density partitioning · Data backfilling

Extended author information available on the last page of the article

Published online: 06 December 2025

1 Introduction

In recent years, as the scale of scientific simulations in high-performance computing (HPC) systems continues to grow, storage capacity and I/O bandwidth bottlenecks have become increasingly serious [1]. To mitigate these constraints, many HPC packages, such as AMReX [2], Athena [3], BoxLib [4], Chombo [5] [6], etc., have deployed adaptive mesh refinement (AMR) technology to reduce unnecessary computational overhead while ensuring the accuracy of the calculation results. Compared with the traditional uniform grid solution methods, AMR optimizes the use of computing resources and significantly lowers storage requirements, making it particularly suitable for large-scale scientific simulations. [7, 8].

While AMR technology successfully reduces output data volume, its efficacy can be insufficient for extreme-scale scientific simulations, and significant storage costs and I/O overheads persist [9]. For example, the WarpX project team studies the simulation of plasma accelerators, which reach the exascale computing level [10]. Because the simulation of plasma accelerators requires resolving the evolution of the driver (laser or particle beam) and the accelerating beam into structures that are several orders of magnitude longer than the accelerating beam. This will require several or even more orders of magnitude of acceleration based on the existing technology level.[1] Managing such large amounts of data is a preeminent supercomputing challenge. Saving all the generated raw data to disk is often impractical due to limited storage capacity and constrained I/O bandwidth.

A straightforward approach is to use data compression technology to fill this gap. As reported, scientific applications usually require more than 10× of data reduction [11]. However, data deduplication and lossless compression techniques are no longer effective due to their lower data compression performance [12]. Existing leading lossless compressors can only achieve a maximum data compression ratio of 2×, while lossy compressors can significantly reduce data size with controllable errors [13]. Specifically, the error bound can be set according to user requirements, such as absolute error bounds, and bounding values. The compressor ensures that the differences between the decompressed and original data do not exceed this error bound. Typical error-bounded lossy compressors include SZ [14–17], QoZ [18] and MGARD [19] based on prediction methods, ZFP [20] and TTHRESH [21] based on transformation methods, etc. Generally, the metrics for evaluating lossy compression performance include compression ratio, data distortion, and compression throughput. Among the state-of-the-art compressors, SZ usually has better compression performance [22].

Several prior works have explored the combination of AMR and lossy compression technology. For instance, zMesh considers reorganizing the data in different refinement levels of AMR into a 1D array for processing to make the data smoother, which improves the data compression ratio compared to directly input the data into the compressor for compression [23]. Based on this idea, LAMP proposes a

[1] https://www.exascaleproject.org/research-project/warpx/

hierarchical mapping method that can effectively reduce the redundancy between AMR hierarchical data and improve compression performance [24]. However, these methods process data in 1D space, which causes the loss of locality and topological information of high-dimensional spatial data. To solve this problem, the three-dimensional AMR Compressor (TAC) presents a method for compressing 3D AMR data, and divides the density of spatial data into three types: low, medium, and high density, and designs multiple strategies for data processing according to different densities [25]. AMRIC proposes an in-situ lossy compression framework and deploys HDF5 filters to improve compression performance for AMR applications [26].

Despite these advances, significant performance gaps remain. Specifically, TAC's k-d tree structure incurs substantial computational overhead during data backfilling for medium-density data, and its block patching strategy for high-density data fails to leverage the design of modern prediction-based compressors, yielding suboptimal compression. To this end, we propose AMRDPC, an efficient lossy compression framework based on density partitioning, explicitly designed for the challenges of AMR data in HPC environments.

Our main contributions are outlined as follows:

- We conducted an in-depth analysis of the shortcomings of existing AMR 3D compression technology, proposed an optimization design method for data processing strategies under different density partitions and constructed a lossy compression framework AMRDPC[2]
- To address the high computational overhead of AMR data processing at medium density, we design a fast k-d tree backfilling (FBKDTree) technique while ensuring that data fidelity remains unchanged compared to the original k-d tree backfilling strategy;
- To address the problem of poor compression performance in AMR data processing under high density, we designed an efficient loop reversal patching (ELRP) technology that can significantly improve data compression performance while keeping errors under control;
- We evaluate AMRDPC on the Tianhe supercomputer using seven real-world AMReX datasets, analyzing its compression ratio, distortion, and throughput. We also show that AMRDPC is a backend-agnostic framework, ensuring broad compatibility and flexibility.

Limitations of the proposed approarch. In this work, we focus on patch-based AMR data compression, which efficiently handles data redundancy across different levels of refinement. Unlike tree-based representations, the patch-based approach simplifies refinement computation and facilitates post-analysis and data visualization. Our method is designed for offline AMR data compression, rather than in-situ scenarios, and it is important to position it within a holistic data path.

[2] The source codes are available at https://github.com/liyida1995/AMRDPC;

It is important to note that compression in HPC systems operates at multiple, complementary layers, each addressing distinct challenges. Application-layer compression (such as the method proposed in this work) is primarily concerned with storage efficiency and data preservation. It leverages domain-specific knowledge of the data structure and semantics (e.g., AMR hierarchy, scientific error bounds) to achieve high compression ratios for long-term archiving and subsequent analysis. In contrast, network-layer compression [27–29] is designed for transmission efficiency, prioritizing low latency and high throughput to mitigate I/O and network bottlenecks during data movement. While generic and fast, these methods are typically data-agnostic and do not address the fundamental structural irregularity or rate-distortion trade-offs required for scientific data storage. These approaches are not mutually exclusive; rather, they can be synergistically combined in an end-to-end data pipeline. An optimal strategy may first employ application-specific compression to minimize the fundamental data volume for storage, upon which network-level compression can effectively operate to accelerate the transmission of this already-reduced data.

The remainder of this paper is organized as follows: Section 2 provides the background for this work. Section 3 further provides the motivation. Section 4 proposes the idea and implementation of AMRDPC. Section 5 shows the evaluation results and analysis of AMRDPC. Section 6 introduces related work in compression for scientific data and exploration for AMR applications, along with conclusions and future work in Sect. 7.

2 Background

In this section, we present AMR's principles and data layout, existing scientific data compressors, and related works in classical k-d trees for spatial data partitioning. We also discuss the state-of-the-art methods for AMR data compression and potential problems and challenges.

2.1 AMR principles and data distribution

A key method in HPC, AMR optimizes simulations by employing high-resolution grids in sub-regions with large physical gradients and low-resolution grids elsewhere. This targeted approach delivers high solution precision while maintaining high computational efficiency and managing resource demands.

Specifically, AMR calculates a finer grid in the following steps. **Step 1:** Calculate the local error of each grid point, usually using the Richardson extrapolation method [30]; **Step 2:** When the error is greater than a preset threshold, mark this grid point for further refinement. Find all grid points that need to be further refined and generate a new grid that can cover all grid points that meet the conditions; **Step 3:** Based on the coarse grid, use the interpolation method to fill the new grid. As such, there is a high correlation between the coarser grid and the finer grid; **Step 4:**

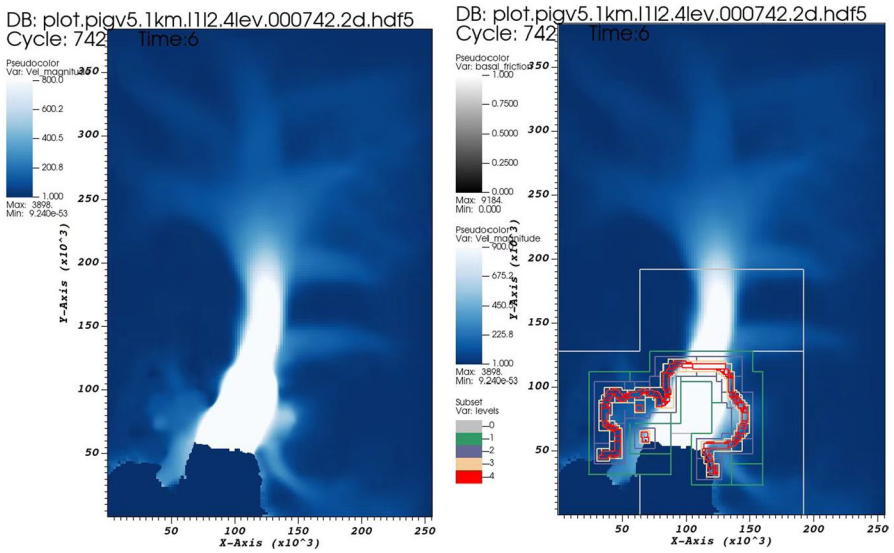
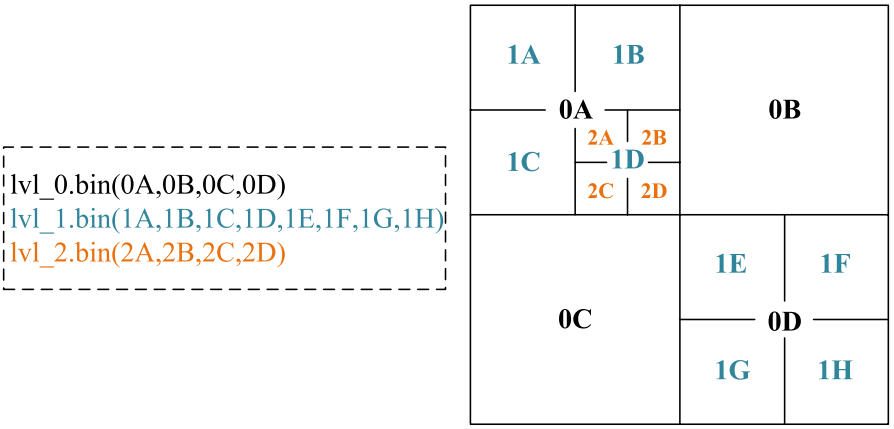


Fig. 1 Viewing of MISMIP3D with VisIt. a) Grounding line migration. b) Nested level boxes for adaptive resolution



(a) 1d baseline (b) AMR data layout

Fig. 2 A typical example of AMR data storage and layout

Repeat **Step 2**, with the error less than the specified threshold or reaching the maximum number of refinement levels as the termination condition.

Fig. 1 shows the visualization results of a real AMR application *MISMIP3D* with VisIt,[3] which is the Marine Ice Sheet Model Intercomparison Project for plan view

[3] https://hpc.llnl.gov/software/visualization-software/visit

models. The progressively small boxes indicate regions of progressively higher resolution. AMR uses finer meshes such that high resolution is maintained at the grounding line. The data from a finer level are highly correlated with those from a coarser level.

The data of each AMR level are usually stored separately (e.g., in a 1D array). When the AMR data are needed for post-analysis or visualization, users will typically convert the data from different levels to a uniform resolution. For example, Fig. 2(a) shows a simple example of three-levels AMR data; 0 means low resolution (the coarse level), 1 and 2 mean high resolution (the fine level). Fig. 2(b) illustrates a dataset of three AMR levels, with one, two, and one box at each level. Each box has four data points, where each point is denoted by the concatenation of its level ID and sequence index within its level. For example, 1A denotes the first data point at level 1.

2.2 Compressors for scientific data compression

Scientific data compression is divided into two categories: lossless compression and lossy compression. Compared with lossless compression, lossy compression can greatly increase the data compression ratio while ensuring controllable errors [31].

In the HPC domain, with the explosive growth of scientific data scale, many high-precision lossy compressors have been designed and developed well [32], for example, SZ and ZFP. SZ is a lossy compressor based on a prediction model, which uses prediction methods such as Lorenzo prediction to remove data correlation and uses quantization methods combined with lossless compression techniques such as Huffman encoding to further compress data [15, 17]. ZFP is a transform-based compressor, which uses orthogonal block transformation to remove the correlation between data and uses embedded encoding to compress transformation coefficients [20]. Within the same error bound, SZ is usually 2× higher than ZFP in compression ratio, but at the same time, SZ has a 20% to 30% performance trade-off in encoding and decoding throughput compared to ZFP in terms of time overhead [31, 33].

This paper focuses on SZ lossy compression because SZ has a high compression performance [22]. Specifically, SZ has four main steps. In **the first step**, based on neighbor data points, a variety of prediction methods (such as Lorenzo prediction and spline interpolation prediction) can be used to predict the current data point; in **the second step**, based on the preset error bound, calculate the difference between the predicted value and the actual value, and quantize the difference; **the third step**, use Huffman coding to encode the quantized value; **the fourth step**, use lossless compression technology (such as Zstd) to further compress the data.

2.3 K-d tree for multi-dimensional spatial data partitioning

The k-d tree is a fundamental data structure for partitioning k-dimensional space, widely used in particle data compression to identify particles and eliminate empty regions [34]. As a binary search tree, it recursively subdivides the space along

alternating axes (e.g., $x \rightarrow y \rightarrow x$) until each sub-region contains either particles or becomes empty, as illustrated in Fig. 3.

However, this axis-aligned partitioning strategy often proves inefficient [35]. For instance, within the red dashed box in Fig. 3, following the conventional order produces two 2×2 sub-regions, both still containing empty areas that require further subdivision. In contrast, partitioning along the x-axis yields one 1×4 sub-region free of empty space, thereby reducing subsequent partitioning effort. This example highlights the limitation of the classical k-d tree approach when handling non-uniform data distributions.

In AMR data compression, the spatial partitioning strategy of traditional k-d trees demonstrates limited efficiency when handling the complex hierarchical density distribution characteristic of AMR data. Notably, the "empty blocks" commonly found in AMR data are not manually specified using external domain knowledge, but represent inherent structural features where all data values equal zero. These empty blocks originate from the fundamental AMR refinement mechanism: computational resources focus on critical regions, while other areas remain at coarse resolution and are populated with zero values. Our proposed method formally defines empty blocks as data blocks whose maximum absolute value is zero. By specifically targeting these structural empty blocks, our enhanced k-d tree strategy achieves efficient adaptive spatial partitioning that is better suited to the characteristics of AMR data. The detailed algorithm design can be found in Sect. 4.

3 Motivation

This section identifies shortcomings in TAC's k-d tree and ghost-shell padding (GSP) strategies for AMR data preprocessing, which result in poor compression ratios and significant time overhead. To address these limitations, we propose a new data processing strategy, detailed after the following analysis.

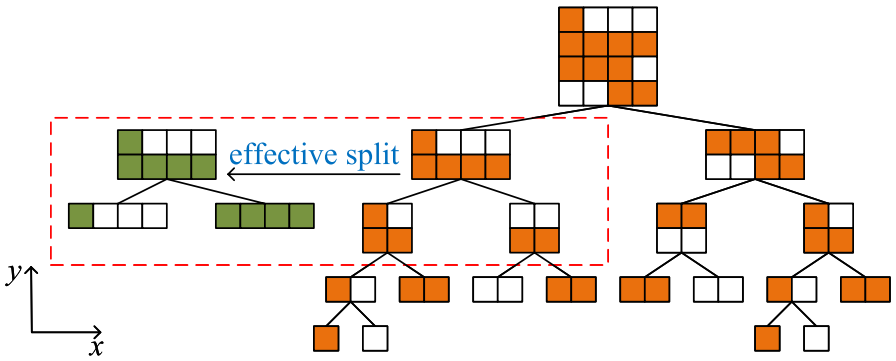


Fig. 3 A classical k-d tree for 2D data partitioning

Observation 1: When compressing medium-density AMR data, the k-d tree construction is the critical performance bottleneck of the algorithm, which prevents TAC from providing high compression throughput for 3D AMR data.

First, we conducted an empirical analysis of the time costs associated with the OpST strategy and k-d tree strategy used by TAC when compressing low-density and medium-density AMR data. Our analysis, as depicted in Fig. 4, reveals the time cost of these two strategies for processing the three datasets Grid_Z2, Grid_Z3, and Grid_Z5, under the same absolute error.

The results show that k-d tree construction accounts for about 90% of the total AMR data compression time. More deeply, we observed that when mapping the k-d tree structure data back to the 3D density grid, multiple nested loops are introduced to traverse all tree blocks and leaf nodes. Specifically, assuming that the number of tree blocks is m and the average number of leaf nodes under each tree block is n, the time complexity of this part is $O(m * n)$. The number of loops involved in the filling operation corresponding to each leaf node depends on the size of the leaf node. Assuming that in the worst case, the size of each leaf node is d, then the time complexity of the filling operation is $O(d^3)$. Since this operation is performed once for each leaf node, the total time complexity is $O(m * n * d^3)$. To overcome this limitation, we can consider appropriately adjusting the construction strategy of the k-d tree. For smaller leaf nodes, we can consider a more efficient backfilling algorithm to reduce the number of loops and thereby reduce time overhead. Please see Sect. 4 for the specific design.

Observation 2: When processing high-density AMR data, the original GSP strategy overlooks its spatial distribution and integrates poorly with block-wise SZ compressors, potentially compromising compression performance.

For block-wise SZ compressors like SZ2, data prediction is pivotal to performance. High prediction accuracy causes the resulting errors to be clustered closely around zero, which enables efficient Huffman encoding and higher compression ratios. This is why SZ2 performs best on data with high spatial locality; low locality directly compromises prediction quality and, in turn, the compression ratio.

As shown in Fig. 5, we use two examples to illustrate the irrationality of the original GSP strategy. This level of high-density AMR data is divided into 5×5, i.e., 25 blocks, where the blank color blocks represent empty areas, and the colored parts

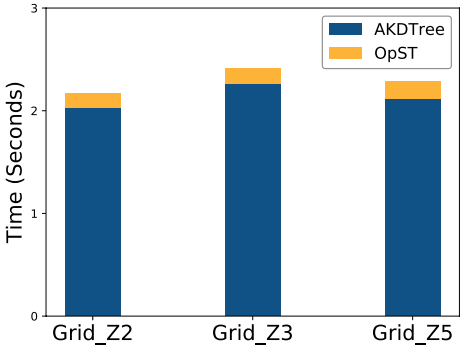


Fig. 4 Time overhead of the algorithm when processing three low–medium density AMR datasets under the same absolute error

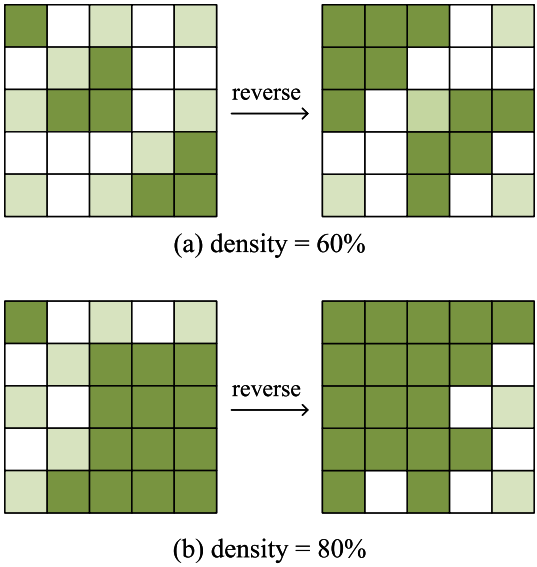


Fig. 5 Analyze the shortcomings of the GSP strategy when processing high-density AMR data

represent the data areas. The left part of Fig. 5(a) and Fig. 5(b) shows the original AMR data distribution. For example, when the density is 60% (15/25), the original GSP method will patch the empty areas from the top-left to the bottom-right. At this time, the dark green blocks represent blocks that can be effectively predicted using SZ, while the light green blocks represent blocks that SZ cannot accurately predict. In this case, the number of blocks that can be accurately predicted is 7, and the number of blocks that are difficult to accurately predict is 8. When we reverse this level, we get the situation shown on the right side of Fig. 5(a). At this time, the original blocks that cannot be accurately predicted have changed their prediction accuracy because the blocks in the upper left can be accurately predicted. Through reversion, the number of blocks that can be accurately predicted becomes 11, and the number of blocks that are difficult to accurately predict becomes 4. Similarly, as shown in Fig. 5(b), through reversion, the number of blocks that can be accurately predicted increases from 14 to 18, while the number of blocks that cannot be accurately predicted decreases from 6 to 2. In other words, improving the data prediction hit rate will bring benefits to the patch operation in empty areas. Therefore, combined with prediction-based compressor design principles, the loop inversion strategy can be considered to optimize the original patching strategy to further improve data compression performance. In this work, we use SZ2 to compress AMR data. For detailed design, please refer to Sect. 4.

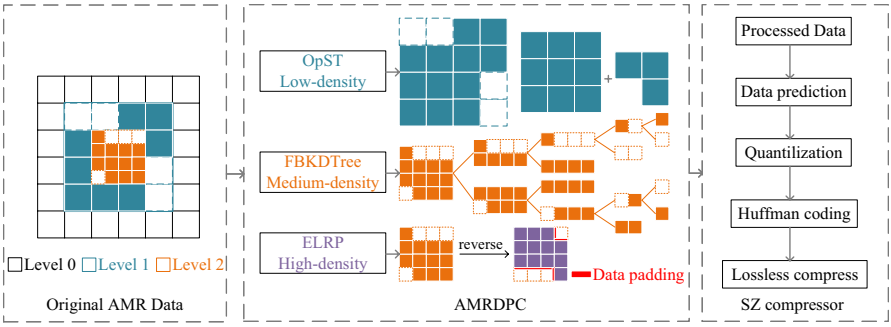


Fig. 6 Overview of our proposed AMRDPC

Table 1 Strategy comparison between TAC and AMRDPC

Bottleneck	TAC	AMRDPC	Gain
\	OpST	OpST (adopted)	\
Slow Backfilling	AKDTree	**FBKDTree**	**+18.83% CT**
Low CR	GSP	**ELRP**	+5.73× **CR**

4 Design methodology

We propose AMRDPC, an efficient framework for 3D AMR data compression illustrated in Fig. 6. It employs tailored strategies for different data densities: FBKDTree for medium-density data and ELRP for high-density data. Section 4.1 presents the overall architecture, while Sect. 4.2 and 4.3 detail the FBKDTree and ELRP strategies, respectively.

4.1 Framework

AMRDPC employs distinct processing strategies tailored to different density patterns in AMR data. The framework processes original AMR data containing multiple adjacent levels, where Level 0 represents the coarse level, Levels 1 and 2 represent refined levels, and dotted boxes indicate empty regions. The core distinctions are systematically compared in Table 1.

As outlined in Table 1, the novelty of AMRDPC resides in two primary contributions that directly address the bottlenecks of TAC:

1) For medium-density data, our FBKDTree replaces TAC's complex hierarchical AKDTree with a flattened partitioning logic. This simplification drastically reduces computational overhead, accelerates compression, and significantly improves CT.
2) To address the suboptimal compression of TAC's GSP on high-density data, we propose the ELRP strategy. This coherence-aware method employs a reverse

analysis to intelligently patch empty blocks from data trends, achieving a higher CR under strict error bounds.

To facilitate detailed analysis of our proposed FBKDTree and ELRP algorithms, Table 2 summarizes the key variables used throughout the algorithmic implementations. These variables play crucial roles in spatial partitioning, indexing, and density computation processes.

Building upon these fundamental variables, we now provide an in-depth analysis of the FBKDTree and ELRP algorithms.

4.2 FBKDtree strategy for medium-density AMR data

We now focus on analyzing the advantages of FBKDTree compared to conventional k-d tree data space partitioning strategies. Compared to conventional k-d tree data space partitioning strategies, our proposed FBKDTree method offers the advantage of adaptive data space partitioning. Specifically, the adaptive k-d tree partitioning involves the following steps:

First, the 3D dataset is divided into multiple sub-data blocks. Second, a tree structure represents the position of each data value within the hierarchical dataset. Each tree node is associated with a sub-data block in the dataset and stores the number of non-empty sub-data blocks within the sub-data block linked to that node. Third, for each tree node, the sub-data block is split into two smaller sub-data blocks along a certain direction. The selection of the splitting direction follows the principle of maximizing the distinction between non-empty sub-data blocks and blank areas. Fourth, the subdivision of sub-data blocks continues iteratively until the sub-data blocks consist entirely of non-empty data regions or entirely blank areas. Fifth, after completing the subdivision of sub-data blocks, i.e., constructing a complete k-d tree,

Table 2 Variable summary

Variable Name	Description
densgrid	Density grid data array
grid/*grid_x*, *y*, *z*	Grid dimensions
blkSize	Block size for processing units
leafCnt	Leaf node count per depth level
maxTreeBlk	Maximum tree blocks or depth level
strideX, *Y*, *Z*	Strides for 3D grid indexing
xBegin, *xEnd*, *yBegin*, *yEnd*, *zBegin*, *zEnd*	Node spatial indices
xSize, *ySize*, *zSize*	Node spatial sizes
blkX, *Y*, *Z*	Block counts in spatial dimensions
blkSize	The length of the data block
avgDensity	Calculated average density
avgCount	Sample count for density calculation
hasNeighbor	Flag for valid neighbors

the non-empty sub-data blocks contained in the leaf nodes of the k-d tree are fed into the compressor for data compression. It is worth noting that non-empty leaf nodes do not contain any empty areas; otherwise, the node would undergo further partitioning. Therefore, a leaf node can only be a data region containing non-empty sub-data blocks or an empty block.

Unlike traditional k-d tree partitioning rules, the splitting direction in the FBKDTree algorithm is dynamically determined. As shown in Fig. 7, a data block is partitioned along the x, y, and z axes, resulting in eight smaller sub-data blocks. The number of non-empty sub-data blocks within these eight sub-data blocks is counted and denoted as n_1, n_2, ..., n_8. Based on the counts of non-empty sub-data blocks along the x, y, and z axes, the direction for data partitioning is determined. The calculation formula is as follows:

$$\begin{aligned} div_x &= |n_1 + n_4 + n_5 + n_8 - n_2 - n_3 - n_6 - n_7| \\ div_y &= |n_1 + n_2 + n_5 + n_6 - n_3 - n_4 - n_7 - n_8| \\ div_z &= |n_1 + n_2 + n_3 + n_4 - n_5 - n_6 - n_7 - n_8| \end{aligned} \tag{1}$$

The direction corresponding to the highest value in the *div* set determines the next splitting direction for the current data block. As illustrated in Fig. 7, assuming div_z has the largest value, the data block is split into upper and lower sub-blocks along the z-axis. Subsequently, only the values of div_x and div_y need to be calculated, with the larger value determining the subsequent splitting direction. This adaptive partitioning process continues until it has no empty sub-block or itself is empty.

In addition to its adaptive capability in spatial data partitioning, the FBKDTree algorithm can rapidly backfill data from the k-d tree structure into the *densgrid*. Its main steps include traversing all depth levels *treeDepth* of the tree and performing operations such as calculating node boundaries, calculating node sizes, and adjusting index steps for all leaf nodes *leafCnt*[*treeDepth*] at each level. By traversing all elements in the node range, the values are read from the *tree*[*treeDepth*][*count*[*treeDepth*]] and filled into the *densgrid*. Please refer to Algorithm 1 for implementation details.

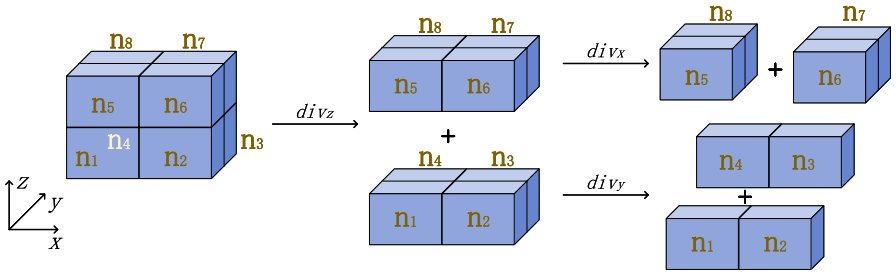


Fig. 7 3D Example of the adaptive FBKDTree, demonstrating recursive splitting that terminates when a node becomes either a full data block or empty

Algorithm 1 FBKDTree algorithm.

Input: k-d tree information
Output: densgrid data

```
memset(count, 0, treeDepth);
strideZ ← grid * grid;
strideY ← grid;
strideX ← 1;
for each i ∈ [0, treeDepth − 1] do
    for each j ∈ [0, leafCnt[i] − 1] do
        get the block b information xBegin, ..., zEnd;
        calculate block size bSize for this node;
        offsetX ← strideX * bSize;
        offsetY ← strideY * bSize;
        offsetZ ← strideZ * bSize;
        for each data(x, y, z) ∈ b do
            id ← x * offsetX + y * offsetY + z * offsetZ;
            densgrid[id] ← tree[i][count[i]];
            count[i] + +;
        end for
    end for
end for
    return reconstructed data
```

Compared with the original design, FBKDTree has the following advantages: 1) Reduce array access: By precomputing the stride (*strideX*, *strideY*, *strideZ*), reduce the number of accesses to the dense grid array and improve data access efficiency; 2) The calculation of the step size is based on the *grid* size and *block* size, ensuring the consistency and accuracy of the index calculation; 3) The original nested loop is relatively complex, but our method can avoid unnecessary complex calculations and improve execution efficiency through efficient index calculation of the inner loop.

Complexity analysis: The FBKDTree algorithm implements the backfilling operation of k-d tree structure data to the AMR grid, and its time complexity can be decomposed into key steps such as tree depth traversal, leaf node traversal, and data space traversal. First, the time complexity of the outermost tree depth traversal process is $O(maxTreeBlk)$, where *maxTreeBlk* represents the maximum depth of the tree. Second, the time complexity of the leaf node traversal process is $O(leafCnt[i])$, with $leafCnt[i]$ denoting the number of leaf nodes at depth i. Furthermore, to analyze the time complexity of the data space traversal process, assume that the size of each data block at depth i is defined as follows: $xSize = xEnd - xBegin + 1$, $ySize = yEnd - yBegin + 1$, and $zSize = zEnd - zBegin + 1$. In this case, the time complexity of this step is expressed as $O(xSize \times ySize \times zSize)$. If the total number of leaf nodes is S and the total number of grid points is $N = grid^3$, the time complexity in the worst-case scenario is $O(N \times maxTreeBlk)$. In an ideal scenario, each grid point only needs to be accessed once, resulting in a time complexity of $O(N)$. In addition, the space complexity of the FBKDTree algorithm is mainly determined

by the input data scale and the tree structure. The overall space complexity is $O(N)$, classifying it as an efficient and scalable algorithm.

4.3 ELRP strategy for high-density AMR data

The algorithm of ELRP includes calculating the number and offset of grid blocks (*blkX*, *blkY*, *blkZ*), traversing all blocks from bottom right to top left, and calculating the average density *avgDensity* of adjacent blocks of each block. If there are adjacent blocks that are non-empty, update the density value of the current block, using the *avgDensity* as the patch value of the current empty area. Please refer to Algorithm 2 for implementation details.

Algorithm 2 ELRP algorithm.

Input: AMR data
Output: Data after patching
get the number of blocks, $blkX, blkY, blkZ$;
for $z = blkZ$ *downto* 1 **do**
  for $y = blkY$ *downto* 1 **do**
    for $x = blkX$ *downto* 1 **do**
      if (block $b(x, y, z)$ is empty) **then**
        $avgDensity \leftarrow 0.0$;
        $Count \leftarrow 0$;
        $hasNeighbor \leftarrow false$;
        if b has a non-empty neighbor in XYZ directions **then**
          $hasNeighbor \leftarrow true$;
          count $Count$;
          $avgDensity = \Sigma\{b'sNeighbors\}/Count$;
        end if
      else
        $continue$;
      end if
    end for
  end for
end for
  return patched data

Compared with the original design, ELRP has the following advantages: 1) Using a reverse loop of grid blocks improves the hit rate of data prediction, thereby improving the accuracy of data patching for the empty areas; 2) For each block, calculate the *avgDensity* of its surrounding neighbor blocks, only consider non-empty neighbors, and skip the central block itself when calculating the *avgDensity*; 3) If there are non-empty neighbors, update the density values of all grid points in the current block. If the grid points have not been processed yet, update the density values directly; otherwise, update based on the weighted average.

Complexity analysis: The computation time of the ELRP algorithm mainly consists of three components: adjacent data checking, data density calculation, and data block update operations. Specifically, for the process of checking adjacent data within a data block in three-dimensional space, its time complexity is $O(3^3) = O(1)$, which is a constant time. When calculating data density, the running time of this

process is affected by the length of the data block, and the time complexity is expressed as $O(blkSize \times blkSize \times C)$, where C is a constant representing the data offset in a specified direction within the data block ($C \leq blkSize$). In the data block update operation, its time complexity is affected by the size of the data block, which is $O(blkSize^3) = O(n^3)$. When it is necessary to traverse all data blocks contained in the space, the time complexity of the ELRP algorithm is expressed as $T(n) = O(N \times \frac{C}{blkSize} + 1)$, where N represents the number of grids. If the offset C is much smaller than $blkSize$, $T(n) = O(N)$; if the offset C is approximately equal to $blkSize$, $T(n) = O(2N)$, which is a linear complexity. In addition, the space complexity of the ELRP algorithm is composed of two parts: input data and auxiliary data structures. Therefore, the space complexity of the ELRP algorithm is $O(N)$, making it an efficient and scalable algorithm.

5 Evaluation

In this section, we first illustrate the experimental setup and evaluation metrics and then analyze the experimental results. We consider multiple indicators for experimental evaluation, including data compression ratio, multiple quality metrics for data distortion, compression speed, etc. The experiments are conducted based on multiple real-world scientific datasets from AMReX, which are described as follows.

5.1 Experimental setup

1) Execution environment: We conduct our experiments on a Linux server with OS kernel of Linux 5.15, CPU of 12th Gen Intel(R) Core(TM) i9–12900K, main memory of 32 GB DDR5 RAM, and storage device of 1TB M.2 Gen4 NVMe SSD.

2) Applications: Our experimental evaluation focuses on the AMReX framework and in particular the Nyx cosmological simulations. Nyx is a state-of-the-art extreme-scale cosmology code that uses AMReX to generate multiple fields including baryon density, dark matter density, temperature, and velocity.[4] Specifically, we employ a 64-Mpc region, using seven datasets derived from two real-world simulation runs with different numbers of AMR levels.

Table 3 outlines the key information of the test datasets, including the number of AMR levels, and presents the following details ranging from coarse to fine: the grid size of each level, the number of data blocks, the data density distribution across levels, as well as the size of each dataset. Here, AMR levels are denoted as Level 0, Level 1, Level 2, ..., in the order from Coarse to Fine.

Furthermore, the first simulation run yielded datasets such as Grid_Z2, Grid_Z3, Grid_Z5, and Grid_Z10. The AMR refinement level was set to 2, which means there are both a coarser level (Level 0) and a finer level (Level 1). The grid sizes for Level 0 and Level 1 are 512 and 256, respectively. It is important to note that the number

[4] https://amrex-astro.github.io/Nyx/

Table 3 Our tested datasets

Dataset	Levels	Grid Size	Block Count	Density	Total Size
Grid_Z2	2	512/256	20,494/12,274	62.54%/37.46%	4.3 GB
Grid_Z3	2	512/256	20,934/11,834	63.89%/36.11%	4.4 GB
Grid_Z5	2	512/256	19,197/13,571	58.58%/41.42%	4.1 GB
Grid_Z10	2	512/256	7,597/25,171	23.18%/76.82%	2.1 GB
Run2_T2	2	256/128	8/4,088	0.20%/99.80%	102.0 MB
Run2_T3	3	512/256/128	8/184/32,576	0.02%/0.57%/99.41%	106.2 MB
Run2_T4	4	1024/512/256/128	8/56/5,888/256,192	0.08%/0.02%/2.17%/97.73%	119.4 MB

of data blocks contained in the grid refinement varies across different levels, resulting in an uneven distribution of data density. Among them, the density of the finest level describes the proportion of data in the dataset that reaches the highest resolution; in other words, a higher density of the finest level indicates that a larger amount of data is refined to the highest resolution.

For the convenience of discussion, we classify levels based on AMR data density:

- A level is categorized as a low-density level if its density is below 50%.
- It is classified as a medium-density level if the density ranges between 50% and 70%.
- A high-density level refers to one with a density exceeding 70%.

This classification allows us to demonstrate the effectiveness of our proposed method for different density distribution characteristics. For instance, in the Grid_Z2, Grid_Z3, and Grid_Z5 datasets, Level 0 contains more data blocks, which implies that a higher solution accuracy is set for this refined level, leading to a higher data density. In the Grid_Z10 dataset, however, Level 1 contains more data blocks (with a data density as high as 76.82%), so Level 1 is defined as the level with high-density AMR data in this case.

The second simulation run generated datasets including Run2_T2, Run2_T3, and Run2_T4 by setting different AMR refinement levels (2, 3, and 4). The refined grid scale was expanded from 128 to a maximum of 1024. For example, Run2_T4 was configured with an AMR refinement level of 4; the number of data blocks at Level 3 reached 256,192, which is much higher than that of other levels, and its data density reached 97.73%, thus being defined as the level with high-density AMR data. Similarly, the data densities of Level 1 in Run2_T2 and Level 2 in Run2_T3 are as high as 99.80% and 99.41%, respectively, both of which fall into the category of high-density levels.

3) Comparison baseline: We adopt two 3D comparison baselines. Specifically, (1) the primitive 3D baseline: Different AMR levels are unified to the same resolution for 3D compression; (2) the state-of-the-art 3D compressor TAC, which involves multiple density-based strategies to process AMR data.

5.2 Evaluation metrics

We perform the evaluation based on four critical metrics:

(1) Compression ratio (*CR*): We use compression ratio to evaluate the reduction in data size. The compression ratio is defined as the reduction ratio of the original AMR data size to the compressed data size ($CR = \frac{original\ size}{compressed\ size}$). Bit rate (bits/value) represents the amortized storage cost of each value. For example, the bit rate of single/double-precision floating-point data before compression is 32/64 bits per value. For single/double-precision floating-point data, the product of the compression ratio and the bit rate is 32/64. Therefore, the compression ratio will be higher when the bit rate is lower.

(2) Rate-PSNR plots: Peak signal-to-noise ratio (PSNR) is an important metric in lossy compression ratio distortion evaluation.

$$PSNR = 20\log_{10}\frac{vrange(D_1)}{\sqrt{mse(D_1, D_2)}} \tag{2}$$

where D_1 represents the data before compression, D_2 represents the data after decompression, and *vrange* represents the numerical range of the data before compression.

(3) Rate-SSIM plots: Moreover, to justify that the similarities are high and common between AMR levels, we introduce one quality assessment metric named Structural Similarity Index Measure (SSIM). The SSIM is a method for measuring the similarity between two images, which has been widely used in the community of HPC scientific data compression [18]. The higher the SSIM value, the more similar. The formula for calculating SSIM with original data D_1 and decompressed data D_2 is:

$$SSIM(D_1, D_2) = \frac{(2\mu_{D_1}\mu_{D_2} + C_1)(2cov_{D_1D_2} + C_2)}{(\mu_{D_1}^2 + \mu_{D_2}^2 + C_1)(\sigma_{D_1}^2 + \sigma_{D_2}^2 + C_2)} \tag{3}$$

where μ is the mean, σ^2 is the variance, C is constant and the *cov* represents the covariance between two data.

(4) Compression speed: We check the overall compression throughput of our AMRDPC framework to show the low computational overhead in our solution. Compression throughput is defined as $CT = \frac{original\ size}{compression\ time}$ (MB/s).

5.3 Compression ratio

As shown in table 4, we compared the compression performance of AMRDPC, 3D baseline, and TAC for high-density AMR data under different absolute error bounds δ, and introduced the Normalized Root Mean Square Error (NRMSE) to evaluate the deviation between decompressed data and real data. NRMSE shows the statistical difference between the original values and the values decompressed. Furthermore,

Table 4 Information loss metrics on four high-density AMR datasets

Method	δ	Grid_Z10			Run2_T2			Run2_T3			Run2_T4		
		N	CR	CT	N	CR	CT	N	CR	CT	N	CR	CT
3D-Base	1E+9	3.3E-5	37.13	157.09	2.1E-3	12.37	56.22	4.1E-5	2.39	8.22	1.0E-5	0.72	1.18
	2E+9	4.5E-5	74.97	159.77	2.8E-4	28.84	58.71	5.5E-5	4.78	8.46	1.5E-5	1.10	1.19
	3E+9	5.1E-5	106.08	163.57	3.2E-4	43.96	60.31	6.4E-5	6.89	8.62	1.8E-5	1.41	1.21
	4E+9	5.6E-5	130.08	167.53	3.4E-4	55.61	61.31	7.0E-5	8.57	8.74	2.1E-5	1.67	1.22
	5E+9	6.0E-5	149.62	169.56	3.6E-4	64.72	61.97	7.4E-5	10.01	8.90	2.3E-5	1.92	1.23
TAC	1E+9	1.2E-2	48.02	260.13	4.2E-3	41.53	242.83	2.0E-3	37.22	196.66	1.9E-3	31.10	168.17
	2E+9	1.9E-2	88.39	290.92	7.0E-3	78.36	262.79	3.3E-3	61.36	208.99	3.3E-3	45.50	176.73
	3E+9	2.2E-2	129.97	297.82	8.4E-3	132.09	276.41	4.2E-3	94.03	218.53	4.2E-3	59.99	186.21
	4E+9	2.5E-2	176.98	304.99	9.3E-3	185.90	287.86	4.8E-3	123.85	227.47	5.0E-3	73.61	191.34
	5E+9	2.6E-2	226.01	310.16	10E-3	240.11	293.02	5.2E-3	152.60	231.74	5.6E-3	85.50	196.22
AMRDPC	1E+9	8.9E-3	65.41	255.82	4.0E-3	237.90	246.44	2.1E-3	119.97	190.38	2.3E-3	48.20	141.11
	2E+9	1.5E-2	110.92	282.91	6.0E-3	379.36	251.14	6.4E-3	212.55	190.99	6.5E-3	74.07	145.07
	3E+9	2.0E-2	152.89	284.53	6.9E-3	508.52	251.95	7.0E-3	267.39	196.15	1.1E-2	96.17	148.91
	4E+9	2.3E-2	204.88	290.11	7.5E-3	612.47	254.42	7.4E-3	299.27	197.38	1.3E-2	111.60	149.52
	5E+9	2.3E-2	258.27	292.21	8.0E-3	698.74	256.20	7.8E-3	336.94	194.35	1.4E-2	127.61	151.66

N: NRMSE (smaller better), CR: Compression ratio, CT: Compression throughput

we calculated the compression throughput to evaluate the computational overhead of the algorithms.

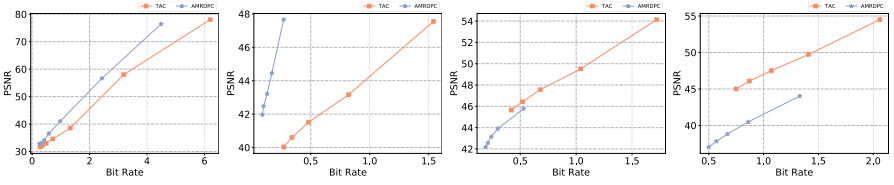
When the absolute errors are the same, both AMRDPC and TAC achieve significant improvements in CR and CT over the 3D baseline. Across the four tested datasets, AMRDPC demonstrates a consistent and substantial advantage in CR, which is its primary design objective. For instance, on the Run2_T2 dataset at $\delta = 1E + 9$, AMRDPC achieves a CR of 237.90, which is 5.73× higher than that of TAC (41.53), while also reducing the NRMSE. Similar CR improvements are observed in the Grid_Z10, Run2_T3, and Run2_T4 tests.

It is noteworthy that while AMRDPC's CT is highly competitive and often superior, we observed a slight throughput trade-off on the Run2_T2 dataset (e.g., 246.44 vs. 242.83 at $\delta = 1E + 9$). This is attributed to the highly irregular and fine-grained mesh structure of Run2_T2, which maximizes the CR gain through our patching and encoding techniques but introduces manageable computational overhead. This illustrates a well-justified trade-off, as the significant gain in compression efficiency far outweighs the minor cost in processing speed for high-density AMR data. To further explore the effectiveness of AMRDPC, we analyzed the rate distortion of the data using PSNR and SSIM.

5.4 Post-analysis quality

In the following text, we present the overall rate-distortion results of AMRDPC versus another comparison baseline, in regard to different quality metrics.

As shown in Fig. 8dcba, we used PSNR to perform rate distortion tests on four high-density AMR datasets. The results on the Grid_Z10 and Run2_T2 datasets show that while AMRDPC does not achieve the highest PSNR at all bitrates, it delivers a highly competitive rate-distortion performance characterized by an exceptional compression ratio. Consequently, for AMR applications where the primary objective is to maximize data reduction, AMRDPC's design strategically prioritizes overall storage efficiency over the pursuit of uniformly peak fidelity. Specifically, AMRDPC achieves 14.27% to 36.21% improvement in compression ratio on Grid_Z10 dataset than TAC when PSNR is around 30–80, and 5.73× improvement in compression ratio on Run2_T2 dataset when PSNR is 47.24.



(a) Grid_Z10 (Level 1) (b) Run2_T2 (Level 1) (c) Run2_T3 (Level 2) (d) Run2_T4 (Level 3)

Fig. 8 Rate distortion evaluation (PSNR)

On the Run2_T3 and Run2_T4 data sets, it can be observed that at the same bit rate, the PSNR curve of TAC is generally above that of AMRDPC. This indicates that the data fidelity of AMRDPC is lower than that of TAC at these data levels.

From the mechanism of AMR refined grids, it starts from the entire data domain and determines whether to refine a data block into smaller sub-blocks based on the set error bounds. Data at higher refinement levels (Level 2, Level 3) may contain more key features with strong locality, small scale, but large numerical variation. In such cases, AMR may incorrectly identify these regions as smooth areas. For compressors based on prediction models, when some sub-data blocks face the challenge of over-aggregation, the processing mechanism of ELRP will cause these fine but important physical features to be smoothed out or directly lost during the compression process, thereby reducing the overall fidelity of the data.

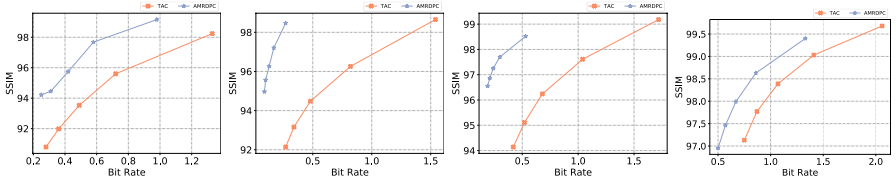
This phenomenon also reveals that when the AMRDPC method processes more complex high-density AMR level data, there may be costs associated with data over-aggregation. However, in terms of the overall improvement in compression performance, compared with TAC, AMRDPC achieves a compression ratio increase of 1.21× to 2.46× and 49.25% to 62.79%, respectively.

We further explore the rate distortion of the evaluation data using SSIM. Fig. 9 shows the rate-SSIM plots of different methods, AMRDPC uses less bit rate while maintaining better fidelity of decompressed data than TAC. In the plots, the x-axis is bit rate and the y-axis is SSIM. Although the SSIM values are relatively low on the Run2_T3 and Run2_T4 datasets, this is negligible in terms of the overall compression performance improvement.

For accurate rate-distortion comparison, the 3D baseline was excluded as it employs a uniform compression method that does not account for the complex density variations across AMR levels. Consequently, to evaluate the fidelity of AMRDPC on high-density data, our analysis specifically targets those AMR levels that meet the high-density criterion.

5.5 Comprehensive compression overhead analysis

AMRDPC achieves substantial performance gains in medium-density AMR data compression, driven by its FBKDTree strategy which minimizes pre-compression overhead without compromising compression performance.



(a) Grid_Z10 (Level 1) (b) Run2_T2 (Level 1) (c) Run2_T3 (Level 2) (d) Run2_T4 (Level 3)

Fig. 9 Rate distortion evaluation (SSIM)

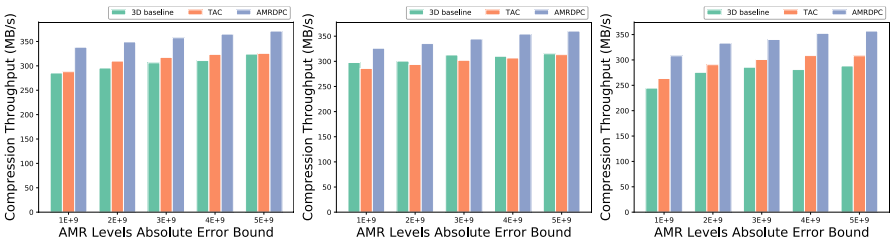
Table 5 quantifies the full pipeline overhead for all methods. The results confirm the superior efficiency of AMRDPC, particularly in pre-compression, where it outperforms the 3D baseline by a wide margin.

As shown in Fig. 10cba, AMRDPC improves end-to-end throughput by up to 18.83% (Grid_Z2), 17.42%, and 14.48% across the datasets. Critically, these gains originate entirely from the pre-compression phase, with compression and decompression overheads remaining consistent with baseline methods.

As shown in Fig. 11 and Fig. 12cba, AMRDPC improves compression speed while maintaining data fidelity. We further analyzed whether the data fidelity changes under different absolute error bounds. We do not introduce a 3D baseline

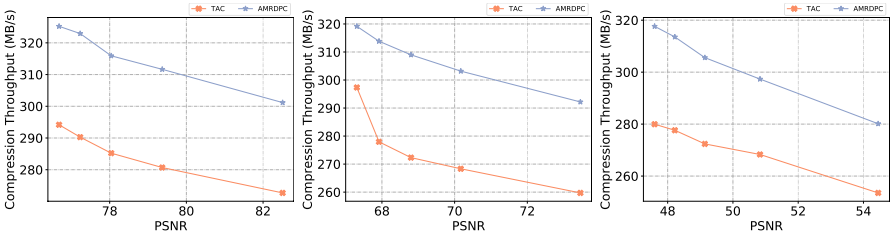
Table 5 Compression overhead comparison (Time: seconds)

Dataset	Method	Pre-compression	Compression	Decompression
Grid_Z2	AMRDPC	0.0466	1.2823	0.5840
	TAC	0.0670	1.2905	0.5854
	3D baseline	9.9825	1.8028	2.9806
Grid_Z3	AMRDPC	0.0462	1.3241	0.5989
	TAC	0.0668	1.3273	0.6003
	3D baseline	10.0191	1.8134	0.7885
Grid_Z5	AMRDPC	0.0456	1.2496	0.5595
	TAC	0.0677	1.2611	0.5596
	3D baseline	9.4321	1.8479	0.8138



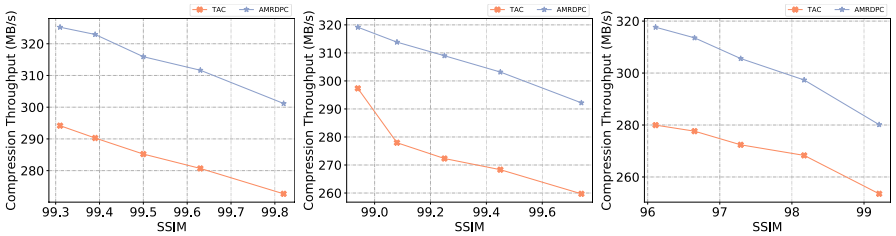
(a) Grid_Z2 (b) Grid_Z3 (c) Grid_Z5

Fig. 10 Comparison of compression throughput with different absolute error bounds



(a) Grid_Z2 (Level 0) (b) Grid_Z3 (Level 0) (c) Grid_Z5 (Level 0)

Fig. 11 Comparison of data compression speed under the same PSNR



(a) Grid_Z2 (Level 0) (b) Grid_Z3 (Level 0) (c) Grid_Z5 (Level 0)

Fig. 12 Comparison of data compression speed under the same SSIM

and focus on the differences between AMRDPC and TAC when dealing with specific level-based medium-density AMR data. At identical PSNR or SSIM levels, AMRDPC consistently delivers faster compression than TAC. These results confirm that for medium-density AMR data, our method accelerates processing without compromising data quality.

5.6 System-level evaluation on Tianhe supercomputer

To quantitatively validate our core claim that AMRDPC mitigates real-world I/O bottlenecks, we conducted system-level experiments on the Tianhe supercomputing platform. The results demonstrate that our method delivers substantial gains by targeting the most costly aspects of HPC I/O: storage capacity and end-to-end workflow efficiency.

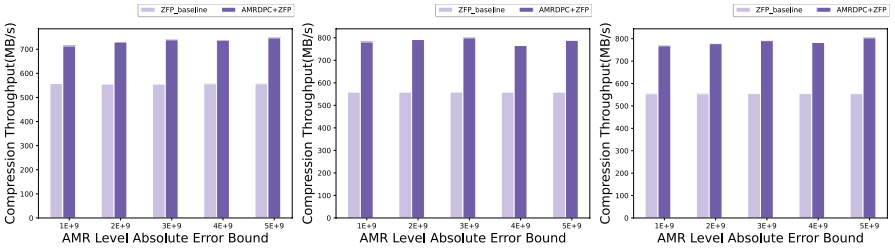
As quantified in Table 6, AMRDPC directly attacks this problem by reducing storage requirements by over 98% across all tested datasets. This effectively increases the effective storage capacity by more than 50×, which proportionally reduces the capital cost of storage hardware and the operational cost for data

Table 6 System-level storage benefits

Dataset	Original → Compressed	Storage reduction	CR	I/O Speedup
Grid_Z2	656.38 MB → 8.40 MB	98.72%	78.13%	82.3×
Grid_Z3	700.41 MB → 9.62 MB	98.63%	72.83%	73.8×
Grid_Z5	652.43 MB → 10.67 MB	98.36%	61.12%	61.1×

Table 7 Comparative performance on Tianhe supercomputers

Dataset	Compression time (s)		Decompression time (s)		Total time (s)	
	TAC	AMRDPC	TAC	AMRDPC	TAC	AMRDPC
Grid_Z2	7.95	7.01	3.21	2.30	11.16	9.31
Grid_Z3	6.55	6.12	2.54	2.21	9.09	8.33
Grid_Z5	7.90	7.55	2.75	2.73	10.65	10.28



(a) Grid_Z2 (b) Grid_Z3 (c) Grid_Z5

Fig. 13 AMRDPC+ZFP compression throughput at varied asolute error bounds

Table 8 Information loss metrics on three high-density AMR datasets

Method	σ	Run2_T2		Run2_T3		Run2_T4	
		CR	CT	CR	CT	CR	CT
ZFP baseline	1E+9	6.14	80.27	0.42	23.09	0.14	3.25
	2E+9	13.58	81.16	1.26	24.06	0.18	3.42
	3E+9	24.26	81.24	3.85	24.82	0.26	3.78
	4E+9	36.41	83.18	4.06	25.18	0.31	4.15
	5E+9	42.14	83.25	4.68	25.49	0.35	4.49
AMRDPC+ZFP	1E+9	60.07	218.80	64.27	208.05	15.20	117.89
	2E+9	68.20	223.60	72.07	219.34	18.73	108.23
	3E+9	78.81	233.84	81.91	222.90	22.38	122.47
	4E+9	80.83	234.48	82.95	218.31	24.06	113.48
	5E+9	92.86	237.68	94.38	230.72	32.30	106.78

transfers over the network or to/from tape archives. Furthermore, AMRDPC also reduces the total computational overhead compared to a baseline method (TAC).

Table 7 shows that AMRDPC achieves faster compression and decompression times across all datasets. This acceleration directly results in faster end-to-end workflow execution.

5.7 Generality analysis

To address the question of whether our AMRDPC method is specifically tailored to the SZ compressor or represents a general-purpose optimization, we conducted comprehensive generality experiments with ZFP, a compressor based on orthogonal transforms rather than data prediction.

As illustrated in Fig. 13, for medium-density AMR data, AMRDPC+ZFP consistently outperforms the ZFP baseline across various error bounds. A notable example is the Grid_Z5 dataset, where our method achieves a compression throughput improvement of up to 44.86%.

The performance advantage is even more pronounced on high-density AMR datasets, as quantitatively summarized in Table 8. The results for datasets Run2_T2, T3, and T4 demonstrate that AMRDPC+ZFP delivers dramatically higher compression ratios (CR) and compression throughput (CT) compared to ZFP alone. For instance, on the Run2_T2 dataset with an error bound of 1E+9, AMRDPC+ZFP boosts the compression ratio from 6.14 to 60.07 and accelerates the compression throughput from 80.27 to 218.80 (a 2.7× speedup). This trend of substantial improvement is consistent across all tested configurations. Our analysis confirms the superior generality of the integrated AMRDPC+ZFP approach.

5.8 Discussion

This work focuses on application-layer compression, which addresses the fundamental challenge of storing complex, irregular scientific data structures such as AMR. Looking ahead, we propose a collaborative, multi-layered compression pipeline for HPC systems. In this envisioned paradigm, domain-specific methods like ours form the foundational first stage. This stage achieves significant data reduction for long-term storage while adhering to strict scientific constraints. The resulting output is a substantially smaller dataset that retains full semantic integrity. This compacted dataset subsequently serves as the ideal input for a network-layer compression stage, where high-speed lossless techniques such as Blosc or LZ4 can be applied with greater efficiency. By operating on this pre-reduced data, network transfer times are minimized without compromising the storage efficiency gains achieved in the first stage. This synergistic approach establishes a holistic strategy wherein each layer addresses the specific problem it is best suited for, culminating in an optimal end-to-end data handling solution.

6 Related work

Lossy compression for scientific data: Data compression is becoming a critical technique in the HPC domain. Although lossless compression can ensure data fidelity, it is not suitable for processing large-scale scientific data [36, 37]. Today's Scientific applications often require more than 10× data reduction ratio [11]. However, state-of-the-art lossless compressors can only achieve a low data compression ratio (about 2×), which to some extent restricts the widespread application of this technology [38]. Therefore, lossy compressors have been favored by researchers in recent years and have been widely used in the HPC community. In general, they can be divided into four categories: prediction-based models, transformation-based models, higher-order singular value decomposition-based models, and machine learning-based models. The prediction-based compressors predict the current data based on the adjacent data, such as using Lorenzo prediction method [15], spline interpolation prediction method [17], etc., and then use quantization methods to control the prediction error within the error range specified by the users, which effective ensure data fidelity. Typical compressors include FPZIP, SZ2 [16], SZ3 [17], QoZ [18],

MGARD [19]. The principle of the transform-based compressors is to transform the original data into another coefficient domain to effectively remove the dependencies between the data. The coefficient data obtained by the transformation is easier to compress due to its relatively high sparsity. A typical compressor is ZFP, which compresses and decompresses data at the block level and uses orthogonal block transforms to decorrelate the data and embedded encoders to compress the transform coefficients [39]. One typical dimension-reduction compressor is TTHRESH, which is intended for Cartesian grid data of three or more dimensions and leverages the higher-order singular value decompression(HOSVD) to compress data [21]. Compressors based on machine learning leverage neural network techniques such as autoencoders to compress and reconstruct data. Typical compressors such as LFZIP [40] and AE-SZ [41]. Despite the recent impressive success of lossy compression, existing work on how to effectively combine error-bounded lossy compression with AMR applications is still in its infancy, and there is still much room for improvement.

Exploration for AMR applications: AMR is an efficient numerical technique that is widely used in the HPC community to solve partial differential equations. In recent years, some works have emerged in the research of large-scale AMR applications. Specifically, it is divided into offline AMR data compression and in-situ AMR data compression. zMesh provides an offline compression solution for AMR data, which is designed to exploit data redundancy at different AMR levels [23]. It reorders the AMR data in different refinement levels through methods such as Z-ordering and forms a 1D array, further improving the smoothness of the data. However, when eliminating the high redundancy between different AMR levels data, zMesh does not make good use of the similarity among the adjacent AMR levels, besides, it takes a lot of time to rebuild the AMR hierarchy. To address this problem, LAMP proposes a level-associated mapping method that fully considers the data similarity between AMR adjacent levels [24]. Compared with the verification of zMesh on the AMR applications, LAMP effectively reduces the time overhead of AMR hierarchy construction while improving the data compression ratio. To further explore the high-dimensional AMR compression, TAC provides a 3D compression method [25]. Different from the 1D compression strategy, TAC considers the density distribution of AMR data at different levels, efficiently divides different areas, and proposes multiple strategies to solve data processing under different densities to match the data compression mechanism of existing data compressors. In the in-situ compression of AMR data, AMRIC further reduces the I/O cost by introducing a 3D in-situ AMR compression framework through HDF5 while improving compression quality for AMR applications [26].

7 Conclusions

In conclusion, we proposed an efficient lossy compression framework AMRDPC for AMR applications, which improves the data compression ratio while reducing the computational overhead of data processing. AMR hierarchical data has the characteristic of uneven density distribution, and we designed two optimization strategies

to deal with it, respectively. Our main contributions include two points. First, a fast k-d tree data backfilling density grid strategy is designed for medium-density AMR data, which can further improve data compression throughput while ensuring that the data compression ratio remains unchanged. Second, for high-density AMR data, we designed a loop reversal data patching strategy, which can effectively improve the data compression ratio. We evaluate the effectiveness of the AMRDPC method on seven practical AMReX application datasets. Experimental results show that AMRDPC improves the data compression ratio by up to 5.73× while ensuring high data fidelity compared with the state-of-the-art method. The data compression speed is increased by up to 18.83% without reducing the data quality.

In future work, we plan to investigate how to scale our optimization technique on GPUs and apply our compression framework to in-situ AMR applications. Furthermore, we will evaluate AMRDPC in a wide range of HPC systems.

Acknowledgements This work was supported in part by the National Key R&D Program of China (Grant No. 2023YFB3001705); in part by the National Natural Science Foundation of China (Grant No. U21A20461, 62472160, 62572180, 2024JJ7375); in part by the Natural Science Foundation of Hunan Province of China (Grant No. 2024JJ7375) and in part by the Aid Program for Science and Technology Innovative Research Team in Higher Educational Institutions of Hunan Province.

Author contributions Yida Li wrote the main manuscript text and all authors reviewed the manuscript.

Data availability No datasets were generated or analyzed during the current study.

Declarations

Conflict of interest The authors declare no conflict of interest.

References

1. Jin S, Di S, Vivien F, Wang D, Robert Y, Tao D, Cappello F (2024) Concealing compression-accelerated i/o for hpc applications through in situ task scheduling. In: Proceedings of the Nineteenth European Conference on Computer Systems, pp. 981–998
2. Zhang W, Almgren A, Beckner V, Bell J, Blaschke J, Chan C, Day M, Friesen B, Gott K, Graves D (2019) Amrex: A framework for block-structured adaptive mesh refinement. J Open Source Softw 4(37):1370–1370
3. Stone JM, Tomida K, White CJ, Felker KG (2020) The athena++ adaptive mesh refinement framework: Design and magnetohydrodynamic solvers. Astrophys J Suppl Ser 249(1):4
4. Zhang W, Almgren AS, Day M, Nguyen T, Shalf J, Unat D (2016) Boxlib with tiling: An adaptive mesh refinement software framework. SIAM J Sci Comput 38(5):S156–S172
5. Colella P, Graves D, Ligocki T, Martin D, Modiano D, Serafini D, Van Straalen B Chombo (2009) software package for amr applications design document. Available at the website: URL:http://seesar lbl.gov/ANAG/chombo/
6. Dubey A, Almgren A, Bell J, Berzins M, Brandt S, Bryan G, Colella P, Graves D, Lijewski M, Löffler F (2014) A survey of high level frameworks in block-structured adaptive mesh refinement packages. J Parallel Distrib Comput 74(12):3217–3227
7. Sala K, Rico A, Beltran V (2020) Towards data-flow parallelization for adaptive mesh refinement applications. In: International Conference on Cluster Computing, pp. 314–325
8. Nguyen T, Unat D, Zhang W, Almgren A, Farooqi N, Shalf J (2016) Perilla: Metadata-based optimizations of an asynchronous runtime for adaptive mesh refinement. In: International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 945–956

9. Xie B, Oral S, Zimmer C, Choi JY, Dillow D, Klasky S, Lofstead J, Podhorszki N, Chase JS (2020) Characterizing output bottlenecks of a production supercomputer: Analysis and implications. ACM Trans Storage 15(4):1–39
10. Fedeli L, Huebl A, BoillodCerneux F, Clark T, Gott K, Hillairet C, Jaure S, Leblanc A, Lehe R, Myers A, Piechurski C, Sato M, Zaim N, Zhang W, Vay J, Vincenti H (2022) Pushing the frontier in the design of laser-based electron accelerators with groundbreaking mesh-refined particle-in-cell simulations on exascale-class supercomputers, pp. 1–12
11. Cappello F, Di S, Li S, Liang X, Gok AM, Tao D, Yoon CH, Wu X-C, Alexeev Y, Chong FT (2019) Use cases of lossy compression for floating-point data in scientific data sets. Int J High Perform Comput Appl 33(6):1201–1220
12. Meister D, Kaiser J, Brinkmann A, Cortes T, Kuhn M, Kunkel J (2012) A study on data deduplication in hpc storage systems. In: International Conference on High Performance Computing, Networking, Storage and Analysis, pp. 1–11
13. Dubois Y, BloemReddy B, Ullrich K, Maddison CJ (2021) Lossy compression for lossless prediction. Adv Neural Inf Process Syst 34:14014–14028
14. Di S, Cappello F (2016) Fast error-bounded lossy hpc data compression with sz. In: IEEE International Parallel and Distributed Processing Symposium, pp 730–739
15. Tao D, Di S, Chen Z, Cappello F (2017) Significantly improving lossy compression for scientific data sets based on multidimensional prediction and error-controlled quantization. In: IEEE International Parallel and Distributed Processing Symposium, pp 1129–1139
16. Liang X, Di S, Tao D, Li S, Li S, Guo H, Chen Z, Cappello F (2018) Error-controlled lossy compression optimized for high compression ratios of scientific datasets. In: IEEE International Conference on Big Data, pp 438–447
17. Zhao K, Di S, Dmitriev M, Tonellot TD, Chen Z, Cappello F (2021) Optimizing error-bounded lossy compression for scientific data by dynamic spline interpolation. In: International Conference on Data Engineering, pp 1643–1654
18. Liu J, Di S, Zhao K, Liang X, Chen Z, Cappello F (2022) Dynamic quality metric oriented error bounded lossy compression for scientific datasets. In: International Conference for High Performance Computing, Networking, Storage and Analysis, pp 1–15
19. Liang X, Whitney B, Chen J, Wan L, Liu Q, Tao D, Kress J, Pugmire D, Wolf M, Podhorszki N (2021) Mgard+: Optimizing multilevel methods for error-bounded scientific data reduction. IEEE Trans Comput 71(7):1522–1536
20. Lindstrom P, Isenburg M (2006) Fast and efficient compression of floating-point data. IEEE Trans Visual Comput Graphics 12(5):1245–1250
21. BallesterRipoll R, Lindstrom P, Pajarola R (2020) Tthresh: Tensor compression for multidimensional visual data. IEEE Trans Visual Comput Graphics 26(9):2891–2903
22. Liang X, Gong Q, Chen J, Whitney B, Wan L, Liu Q, Pugmire D, Archibald R, Podhorszki N, Klasky S (2021) Error-controlled, progressive, and adaptable retrieval of scientific data with multilevel decomposition. In: International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–13
23. Luo H, Wang J, Liu Q, Chen J, Klasky S, Podhorszki N (2021) zmesh: Exploring application characteristics to improve lossy compression ratio for adaptive mesh refinement. In: IEEE International Parallel and Distributed Processing Symposium, pp 402–411
24. Li Y, Luo H, Li F, Wang J, Li K (2023) Lamp: Improving compression ratio for amr applications via level associated mapping-based preconditioning. IEEE Trans Comput 72(12):3370–3382
25. Wang D, Pulido J, Grosset P, Jin S, Tian J, Ahrens J, Tao D (2022) Tac: Optimizing error-bounded lossy compression for three-dimensional adaptive mesh refinement simulations. In: International Symposium on High-Performance Parallel and Distributed Computing, pp 135–147
26. Wang D, Pulido J, Grosset P, Tian J, Jin S, Tang H, Sexton JM, Di S, Zhao K, Fang B, Lukic Z, Cappello F, Ahrens JP, Tao D (2023) Amric: A novel in situ lossy compression framework for efficient i/o in adaptive mesh refinement applications. In: International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–15
27. Xu L, Anthony Q, Zhou Q, Alnaasan N, Gulhane R, Shafi A, Subramoni H, Panda DKDK (2024) Accelerating large language model training with hybrid gpu-based compression. In: IEEE International Symposium on Cluster, Cloud and Internet Computing, pp 196–205
28. Lee M, Jin H, Kim I, Kim T (2009) Improving TCP goodput over wireless networks using kernel-level data compression. In: Proceedings of the 18th International Conference on Computer Communications and Networks, pp 1–6

29. Li Y, Kashyap A, Chen W, Guo Y, Lu X (2024) Accelerating lossy and lossless compression on emerging bluefield DPU architectures. In: IEEE International Parallel and Distributed Processing Symposium, pp. 373–385
30. Roache PJ, Knupp PM (1993) Completed richardson extrapolation. Commun Numer Methods Eng 9(5):365–374
31. Lu T, Liu Q, He X, Luo H, Suchyta E, Choi J, Podhorszki N, Klasky S, Wolf M, Liu T (2018) Understanding and modeling lossy compression schemes on hpc scientific data. In: IEEE International Parallel and Distributed Processing Symposium, pp 348–357
32. Tao D, Di S, Liang X, Chen Z, Cappello F (2019) Optimizing lossy compression rate-distortion from automatic online selection between sz and zfp. IEEE Trans Parallel Distrib Syst 30:1857–1871
33. Zou X, Lu T, Xia W, Wang X, Zhang W, Zhang H, Di S, Tao D, Cappello F (2020) Performance optimization for relative-error-bounded lossy compression on scientific data. IEEE Trans Parallel Distrib Syst 31(7):1665–1680
34. Bentley JL (1975) Multidimensional binary search trees used for associative searching. communications of the ACM 18(9), 509–517
35. Hoang D, Bhatia H, Lindstrom P, Pascucci V (2024) Progressive tree-based compression of large-scale particle data. IEEE Transact Vis Comput Graph 30(7):4321–4338
36. Burtscher M, Ratanaworabhan P (2009) Fpc: A high-speed compressor for double-precision floating-point data. IEEE Trans Comput 58(1):18–31
37. Meister D, Kaiser J, Brinkmann A, Cortes T, Kuhn M, Kunkel JM (2012)A study on data deduplication in hpc storage systems. In: International Conference for High Performance Computing, Networking, Storage and Analysis, pp 1–11
38. Lu T, Zhong Y, Sun Z, Chen X, Zhou Y, Wu F, Yang Y, Huang Y, Yang Y (2023) Adt-fse: A new encoder for sz. In: International Conference for High Performance Computing, Networking, Storage and Analysis, pp 1–13
39. Lindstrom P (2014) Fixed-rate compressed floating-point arrays. IEEE Trans Visual Comput Graphics 20(12):2674–2683
40. Chandak S, Tatwawadi K, Wen C, Wang L, Ojea JA, Weissman T (2020)Lfzip: Lossy compression of multivariate floating-point time series data via improved prediction. In: Data Compression Conference, pp. 342–351
41. Liu J, Di S, Zhao K, Jin S, Tao D, Liang X, Chen Z, Cappello F (2021) Exploring autoencoder-based error-bounded compression for scientific data. In: International Conference on Cluster Computing, pp 294–306

Authors and Affiliations

Yida Li[1] · Huizhang Luo[1] · Yufeng Zhang[1] · Keqin Li[1] · Kenli Li[1]

✉ Huizhang Luo
luohuizhang@hnu.edu.cn

Yida Li
liyida@hnu.edu.cn

Yufeng Zhang
yufengzhang@hnu.edu.cn

Keqin Li
likq@hnu.edu.cn

Kenli Li
lkl@hnu.edu.cn

[1] The College of Computer Science and Electronic Engineering, Hunan University, ChangSha 410082, China