arXiv:2301.01901v3 [cs.DC] 5 Dec 2023

TAC+：优化三维AMR模拟的误差有界有损压缩

Daoce Wang、Jesus Pulido、PascalGrosset、Sian Jin、Jiannan Tian、KaiZhao、James Ahrens 和 Dingwen Tao，*IEEE*高级会员

摘要—如今，科学模拟产生的数据量极其庞大，而 I/O 带宽和存储空间有限，因此亟需大幅缩减数据量。误差有界有损压缩被认为是解决这一问题最有效的方法之一。然而，针对自适应网格细化（AMR）模拟数据改进误差有界有损压缩的研究还很少。不同于以往仅采用一维压缩的工作，我们提出了一种方法（TAC），对 AMR 数据的每个细化层级采用高维 SZ 压缩。为消除不同层级间的数据冗余，我们提出了若干预处理策略，并根据数据特征自适应地使用这些策略。我们进一步改进 TAC，得到 TAC+：通过优化 SZ 压缩的无损编码阶段，高效处理预处理后产生的大量小型 AMR 数据块。在来自三个真实世界大规模 AMR 模拟的 10 个 AMR 数据集上进行的实验表明，与最先进的方法相比，在数据失真相同的情况下，TAC+ 可将压缩比提高至多 4.9×。此外，我们利用该方法的灵活性，为各层级调整误差界，从而在两个应用特定指标上显著降低数据失真。

索引词—**数据约简、有损压缩、自适应网格细化（AMR）、科学计算。**

I. 引言

过去几十年来，超级计算机性能的提升仍不足以解决许多具有挑战性的建模与仿真问题。例如，求解演化偏微分方程的复杂度按 $\Omega(n^4)$ 增长，其中 n 是每个维度上的网格点数。因此，过去 30 年间约三个数量级的性能提升，仅使时空分辨率提高了 5.6× [1]。为解决这一问题，许多高性能计算（HPC）模拟软件包 [2]（例如 AMReX [3] 和 Athena++ [4]）采用自适应网格细化（AMR）——仅对最受关注的特定区域进行计算——以提高分辨率。与在所有区域都采用高分辨率的方法相比，AMR 方法大幅降低了计算复杂度和存储开销，因此成为许多高性能计算应用中最广泛使用的框架之一 [5]–[8]。

尽管 AMR 在一定程度上可以节省存储空间，但在超级计算机上运行的 AMR 应用仍会生成大量数据，给数据传输和存储带来挑战。例如，一次 Nyx 模拟 [9] 的分辨率为 4096^3（即粗层级有 0.5×2048^3 个网格点，细层级有 0.5×4096^3 个网格点），单个快照最多可生成 1.8 TB 的数据；假设该模拟运行 5 次，每次模拟转储 200 个快照，则总共需要 1.8 PB 的磁盘存储。因此，减少数据大小对于降低存储开销和 I/O 成本、提升在超级计算机上运行大规模 AMR 模拟的整体应用性能至关重要。

解决这一问题的直接方法是使用数据压缩。然而，传统的无损压缩技术（如 GZIP [10] 和 Zstandard [11]）对科学数据 [12]的压缩比最高只能达到 2× 。另一方面，新一代有损压缩器能够提供严格的误差控制（称为“误差有界”有损压缩），例如 SZ [13]–[15],ZFP [16],MGARD [17], [18], 和 TTHRESH [19]。如 [20]–[27]所示，使用这些误差有界有损压缩器，科学家可以在尽量减少重构数据质量损失及其后续分析影响的同时，获得相对较高的压缩比。

然而，数据压缩与 AMR 数据之间仍存在差距。这种差异的根源在于 AMR 数据的层次结构特性：整个数据集具有不同的分辨率/层级。每个层级的数据都是稀疏的，因为它只覆盖区域的一部分。然而，目前的科学压缩器仅支持压缩分辨率均匀的非稀疏数据。因此，压缩前必须对 AMR 数据进行预处理。

一种直接的预处理方法是将不同层级的高维 AMR 数据展平为一维数组后进行压缩。然而，这种方法会导致数据丢失大量空间信息，而这些信息对于压缩性能优化至关重要。

为了利用高维压缩，一种常见方法是对粗层级数据进行上采样，并将其与最细层级数据合并，从而生成均匀分辨率数据。然而，这种方法会引入冗余信息，显著降低压缩比。当上采样率较高或需要对多个粗层级进行上采样时，这种性能下降尤为明显。这使 AMR 数据压缩陷入两难：我们不得不在丢失空间局部性（采用一维压缩）和引入冗余信息（采用上采样后的三维数据）之间做出选择。

目前只有少数研究成果探讨了面向自适应网格细化应用和数据集的先进误差有界有损压缩。近期，Luo 等人提出了 zMesh [28], ，这是一种技术——

Daoce Wang、Jiannan Tian 和 Dingwen Tao（通讯作者）就职于美国印第安纳大学布卢明顿分校，邮编 47405，印第安纳州布卢明顿。Jesus Pulido、Pascal Grosset 和 James Ahrens 就职于美国洛斯阿拉莫斯国家实验室，邮编 87545，新墨西哥州洛斯阿拉莫斯。Sian Jin 就职于美国天普大学，邮编 19122，宾夕法尼亚州费城。Kai Zhao 就职于美国佛罗里达州立大学，邮编 32306，佛罗里达州塔拉哈西。

该技术通过将映射到相同或相邻几何坐标的数据点分组来预处理数据，从而使数据集更平滑、更易压缩。然而，由于 zMesh 将来自不同自适应网格细化层级的数据点映射到相邻的几何坐标，并生成一维数组，因此仍无法采用大多数 HPC 模拟所使用的三维压缩。此外，zMesh 是为基于网格块的自适应网格细化数据设计的[1]，而 zMesh 提出的重组方法无法适当地提升数据平滑性（见第四节）。

为解决这些问题，我们提出了 TAC，它会移除较粗层级中的冗余数据，并对每个层级进行三维有损压缩。我们注意到，每个层级可能包含许多空区域/零值区域，其中的数据点存储在其他层级中，这可能会显著降低数据平滑性，进而降低压缩比。为此，TAC 根据空区域的密度，采用自适应分区策略移除这些空区域，或使用适当的值对其进行部分填充。

另一个挑战是，划分策略可能会生成许多（例如，3,000+）个较小的数据块，而由于霍夫曼编码开销，SZ压缩器在处理小型数据集时表现不佳（详见第III-D节）。针对霍夫曼编码开销过大的问题，TAC 的解决方案是将较小的数据块线性化/合并，然后将合并后的数据块传递给 SZ。这种方法可以降低霍夫曼编码的开销。然而，TAC 仍面临一个关键限制：合并后的较小数据块大多在原始数据集中并不相邻，导致这些非邻居数据块之间的数据值迅速变化，从而可能对 SZ 预测器的精度产生负面影响。

为解决这些局限性，我们进一步优化 TAC，将其升级为 TAC+ ，并为 SZ 压缩器设计了共享霍夫曼编码（SHE）方法。该方法允许对每个小块进行独立预测，同时使用单棵共享霍夫曼树进行编码，从而提高预测准确率和压缩比。

主要研究成果总结如下。

- 我们提出利用三维 SZ 压缩分别压缩 AMR 数据集的每个层级。我们基于以下三种预处理策略和数据特征，提出了一种混合压缩方法。
- 我们提出了一种优化稀疏张量表示，以便高效划分数据，并移除稀疏 AMR 数据中的空区域。
- 我们提出了一种改进的 k-d 树方法，以减少移除空区域所需的时间开销。
- 我们提出了一种填充方法，以改善密集 AMR 数据的平滑性和可压缩性。
- 我们在 SZ 压缩器中采用 SHE 方法，以降低压缩分区后的 AMR 数据时处理多个小数据块所产生的高时间和存储开销，从而实现更好的压缩效果。
- 我们为每个 AMR 层级调节误差界，以进一步提升压缩质量，具体以两项应用特定的后分析指标衡量。
- 实验表明，与最先进的方法 zMesh 相比，在测试数据集上实现相同数据失真的情况下，我们提出的 AMR 压缩方法可将压缩比提高至多 4.9×

我们在来自三个真实世界自适应网格细化应用的十个数据集上评估所提出的压缩方法：Nyx [9],WarpX [29],和 IAMR [30]。我们将该方法与四种基线方法（包括 zMesh）进行比较，采用压缩比和峰值信噪比（PSNR）等通用指标，以及功率谱和晕团搜寻器等应用特定指标。我们的代码可在 https://github.com/hipdac-lab/ HPDC22-TAC 获取。

本文其余部分组织如下。第二节介绍误差有界有损压缩、自适应网格细化方法、 k-d 树的背景知识，以及自适应网格细化数据压缩的相关工作。第三节介绍我们提出的预处理策略、SHE 方法和混合压缩。第四节展示在不同自适应网格细化数据集上的实验结果。第五节总结本文并讨论未来工作。

II. 背景与相关工作

A. 科学数据的有损压缩

数据压缩主要分为两类：无损压缩和有损压缩。与无损压缩相比，有损压缩可以通过牺牲少量精度来实现更高的压缩比 [31]。针对图像和视频，已有一些成熟的有损压缩器，例如 JPEG [32]和 MPEG [33],，但它们处理科学数据的效果不佳，因为它们主要针对整数而非浮点数设计。

近年来，出现了新一代面向科学数据设计的有损压缩器，例如 SZ [13]–[15],ZFP [16], MGARD [17], 和 TTHRESH [19]。这些有损压缩器提供了相关参数，使用户能够精细控制有损压缩引入的信息损失。与用于图像（整数数据）的传统有损压缩器（如 JPEG [32]）不同，SZ、ZFP、MGARD 和 TTHRESH 专为压缩浮点数据而设计，并能根据用户需求提供严格的误差控制方案。通常，有损压缩器提供多种压缩模式，例如误差界定模式和固定码率模式。误差界定模式要求用户设置误差类型（例如逐点绝对误差界和逐点相对误差界）以及误差界级别（例如 10^{-3} ）。压缩器确保原始数据与重构数据之间的差异不超过用户设定的误差界级别。

在这项工作中，我们重点关注 SZ 有损压缩（2021年R&D 100大奖得主），因为与 ZFP 相比，SZ 通常具有更高的压缩比 [23], [34] ，且速度比 MGARD [34], [35]和 TTHRESH [19]更快。SZ 是一种面向科学数据的基于预测的误差有界有损压缩器，主要包含三个步骤：(1) 使用不同的预测方法预测每个数据点的值；(2) 根据用户设定的误差界，量化真实值与预测值之间的差值；(3) 应用定制的霍夫曼编码和无损压缩。

[1]基于网格块的自适应网格细化数据会在当前粗层级中冗余存储下一更细层级将要细化的数据块（见第二节 C）。

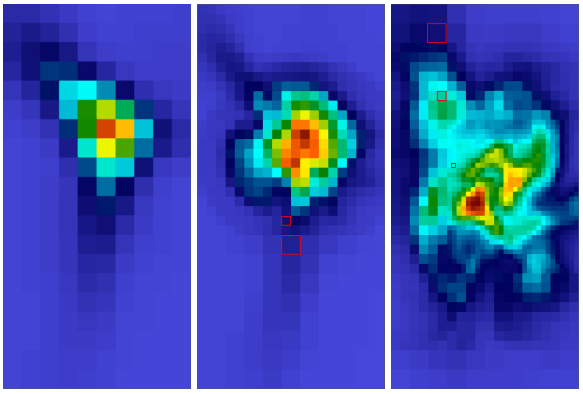


图 1：基于自适应网格细化的宇宙学模拟生成的三个关键时间步的可视化结果（一个放大的二维切片）。网格结构随宇宙演化而变化。红色方框表示同一自适应网格细化层级内的不同分辨率。

SZ框架包含一系列算法，以满足不同用户和应用的需求。在本研究中，我们主要关注其中两种核心算法：利用 Lorenzo 和线性回归预测器的压缩算法，称为 “Lor/Reg” [15]，；以及基于样条插值方法的压缩算法，称为“Interp”[34]。具体而言，Lor/Reg 方法首先将整个输入数据截分成较小的数据块，然后分别对每个数据块应用 Lorenzo 预测器或高维线性回归。相比之下，Interp 算法对完整数据集的全部三个维度进行全局插值。Lor/Reg 和 Interp 算法存在显著差异，因此在压缩自适应网格细化数据时，性能和特性也各不相同。关于这一点的更深入讨论见第 III-D、III-E 和 IV-E 节。

B. 自适应网格细化方法与自适应网格细化数据

自适应网格细化是一种通过使用非均匀网格来调整解的精度的方法，可在达到所需精度的同时提高计算效率并节省存储开销。自适应网格细化应用会根据模拟所需的细化层级改变网格或空间分辨率，并在更重要或更受关注的区域使用更精细的网格，在不太重要或较少受关注的区域使用更粗糙的网格。图 1 展示了当数值满足细化准则时网格将进行细化，例如，当某个块的梯度范数或最大值大于阈值时，就对该块进行细化。

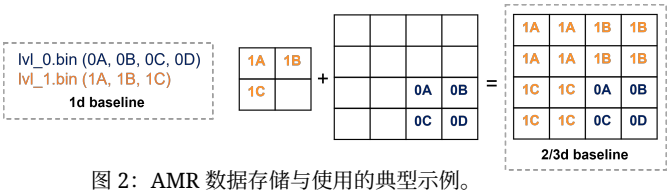


图 2：AMR 数据存储与使用的典型示例。

显然，AMR 应用生成的数据具有层次结构，且分辨率各不相同。每个 AMR 层级的数据通常分别存储（例如，存储在一维数组中）。例如，图 2（左）展示了一个简单的双层 AMR 数据示例；“0”表示高分辨率（细层级），“1”表示低分辨率（粗层级）。当需要对 AMR 数据进行后处理分析或可视化时，用户通常会将不同层级的数据转换为统一分辨率。在前面的示例中，我们会对粗层级的数据进行上采样，并将其与细层级的数据合并，如图 2（右）所示。

C. 基于树和基于网格块的 *AMR* 数据

表示 AMR 数据的技术有两种：基于网格块的 AMR 和基于树的 AMR [36]。两者的主要区别在于，基于网格块的 AMR 技术生成的数据在不同层级间存在冗余。换句话说，基于网格块的 AMR 数据结构会在当前层级冗余地保存下一层级将要细化的数据块，从而简化细化过程中的计算。相比之下，基于树的 AMR 技术将网格组织在树叶节点上，因此不同层级间不存在冗余数据。但与基于网格块的 AMR 数据相比，基于树的 AMR 数据更难进行后处理分析和可视化 [37], [38]。

在这项工作中，我们重点研究一种最先进的基于网格块的 AMR 框架 AMReX。请注意，由于基于网格块的 AMR 中较粗层级的冗余数据通常不会用于后处理分析，因此我们在压缩过程中将其舍弃，以提高压缩比。

D. 现有的自适应网格细化数据压缩

一维 *AMR* 压缩：AMR 数据压缩面临的主要挑战在于，AMR 数据内容丰富、具有层次结构，且分辨率各不相同。一种简单的方法是分别压缩每个 AMR 层级的一维数据。然而，这种方法会丢失大部分拓扑和空间信息，而这些信息对数据压缩至关重要。zMesh [28] 是基于一维方法的先进 AMR 数据压缩技术。与简单的一维方法不同，zMesh 根据各点在二维布局中的坐标重新组织一维数据；换言之，它将二维布局中彼此相邻的点在一维数组中排列得更近。对于在不同 AMR 层级之间存在冗余的基于块的 AMR 数据，这种方法能够提高数据的平滑性和可压缩性，从而有利于后续使用 SZ 等方法进行一维压缩。然而，zMesh 未利用高维压缩，而许多先前研究 [14], [39], [40] 已证明，利用更多维度的信息（例如空间或时间信息）可以显著提升压缩性能。此外，zMesh 仅关注二维 AMR 数据。我们的工作旨在利用高维数据压缩，并支持三维 AMR 数据。

高维 *AMR* 压缩：与第 II-B 节所述的思路类似，利用 3D 压缩处理三维自适应网格细化数据的一种直接方法，是通过对粗层级进行上采样，将不同层级的数据一起压缩。然而，这种方法必须处理上采样过程产生的额外冗余数据。如图 2 所示，*1A*、*1B*和*1C*是压缩过程中的冗余点。请注意，当粗层级中的数据更多或上采样率更高时，这些冗余点带来的存储开销会更大，尤其是对于三维自适应网格细化数据。这是因为对于二维 AMR 数据，上采样率为 2 时，我们只需将粗层级中的一个点复制 4 次；而对于三维自适应网格细化数据，则需复制 8 次，上采样

率为 2。该方法的另一个局限是，无法对不同 AMR 层级应用不同的压缩配置（例如误差界）。这是因为上采样后，所有数据点的重要性都相同。然而，采用 AMR 方法的目的是为不同 AMR 层级设定不同的关注程度，以便为每个 AMR 层级自适应地选择误差界。

E. k维树用于粒子数据压缩

k维树 [41] 是一种二叉树，其中每个节点代表一个特定空间。不失一般性，对于三维情形，k维树中的每个非叶节点通过与三个维度之一相关联的二维平面，将空间划分为两部分。左子空间与该节点的左子节点相关联，而右子空间与右子节点相关联。k维树通常用于粒子数据压缩 [42]–[44]，以定位每个粒子并移除空区域。具体而言，k维树沿一个维度不断将空间进行二分，直到空间为空或仅包含一个粒子为止。我们将优化经典的 k维树，并利用它移除空区域，提高每个AMR层级的可压缩性（详见III-C节）。

III. 我们提出的设计

本节介绍一种专为 AMR 数据设计的压缩框架，该框架对每个 AMR 层级采用高维 SZ 有损压缩。我们最初提出的 TAC 集成了三种预处理策略，以应对不规则数据分布带来的挑战。在 TAC 的基础上，我们提出了 TAC+。在 TAC+ 中，我们改进了原始 TAC 中的两种预处理方法，并引入共享霍夫曼编码（SHE）。随后，我们将 SHE 与 Lor/Reg 算法结合，以提高 AMR 数据的压缩效果。此外，我们还为 TAC/TAC+ 提出了自适应方法，使其能够根据各层级的数据密度选择最合适的预处理方法。

A. 高密度数据的幽灵壳填充

要在三维中压缩 AMR 数据，除了前面提到的三维基线之外，我们还可以在三维中分别压缩每个层级。不过，这样一来，数据会被拆分成多个层级，而每个层级都会有许多空区域，数据分布也不规则，如图4所示。处理不规则三维数据的一种简单方法是用

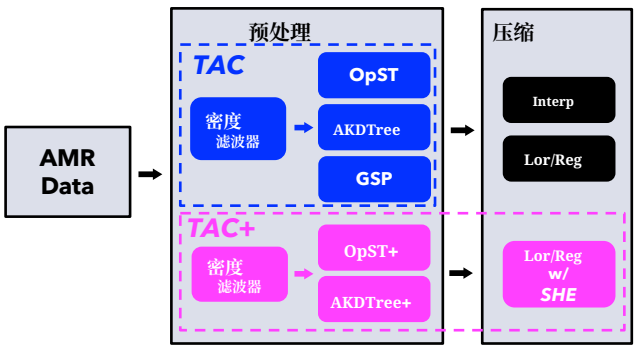
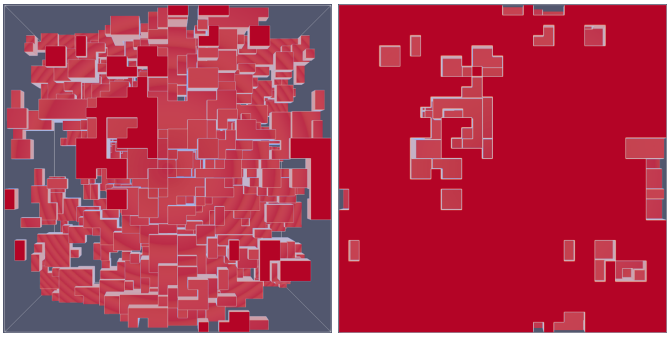


图 3：我们提出的 TAC 和 TAC+ 的工作流程概览。



（a）z10 细层级 （b）z10 粗层级

图 4：示例 AMR 数据“z10”的数据分布可视化，其中 z =表示红移。非空区域以红色显示。

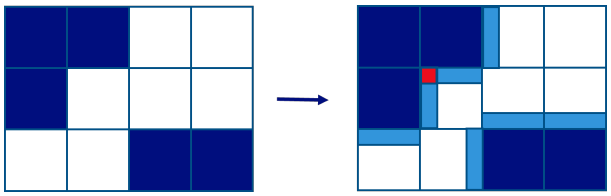


图 5：GSP方法的二维示例。非空数据块为藏蓝色；填充的数据块为浅蓝色或红色；基于多个非空邻居进行填充的数据块为红色。

零填充空区域，然后将一个大型三维块传递给压缩器。尽管填充的零会增加待压缩数据的大小，但对于高密度数据，例如图4b所示的 z10 粗层级（即密度约为 77%），大小开销会很小。

然而，这些填充的零值也会大幅降低压缩性能，尤其是对于 SZ 这类基于预测的有损压缩方法，因为这些零值会显著影响 SZ 的预测准确率，导致边界处出现较大的压缩误差，如图 6a 所示。具体而言，如第 III-B 节所述，SZ 使用每个点的邻居点的值来预测该点的值。因此，对于那些与填充零值相邻的边界点，SZ 会在预测中纳入零值，而这些空区域的实际值通常并非零（而是保存在其他 AMR 层级中），这会严重误导预测。

为消除上述填充零值的问题，我们提出使用幽灵壳填充策略（**GSP**），将邻居值扩散到填充层。图 5 展示了这一思路的总体概况，详细算法见算法 1。具体而言，我们首先将数据划分为单位块，然后使用其非空邻居边界数据值的平均值填充每个空单位块。

请注意，有些空单位块拥有多个非空邻居，例如图 5 中的红色方框。对于这些数据块，我们将使用所有邻居的平均值进行填充。相应地，在解压缩时，我们会根据保存的填充信息移除这些填充值。请注意，由于填充过程仅针对非空数据块，因此对于高密度数据（例如 0.1%），元数据开销几乎可以忽略不计。

填充后，每个边界点都将使用其所属或相邻单位块中的所有边界数据的平均值进行预测。如图 6 所示，与零填充（ZF）方法相比，GSP 能显著降低整体压缩误差，尤其是边界数据的压缩误差。

算法 1： 提出的幽灵壳填充方法

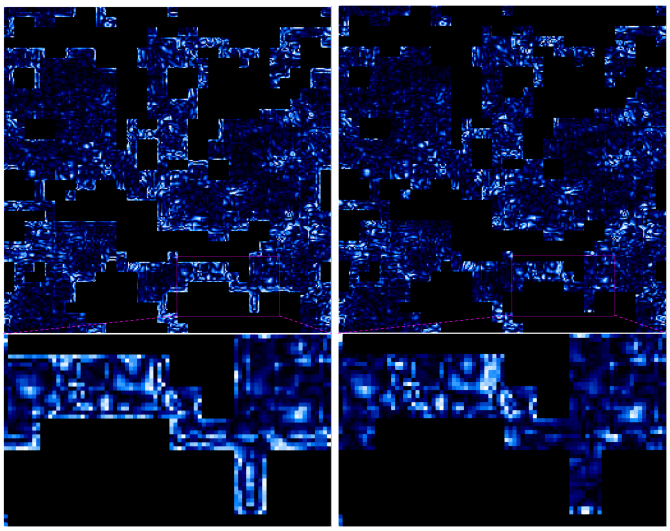
输入： 数据，m
输出： 填充后的数据
对于每个单位块 b_i **执行**
  $m = min(unitBlockSize/2,\ 4)$;
  如果 b_i 为空且 b_i 有非空邻居**则**
    对于每个非空邻居n_j **执行**
      填充切片 = avg（n_j 后续切片中的前 m 个至 b_i）；
      如果存在边缘重叠**则**
        $pad = pad/2$;
      否则如果 角点重叠 **则**
        $pad = pad/3$;
      else
        继续;
      end
      在 b_i 旁添加一个 m层的填充切片 n_j；
    end
  end
结束
返回填充后的数据

此外，对于这种高密度数据，GSP 方法能够达到与 ZF 方法相近的压缩比，因此率失真表现更好。

请注意，除了使用平均值外，我们还可以使用有助于压缩预测器为填充做出最准确预测的数值。不过，这需要额外运行一次压缩预测器来确定最佳数值，从而使这一过程更加耗时。此外，如图 6 所示，与零填充相比，平均填充已显著降低了边界处的压缩误差。因此，我们选择使用平均值，以避免牺牲过多性能。

B. 面向低密度数据的优化稀疏张量表示

当数据中的大多数区域为空时（例如，图 4a 中约 77% 的数据为空），大量



(a) 零填充（压缩率=156.7，峰值信噪比=32.8dB） (b) GSP（压缩率=161.3，峰值信噪比=33.5dB） 图 6：基于 Nyx 的“重子密度”场（即 z10 的粗层级，密度为 77%），使用 SZ 的两种方法的压缩误差视觉比较（一个切片）。颜色越亮表示压缩误差越高。误差界为 6.7×10^{-3} 的相对误差界。

填充数据会大幅增加待压缩数据的大小，导致压缩比偏低。

为了解决这一问题，我们提出采用一种朴素的稀疏张量方法（称为 **NaST**）来移除空区域，如图 7 所示。NaST 的压缩过程包括四个主要步骤：（1）将三维数据划分为多个单位块；（2）移除空数据块；（3）将剩余的三维块线性化为一个四维数组；（4）将该四维数组传递给压缩器。请注意，在解压缩过程中，我们会将解压后的四维数组中的单位块放回原始数据中。

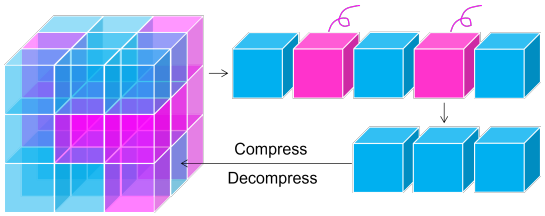


图 7：朴素稀疏张量（NaST）方法的工作流程（粉色标记空区域，蓝色标记非空区域）。

然而，为了彻底移除空区域并形成稀疏表示，单位块大小需要相对于输入数据大小足够小（例如，16^3 与512^3）。这会因分区而降低空间局部性。此外，很大比例的数据会位于数据块边界上。由于线性化后的单位块在原始数据集中可能并不连续，其边界不平滑，因此 SZ 等压缩器难以准确预测数值。”因此，未经单位块大小优化的 NaST 方法压缩性能较低。

为解决上述问题，我们提出了一种优化稀疏张量表示（称为 **OpST**），既能有效移除空区域，又能保持相对较大的单位块大小，从而提高空间局部性并减少边界数据所占比例。我们的算法详见算法2。我们以一个二维示例说明该方法，如图 8 所示。具体而言，(1) 我们将数据划分为许多较小的单位块。(2) 对于每个单位块，我们使用动态规划方法初始化一个数组 BS，用于保存以该单位块为右下角的最大正方形的维度/大小（第 6 行，下一段将对此进行讨论）。(3) 根据 BS 中保存的大小，从原始数据中提取子块（由多个单位块组成）（第 13 行）。(4) 由于提取后原始数据会发生变化，我们需要根据 *maxSide* 部分更新 BS（第 14 行，稍后将对此进行讨论）。我们从右下角到左上角循环执行 (3) 和 (4)，直到原始数据为空。(5) 提取所有子块后，我们根据其大小将它们放入多个三维数组中（以便进行压缩）。请注意，大小相同的子块会合并到同一个数组中，以便于压缩。

在步骤(2)中初始化 BS 时，我们从具有 $i = 0$ 或 $j = 0$ 的 $b'[i][j]$（即左上边缘）开始，其中 $b'[\cdot][\cdot]$ 是单位块：如果 $b'[i][j]$ 为空，则将 $BS[i][j]$ 设为 0，否则设为 1。对于其余单位块，如果该块为空，则将 $BS[i][j]$ 设为 0；否则，将 $BS[i][j]$ 设为以下数值加 1：

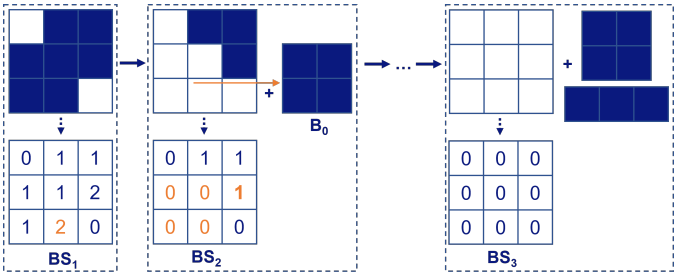


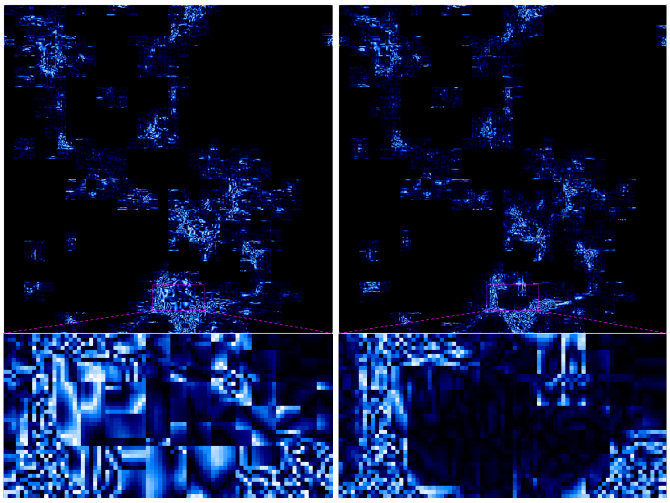
图8：我们提出的 OpST 方法的二维示例。根据保存在 BS 中的优化后的尺寸提取子块。例如，根据 $BS_1[2][1]$ 提取一个 2×2 子块 B_0。

算法2： 提出的优化稀疏张量方法

输入： 稀疏三维数据 S
输出： 多个四维数组 D_n

对于 每个单位块 $b(x, y, z)$ **执行**
  若 $b(x, y, z)$ 非空**则**
    若x 为 0**或**y 为 0**或**z 为 0**则**
      $BS(x,\ y,\ z) = 1$
    else
      $BS(x, y, z) = \min(BS(x-1, y, z),\ BS(x,\ y-1,\ z), BS(x, y, z-1),\ BS(x-1,\ y-1, z), BS(x, y-1, z-1),\ BS(x-1, y, z-1), BS(x-1, y-1, z-1)) + 1$;
      /* BS(x,y,z) 是右下后角位于索引为 (x,y,z) 的原始数据中的单位块所对应的最大立方体的维度大小 */
      $maxSide$ =最大值($maxSide,\ BS(x,\ y,\ z)$)
  end
end
结束
遍历每个单位块 $b(x, y, z)$ **执行**
  如果 $BS(x,\ y,\ z) \geq 1$ **则**
    $size = BS(x,\ y,\ z)$
    $D_{size} \leftarrow S((x-size:\ x) * blkSize,\ (y-size:\ y) * blkSize,\ (z-size:z) * blkSize)$；/* 将子块放入相应的四维数组 */
    $b(x-size:\ x,\ \ y-size:\ y,\ \ z-size:\ z) \leftarrow empty$
    $BS(x-size:\ \ x,\ y-size:\ \ y,\ z-size:\ \ z) = 0$
    $BS = updateBs(BS,\ \ x,\ \ y,\ \ z,\ \ maxSide)$
  end
结束
返回 D_n

其三个相邻块（即上方块、左侧块和左上块）中的最小值。换句话说，对于二维情况，我们有$BS[i][j] = 1+\min($ $BS[i][j-1],\ BS[i-1][j],\ BS[i-1][j-1])$。例如，$BS_1[2][1]$ 为 2，因为其所有左上方邻居的值都是 1（如图 8 所示）。然而，$BS_1[1][1]$ 和 $BS_2[1][2]$ 的值最多只能达到 1，因为它们各自有一个邻居的值为 0，因而无法组成尺寸为 2 的子块。接下来，在步骤 3 中，我们根据 BS 执行提取。例如，如图 8 所示，根据 $BS_1[2][1]$，我们提取一个 2×2 的子块 B_0，其右下角位于 $BS_1[2][1]$。此外，如步骤 (4) 所述，每次提取后都需要更新 BS。具体而言，对于提取的每个子块，都必须将 BS 中对应的值设为 0。例如，如图 8 所示，在 $BS_1[2][1]$ 处提取 2×2 的子块 B_0 后，需要将 $BS_2[1][0]$、$BS_2[1][1]$、$BS_2[2][0]$ 和 $BS_2[2][1]$ 设为 0。此外，还需要重新计算 BS 的一部分（算法2第 17 行），因为提取操作可能会影响其他 BS 的值。例如，提取 B_0 后，需要重新计算 $BS_2[1][2]$（以粗体橙色标出）。请注意，此更新是一个



(a) NaST（压缩率=245，峰值信噪比=77.5 dB）(b) OpST（压缩率=248，峰值信噪比=78.0 dB）图 9：使用基于 SZ 的两种方法对 Nyx 的“重子密度”字段（即 z10 的细层级，23% 密度）产生的压缩误差进行视觉比较（一个切片）。颜色越亮表示压缩误差越高。误差界为 7.2×10^{-4} 的相对误差界。

由于要更新的 BS 值将受*maxSide* 限制（即数据集中最大立方体的维度大小，第 7 行），因此可以进行部分更新。

与 NaST 类似，在解压缩过程中，我们将根据保存的坐标放回子块，以重构数据。请注意，经过我们的优化后，每个子块的大小将相对较大（例如，96^3，而原始数据大小为 512^3），因此保存所有子块坐标的开销可以忽略不计（例如 0.1%）。

最后，我们在图9中直观比较 NaST 和 OpST 的压缩质量。请注意，两者都使用 SZ，且误差界相同。颜色越亮，误差越大。可以观察到，与 NaST 方法相比，OpST 能显著降低整体压缩误差，尤其是边界上的数据点。值得注意的是，即使误差更低，我们的 OpST 仍能提供比 NaST 更高的压缩比。这是因为我们提出的优化会生成更大的子块，从而为 SZ 等基于预测的有损压缩器提供更多信息，以实现更好的率失真表现。详细评估将在第四节中展示。

C. 面向中密度数据的自适应 *k-D* 树

然而，针对低密度数据提出的 OpST 方法具有较高的计算开销，尤其是在数据相对密集时。这是因为，一方面，每次提取子块时，OpST 都需要根据*BS* 更新*maxSide*，而 *maxSide*越大，*BS* 中需要更新的值就越多；另一方面，*maxSide* 是数据集中最大非空立方体的维度大小，与数据集的密度密切相关。因此，OpST 的时间复杂度可表示为 $O(N^2 \cdot d)$，其中 N 为单位块数量，d 为密度。请注意，此处的密度表示数据的密集程度。例如，密度为 77% 意味着 23% 的数据为空。显然，当某个自适应网格细化层级的密度相对较高时，使用 OpST 会比较耗时。

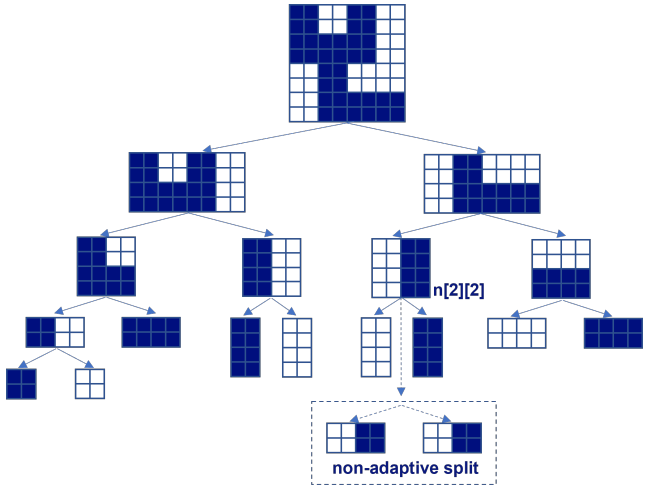


图 10：自适应 k维树的二维示例。子块将自适应地拆分，以有效移除空区域并获得更大的满子块。

算法 3： 动态 k维树

```
1
   输入：数据块 d，计数信息
   输出： k维树
2  节点.count ← 计数信息;
3  如果 d为空 或 d 已满 则
4      继续;                                   /* 停止划分 */
5  否则
6      如果 max(x,y,z)/min(x,y,z) > 2 且预划分尚未完成 则
7          沿最大维度进行划分，得到 d1, d2;
8          node.left  = AKDTree（d1）;
9          node.right = AKDTree（d2）;
10     else
11         如果 d 是一个立方体那么
12             将 d 等分为 8 个八分块:
                s1, ··· s8;
13             获取计数 c1, ...c8          对于 s1, ··· , s8;
14             查找最大差异划分 d1,d2;
15             节点.left  = AKDTree（d1，四 ci 个 d1）;
16             节点.right = AKDTree（d2，四 ci 个 d2）;
17         否则如果 d 是一个扁平长方体则
18             获取计数 c1, ···  ,c4 来自计数
                信息;
19             找到最大差异划分 d1,        d2;
20             节点.left  =AKDTree（d1，两个 ci 的 d1）;
21             node.right =AKDTree（d2，两个 ci 的 d2）;
22         否则，如果 d 是一个细长长方体则
23             从计数信息中获取计数 c1, c2
                信息;
24             沿最大维度拆分 d以获得
                d1,d2;
25             节点.left  =AKDTree（d1,  c1）;
26             节点.right =AKDTree (d2, c2);
27     end
28 结束
29 返回节点;
```

为了解决上述 OpST 开销过高的问题，我们提出一种自适应 k 维树，称为**AKDTree**，用于移除空区域并提取子块（包含多个单位块）。AKDTree 的时间复杂度较低，为 $O(\frac{1}{3}N \cdot \log N)$（稍后将对此进行讨论）。图 10 展示了一个简单的二维示例。具体而言，(1) 我们将数据划分为较小的单位块。(2) 我们使用一棵树对整个数据进行分层表示。树中的每个节点都与一个数据子块相关联。此外，每个节点还存储与其关联的子块中的非空单位块数量。(3) 对于每个节点，我们从中间沿着其关联的子块进行划分沿一个维度进行划分，为其两个子节点生成两个子块。请注意，在保留数据三维特征的同时，我们选择一个维度，使两个子节点中非空单位块数量的差异最大（将在接下来的两段中讨论）。(4) 我们持续划分一个节点，直到其中不再包含空单位块，或该节点本身为空。(5) 构建树完成后，我们收集所有叶节点并将其发送给压缩器。请注意，非空叶节点不含任何空单位块；否则，它会继续被划分。因此，如图 10 所示，叶节点必定为空节点或满节点。更多细节见算法 1。

如步骤 (3) 所述，我们将非空单位块不均匀地分配给每个节点的两个子节点。这样做是为了尽可能增加具有较大子块尺寸的叶节点数量。较大的数据块包含更多空间信息，有助于提升压缩效果。如果我们始终以固定方式划分子块，例如，第一次沿 x 轴划分，第二次沿 y 轴划分，第三次沿 x 轴划分，第四次沿 y 轴划分，依此类推，那么节点 $n[2][2]$ 将得到一个 2×2 的子块，如虚线框所示；而其可能的最大子块尺寸则为 4×2，如图 10 所示。

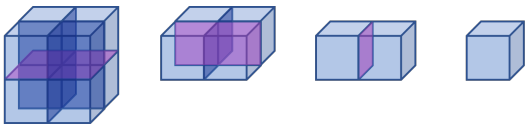


图 11：自适应划分示例，不同形状可选择的划分方式数量不同。

为了选择一个维度，将其非空单位块不均匀地分配给两个子节点，我们提出了动态划分方法。

为了保持数据集的高维特性，优先沿最大维度进行划分至关重要，尤其是当该维度明显大于其他维度时。例如，对于大小为 $8 \times 8 \times 64$ 的数据集，沿 z 轴而非 x 轴或 y 轴划分更为合适。沿 x 轴或 y 轴划分可能会使三维数据集降为二维。因此，我们最初采用的方法是沿主导维度划分数据集，直到：

$$max(x, y, z)/min(x, y, z) < 2 \tag{1}$$

然后，我们将节点分为三种不同类型：“立方体”节点、“扁平”节点和“细长”节点，其维度比分别为 x:y:z、2x:2y:z（或 2x:y:2z，或 x:2y:2z）以及 2x:y:z（或 x:y:2z，或 x:2y:z）。这里，x、y 和 z 表示预划分后数据块的维度。

首先，对于立方体节点 d，我们先将其划分为八个八分块，即 $s_1, s_2, \cdots, s_8$（如图11所示），每个块的大小为 $\frac{x}{2} \times \frac{y}{2} \times \frac{z}{2}$。$\bar{x}$、$\bar{y}$ 和 z 表示预划分后数据块的维度大小。然后，我们可以得到这八个八分块中非空单位块的计数，即 $c_1, c_2, \cdots, c_8$。之后，我们将根据这些计数决定沿哪个维度划分立方体节点 d。具体来说，我们可以计算以下内容

三个差异：

$$\begin{aligned}\text{diff}_x &= |c_1 + c_3 + c_5 + c_7 - c_2 - c_4 - c_6 - c_8|,\\ \text{diff}_y &= |c_1 + c_2 + c_5 + c_6 - c_3 - c_4 - c_7 - c_8|,\\ \text{diff}_z &= |c_1 + c_2 + c_3 + c_4 - c_5 - c_6 - c_7 - c_8|.\end{aligned}$$

最后，我们比较这三个值，并选择差值最大的维度进行划分。例如，如果最大差值为 diff_z，我们将沿 z 轴划分 d（即图11中粉色的二维平面），并得到两个扁平节点 d_1 和 d_2。对于诸如 d_1 这样的扁平节点，我们可以复用 c_1, $\cdots, c_4$ 通过比较以下两个差异并选择较大的一个，来确定沿 x 轴还是 y 轴拆分 d_1。

$$\text{diff}_x = |c_1 + c_3 - c_2 - c_4|,\ \text{diff}_y = |c_1 + c_2 - c_3 - c_4|.$$

对于诸如 d_{11} 这样的细长节点，我们只需沿 x 轴将其拆分为两个立方体节点 s_1 和 s_2。此过程（即步骤 (3) 中的立方体节点→扁平节点→细长节点）将循环执行，直到该节点变为叶节点（空或满）。

请注意，根据上述描述，每条路径中的每三个节点都需要进行计数（即仅对“立方体”节点计数）。得益于这种动态拆分方法，我们可以将 AKDTree 算法的时间复杂度降低至 $O(\frac{1}{3} \cdot N \cdot \log N)$，其中 N 是单位块的数量，同时尽可能提取较大的子块，避免包含空单位块。

此外，动态拆分后，我们将得到一系列大小相同但方向不同的子块（例如，2x:2y:z、2x:y:2z、x:2y:2z）。我们将根据拆分维度对大小相同的子块进行对齐（而不是在内存中进行转置），将它们合并成一个数组，并同时压缩多个合并后的数组。

D. 共享霍夫曼编码

如第 I、III-C 和 III-B 节所述，OpST 和 AKDTree 方法会收集具有相同形状的数据块，并将其线性化为一个四维数组后传递给 SZ。然而，正如第 III-B 节所强调的，这些数据块在原始数据集中可能并不连续。这会导致非邻居数据块之间的边界处缺乏局部性或平滑性，从而仍可能降低预测准确率。这种不利影响对于 SZ 的 Lor/Reg算法尤为明显。由于它是一种完全依赖局部数据的局部预测算法，因此当局部平滑性在数据块边界处缺失时，其性能会急剧下降。

一种可行的解决方案是使用 SZ 分别压缩每个数据块。然而，这种方法会导致霍夫曼编码效率低下，因为整个数据集会被划分为许多小数据块，SZ 必须为每个小数据块单独构建一棵霍夫曼树。换言之，原始 SZ 方法要么需要对小数据块进行联合预测和编码（强制将它们合并为四维数组），从而导致预测准确率较低；要么分别预测和编码每个小数据块，这会造成霍夫曼编码开销较高。此外，即使可以将形状相同的数据块一起压缩，形状不同的数据块仍然需要单独压缩

算法 4： 采用 SHE 的 SZ压缩

输入： 多个数据块 $D_{1\ldots n}$
输出： 压缩数据 S

```
量化码 Q，回归系数 R，压缩数据 S;
对于 每个数据块 D_i 执行
    量化码块 q_i，回归系数块 r_i;
    q_i, r_i =SZ.compress(D_i);
    Q.append(q_i);
    R.append(r_i);
结束
S ← HuffmanEncode(Q);
S ← HuffmanEncode(R);          /* 结束压缩 */
返回 S;
```

，导致霍夫曼编码效率低下，而且多次启动 SZ 的时间开销较大。

为此，对于 OpST 和 AKDTree，我们提出了一种共享霍夫曼编码技术：分别预测各数据块，同时使用一棵共享霍夫曼树对它们进行统一编码。具体过程如算法 4 所示。首先分别预测并量化每个数据块。随后，将各数据块的量化码和回归系数汇总起来，构建一棵共享霍夫曼树，并一次性完成编码。这种方法能够显著提升 SZ 的 Lor/Reg 预测性能，同时不会给 SZ 的编码带来很大的时间开销。如图 15 所示，与原始 AKDTree 相比，采用 SHE 的 AKDTree 能够显著降低整体压缩误差，尤其是数据块边界处的数据，从而显著提升峰值信噪比，如图 16 所示。此外，SHE 减少了数据所需的霍夫曼树数量（因为不同形状的数据块不再需要单独的霍夫曼树），进而提高了编码效率和压缩比。

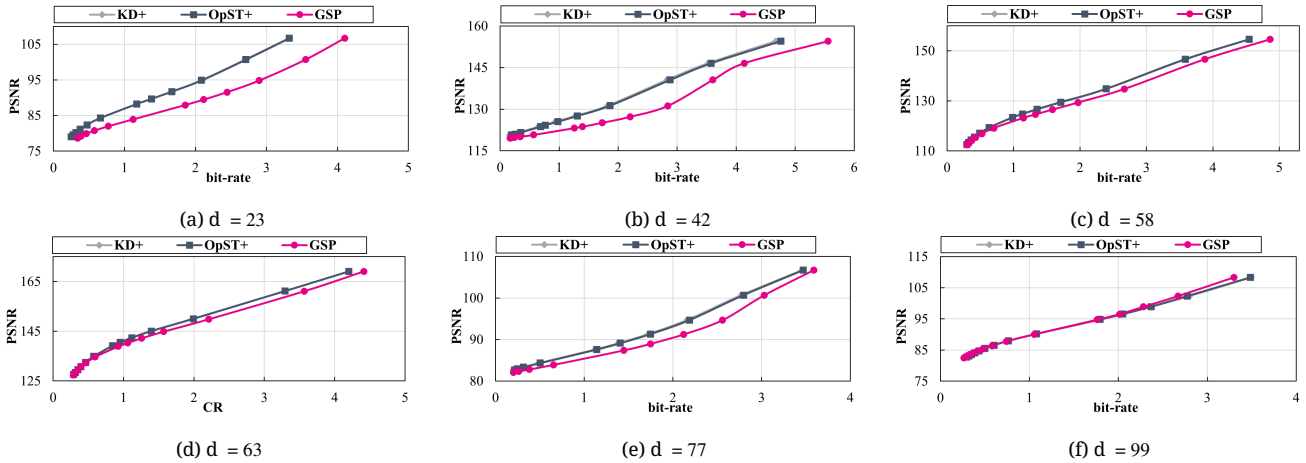
需要注意的是，SZ 的插值无法从 SHE 中获益。与仅使用邻居数据进行预测的 Lor/Reg 不同，Interp 还会考虑数据集中的全局点。然而，采用 SHE 后，OpST 或 AKDTree 生成的每个小数据块都要独立进行预测。这意味着 Interp 会失去所有全局空间信息。相比之下，不采用 SHE 时，原始 OpST 和 AKDTree 方法会将数据块线性化为一个大型四维数组，从而保留更多全局空间信息，以改进预测。因此，我们仅将 SHE 方法应用于 Lor/Reg 预测器。

E. 混合压缩策略

本节提出一种解决方案，用于为 Lor/Reg（结合 SLE）和 Interp 算法自适应地选择最合适的压缩策略。

对于 Lor/Reg（结合 SLE），我们将根据数据特征（即数据密度），从我们提出的采用 *SHE* 的 *OpST*（*OpST+*）、采用 *SHE* 的 *AKDTree*（*AKDTree+*）和 *GSP* 中进行选择。根据第 III-B 节和第 III-C 节，OpST+ 更适用于稀疏（即低密度）数据，而 AKDTree+ 旨在解决数据密度增加时 OpST+ 时间开销较大的问题。因此，应设置一个数据密度阈值，以确定何时使用 OpST+或 AKDTree+。

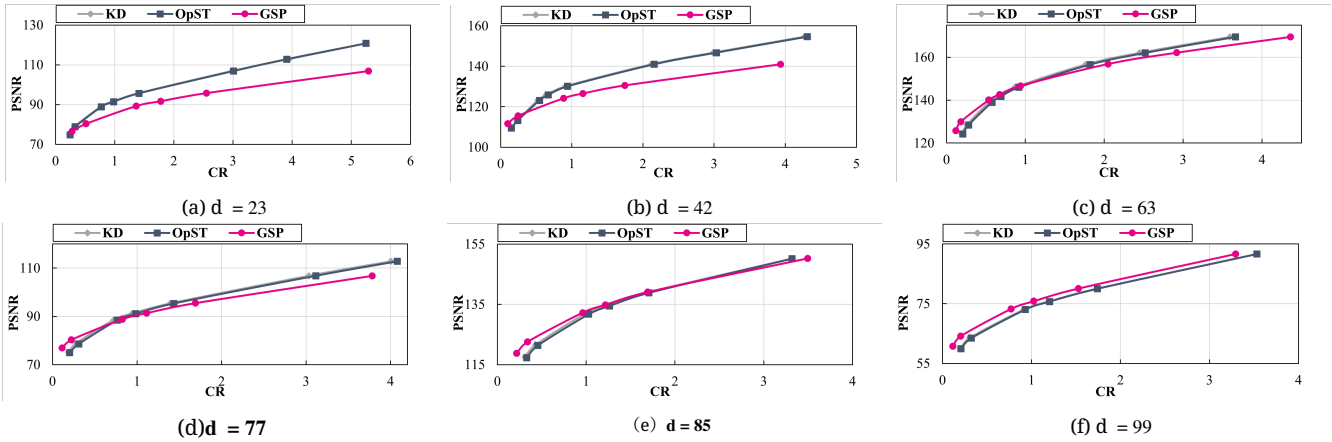
为了确定在 OpST+ 和 AKDTree+之间切换的阈值 T_0，我们使用 Lor/Reg 预测器进行了一系列实验，如图 12 所示。图中显示，OpST+



(a) d = 23 (b) d = 42 (c) d = 58

(d) d = 63 (e) d = 77 (f) d = 99

图 12：使用 Lor/Reg 算法对 6 个不同密度的数据集进行 GSP、OpST+ 和 AKDTree+ 的压缩性能比较。



(a) d = 23 (b) d = 42 (c) d = 63

(d)d = 77 (e) d = 85 (f) d = 99

图 13： 使用 Interp算法在 6 个不同密度的数据集上对 GSP、OpST 和 AKDTree 的压缩性能进行比较 .

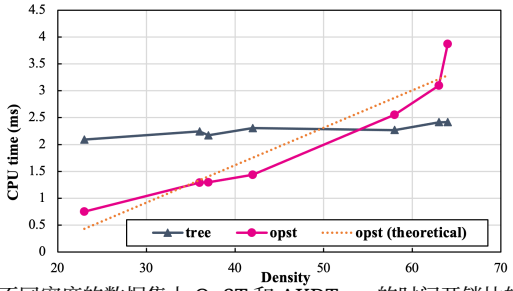
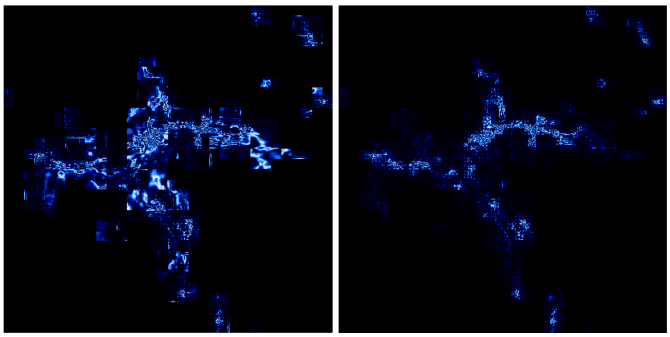


图 14：不同密度的数据集上 OpST 和 AKDTree 的时间开销比较。



(a) AKDtree (压缩率 = 5dB) (b) AKD树 +SHE (压缩率 = 5dB)

图 15：使用 SZ 的 Lor/Reg 预测器对 Nyx 的“重子密度”场（即 z10 的细层级，23% 密度）采用两种方法得到的压缩误差的视觉对比（一个切片）。颜色越亮表示压缩误差越大。误差界为 4.8×10^{-4} 的相对误差界。

和 AKDTree+ 在所有四个具有不同密度的数据集/层级（来自不同时间步）上，就比特率和峰值信噪比而言，压缩性能几乎完全相同。此外，图 14 展示了 OpST+ 和 AKDTree+的时间开销（不包括压缩）。图中表明，AKDTree+ 的耗时相对稳定，而 OpST+的耗时则随数据密度增加而线性增长。总的来说，选择 OpST+还是 AKDTree+ ，唯一的标准是耗时，而非压缩性能。这与我们先前的设计目标一致：AKDTree 主要用于解决 OpST+时间开销过高的问题。由于密度约为 50% 时，OpST+和 AKDTree+的速度相近，因此我们使用 $T_0 = 50$ 来选择 OpST+ 或 AKDTree+。

根据 III-A节，GSP 旨在有效处理数据密度较高的 AMR 层级，以防止

未使用 SHE 进行数据分区会产生负面影响。相比之下，AKDTree 等数据划分方法要求将较小的数据块合并压缩，而不使用 SHE，这会显著降低预测准确率，如第III-D节所示。然而，使用 SHE 压缩分区产生的每个小数据块，就能消除分区对预测准确率造成的负面影响，且开销很小；而 GSP 会显著增加数据大小的开销。如图 12 所示，OpST+和 AKDTree+ 在所有密度下的表现均优于 GSP。因此，采用 SHE 的改进划分策略可以成为适用于所有层级的

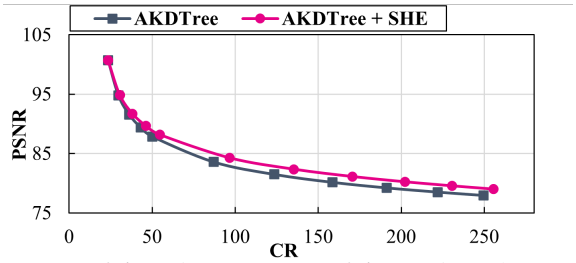


图16：Nyx“重子密度”场（即 z10 的细层级，密度为 23%）上原始 AKDTree 与采用 SHE 的 AKDTree 的比较。

GSP 的替代方案。

总之，对于 Lor/Reg（采用 SLE），我们提出的混合压缩方法如下所述。

1）当密度小于 $T_0 = 50\%$ 时，我们使用 OpST+ 移除空区域，然后进行压缩；2）当密度大于 $T_0 = 50\%$ 时，我们使用 AKDTree+ 移除空区域，然后进行压缩。

对于 Interp算法，我们将根据数据密度，从所提出的方法中进行选择：*OpST*、*AKDTree* 和 *GSP*。值得注意的是，正如第III-D节末尾所述，由于 SHE 方法带来的改进微乎其微，我们不会在 Interp算法中采用该方法。

为了确定 OpST 与 AKDTree 之间切换的合适阈值 T_1，我们进行了一系列实验，如图13所示。这些图表明，对于 Interp算法，在来自不同时间步且密度各异的六个自适应网格细化级别上，OpST 和 AKDTree 的压缩性能也几乎相同。根据这些发现，我们建议在选择 OpST 或 AKDTree 时将 $T_1 = 50\%$ 设为阈值，这与我们对 Lor/Reg 的处理方式一致，旨在提升处理速度。

接下来，为了确定 AKDTree 与 GSP 之间切换的阈值 T_2，我们也在不同密度的数据集上对它们进行了评估。如图13所示，密度相对较低时，AKDTree 在比特率和峰值信噪比方面均优于 GSP；随着密度不断升高，GSP 逐渐优于 AKDTree。我们还发现，当密度约为 85% 时，AKDTree 与 GSP 的压缩性能相近。因此，我们使用 $T_2 = 85\%$ 来选择 AKDTree 或 GSP。

因此，我们针对 Interp 算法提出的混合压缩方法如下：

1）当密度小于 T_1 时，我们将使用 OpST 在压缩前移除空区域；2）当密度介于 T_1 和 T_2 之间时，我们将使用 AKDTree 在压缩前移除空区域；3）当密度大于 T_2 时，我们将使用 GSP 在压缩前填充适当的值。

显然，与采用 SHE 的 Lor/Reg 相比，GSP 再次证明了其对 Interp 的益处。核心原因是，对于 Lor/Reg，我们可以像刚才提到的那样，使用 SHE 完全消除分区带来的影响。分区不可避免地会破坏 Interp 的全局空间信息/局部性。因此，可以将 GSP 应用于数据密度较高的 AMR 层级，以抵消分区的影响，同时尽量减少数据大小开销。

F. 关于通用性与分布的讨论

我们设计的通用性。 TAC 和 TAC+ （包括 SZ 的 Lor/Reg 所采用的预处理策略和 SHE）旨在逐层处理 AMR 数据，其中数据分布不规则，例如包含许多空区域/零值区域。TAC/TAC+也可用于稀疏非 AMR 数据压缩。不过，对于密度极高的稀疏非 AMR 数据（例如，99.9% 的数据均为非零），我们为不规则数据设计的 TAC/TAC+ 可能无法带来太多优势，因为这类数据与没有空区域的普通非稀疏数据非常相似。另一方面，TAC/TAC+ 更适用于 AMR 数据，因为 AMR 数据具有多分辨率层级，且所有层级的密度总和为 1。这意味着至多只有一个超高密度层级，而在该层级上，TAC/TAC+ 可能效果较差。

此外，SHE 方法可以在保持预测性能的同时提高压缩比，尤其是在对大量小尺寸数据集进行整体压缩时。例如，压缩 10,000 张尺寸为 100×100 的二维图像。若单独压缩这些图像，整体压缩比（CR）会降低。相反，将它们堆叠成一个三维块也并非最佳选择，因为图像之间的不平滑可能会损害预测准确率。

分布式场景。“在分布式环境中，并行化 GSP 相对简单，因为其填充方法主要是局部的。唯一的小挑战出现在需要从其他进程计算填充值时。这需要交换边界值，类似于 Stencil 问题。

然而，由于 OpST 和 AKDTree 具有全局性，它们在并行化方面面临着更大的挑战。仅对这些算法采用易于并行化的方法，会导致数据块比串行版本更小，空间局部性也更差。显然，要让这些算法发挥更大效用，就需要进行通信；但如何在从数据集中提取较大子块的能力与通信开销之间取得平衡，将是一项挑战。解决这一挑战将是我们未来工作的主要目标。

IV. 实验评估

A. 实验设置

测试数据。 我们的评估主要针对 AMReX 框架 [3],，尤其是 Nyx 宇宙学模拟 [9],、WarpX 电磁与静电粒子网格（PIC）模拟 [29],，以及用于求解不可压缩 Navier–Stokes 方程的 IAMR [30], 。

如图 17 所示，Nyx 是一款先进的超大规模宇宙学代码，利用了 AMReX。它生成六个场，包括重子密度、暗物质密度、温度和速度（x、 y 和 z）。我们分析了由三次真实模拟运行生成的六个数据集，每次运行的 AMR 层级数不同，模拟区域均为 64 兆秒差距（Mpc）。对于这些数据集，Z 表示红移，即遥远星系和天体的位移，如表 I 所示。具体而言：第一次运行包含两个细化层级，粗层级包含 256^3

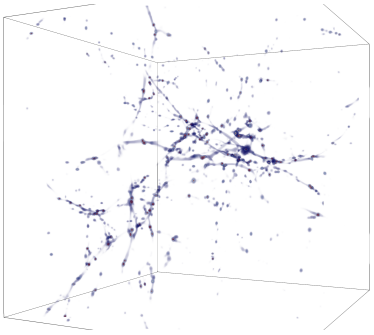


图 17：Nyx 密度场的可视化。

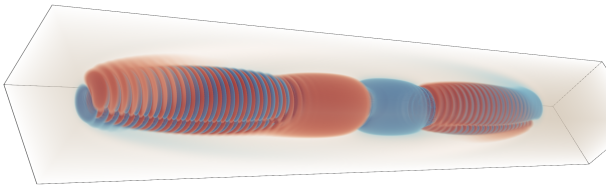


图 18：WarpX 电场（z 方向）的可视化 n).

个网格，细层级包含 512^3 个网格。我们收集了三个时间步的数据，此时最精细层的密度范围为 23% 至 63%。第二次运行最多包含四个细化层级。其初始分辨率为 128^3，之后细化至 1024^3。我们从这次运行中获取了两个时间步的数据，其最精细层分辨率分别为 512^3（三个层级）和 1024^3（四个层级）。最精细层的密度范围为 0.2% 至 0.003%。第三次运行包含三个细化层级，最粗层级、中间层级和最精细层的网格尺寸分别为 128^3、256^3 和 512^3。最精细层的密度为 0.90%。

如图 18 所示，WarpX 是一款高度并行且经过优化的电磁与静电粒子网格（PIC）模拟软件，集成了 AMReX。它专为 GPU 和多核 CPU 运行而设计，并具备负载均衡功能。WarpX 能够扩展至全球最先进超级计算机的计算能力。值得一提的是，它荣获了 2022 年 ACM 戈登·贝尔奖。我们从 WarpX 收集了两个时间步的数据，详见表 I。每个时间步包含两个细化层级，维度为 $128\times128\times1024$ 和 $256\times256\times2048$。最精细层级的密度在 2% 到 8.6% 之间变化。

如图 19 所示，IAMR 是一款高度并行的自适应网格细化代码，用于求解二维或三维的变密度不可压缩纳维–斯托克斯方程。它还提供嵌入边界（切割单元）表示，以处理复杂几何形状。我们专门运行了三维瑞利–泰勒问题，该问题模拟重力作用下重流体位于轻流体上方的情形。我们从 IAMR 中收集了两个时间步的数据。每个

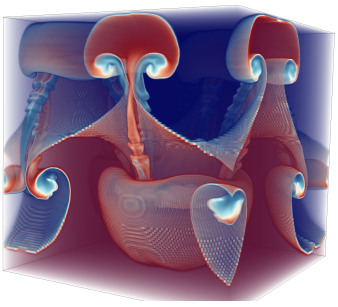


图 19：IAMR 密度场的可视化。

表 I：我们测试的自适应网格细化应用和数据集。

数据集	层数	各层网格尺寸（从细到粗）	各层密度（从细到粗）	数据大小（每个时间步）
Nyx Run1_Z10	2	512、256	23%、77%	3.3 GB
Nyx Run1_Z5	2	512、256	58%、42%	6.4 GB
Nyx Run1_Z2	2	512、256	63%、37%	6.7 GB
Nyx Run2_T3	3	512、256、128	0.02%、0.56%、99.42%	169 MB
Nyx Run2_T4	4	1024、512、256、128	3E-5、0.02%、2.2%、97.7%	190 MB
Nyx Run3_Z1	3	512, 256, 128	0.90%、14.70%、84.40%	416 MB
WarpX_800	2	128*1024; 256*2048	8.6%,91.4%	2 GB
WarpX_1600	2	128*1024; 256*2048	2.0%、98.0%	1.4 GB
IAMR_90	3	512, 256, 128	0.6%、10.5%、88.9%	336 MB
IAMR_150	3	512, 256, 128	14.8%、30.9%、54.3%	2 GB

时间步包含三个细化层级，从最粗层级的维度 $128\times128\times128$ 开始，逐级细化至最精细层的 $512\times512\times512$。最精细层的密度范围为 0.6% 至 14.8%。

请注意，最精细层的密度描述了数据集中有多少数据处于最高分辨率；最精细层的密度越高，表示精细化到最高分辨率的数据越多。通常，在单次运行中，最精细层的数据密度会逐渐增加。

评估平台和压缩器。测试平台配备两颗 28 核英特尔至强金牌 6238R 处理器和 384 GB 内存。本研究采用两种不同的 SZ 压缩算法，即 Lor/Reg 和 Interp。Lor/Reg 使用 Lorenzo 和线性回归预测器，Interp 则采用第 II-A 节所述的样条插值方法。Lor/Reg算法首先将整个输入数据划分为 $6\times6\times6$ 数据块，然后在每个数据块上独立应用 Lorenzo 预测器或高维线性回归。相反，Interp算法会对整个数据集的全部三个维度进行插值。

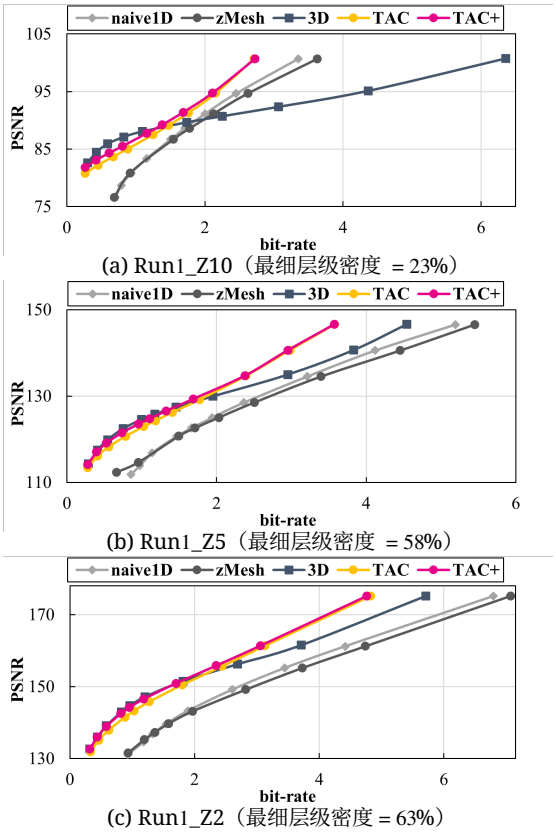
Lor/Reg 与 Interp 的一个关键区别在于，前者基于数据块，而后者是全局的。具体而言，Lor/Reg算法在压缩之前将数据划分为数据块，而 Interp算法则对整个数据集进行全局插值。

比较基线方法。如第二节所述，我们将方法与三种基线方法进行比较，这些方法采用一维或三维方式，并介绍两种我们提出的方案。具体而言，(1) 一维基线（朴素）：将每个自适应网格细化层级分别压缩为一维数组；(2) 一维基线（*zMesh*）[28]：关于 zMesh 方法如何重组自适应网格细化数据以进行一维压缩，详见第二节；(3)三维基线：将不同的自适应网格细化层级统一到相同分辨率，以进行三维压缩；(4)我们提出的 TAC：根据数据密度，使用 OpST（不含 SHE）、AKDTree（不含 SHE）和 GSP 对每个自适应网格细化层级进行压缩。更多信息请参阅 [45]。(5) 我们提出的 TAC+：根据数据密度，使用 OpST+ 或 AKDTree+压缩每个自适应网格细化层级，并通过 Lor/Reg 算法使用 SHE。

如第III-D节所述，TAC+ 专门与Lor/Reg算法配套使用。这是因为TAC+ 旨在利用SHE提升Lor/Reg压缩算法对AMR数据的处理性能。另一方面，TAC作为一种预处理技术，兼容Lor/Reg和Interp算法。

B. 评估指标

我们将根据以下指标评估压缩性能：(1) 压缩比或比特率（通用），

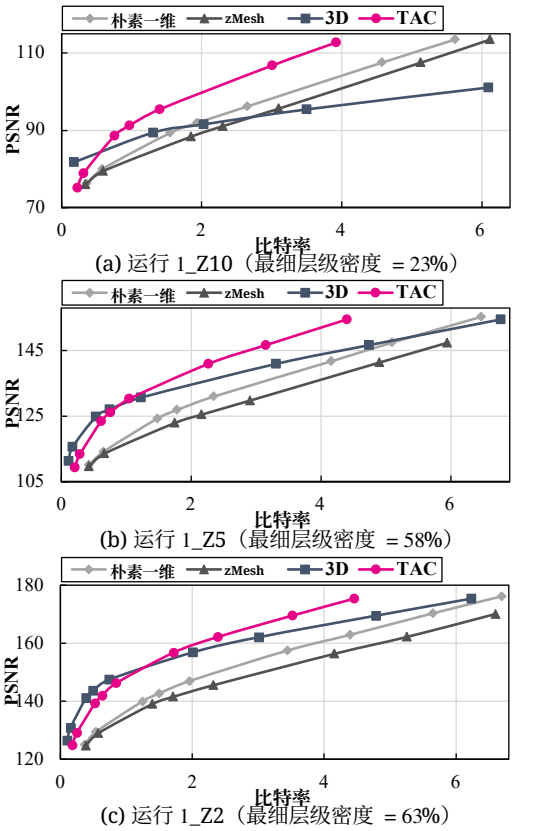


(a) Run1_Z10（最细层级密度 = 23%）

(b) Run1_Z5（最细层级密度 = 58%）

(c) Run1_Z2（最细层级密度 = 63%）

图 20：使用 Lor/Reg 算法，对 Nyx run1 从早期时间步（Z10）到晚期时间步（Z2）的数据进行率失真比较，比较我们的方法（TAC 和 TAC+）与基线方法。



(a) 运行 1_Z10（最细层级密度 = 23%）

(b) 运行 1_Z5（最细层级密度 = 58%）

(c) 运行 1_Z2（最细层级密度 = 63%）

图21：使用 Interp 算法时，TAC 与基线方法在 Nyx run1 上的率失真比较。

(2) 失真质量（通用），(3) 压缩吞吐量（通用），(4) 率失真（通用），(5) 功率谱（宇宙学专用），(6) 晕团搜寻器（宇宙学专用）。

指标 1： 为评估压缩带来的数据大小缩减，我们使用压缩比，即原始数据大小除以压缩数据大小；也可以使用比特率（比特/值），表示每个值的摊销存储成本。单精度/双精度浮点数据在压缩前的比特率为每个值 32/64 比特。压缩比与比特率之间存在数学关系，两者的乘积为 32/64，因此比特率越低，压缩比越高。

指标 2： 失真是评估有损压缩质量的另一项重要指标。我们使用峰值信噪比（PSNR）来衡量失真质量。

$$\mathrm{PSNR} = 20 \cdot \log_{10}(R_X) - 10 \cdot \log_{10}\left(\sum_{i=1}^{N} e_i^2 / N\right),$$

其中，e_i 是点 i处原始值与解压值之间的差异，N 是点的数量，R_X 是 X的取值范围。PSNR 越高，误差越小。

指标 3： 压缩和解压缩吞吐量对提升输入/输出性能至关重要。我们根据原始数据大小和压缩（解压缩）时间计算吞吐量。

指标 4： 与以往工作类似 [14], [15], [21], [35], [39], [46], [47],，我们绘制率失真曲线，以比较相同比特率下的失真质量，从而公平地比较不同压缩方法，并将各种压缩算法纳入考量。

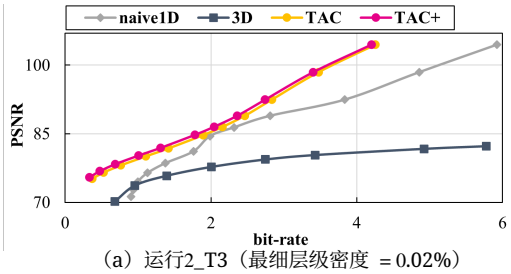
指标5： 宇宙中的物质分布不断演化，在不同物理尺度上形成天体物理结构，从行星到超星系团和星系丝等更大结构。二点相关函数 $\xi(r)$ 描述了在距离另一星系 r 一定距离处发现一个星系的超额概率，可在统计上描述宇宙在各个物理尺度上的物质分布量。$\xi(r)$ 的傅里叶变换称为物质功率谱 $P(k)$，其中 k 为共动波数。物质功率谱描述了各个物理尺度上存在多少结构。我们使用名为 Gimlet 的宇宙学分析工具，对重子密度场计算功率谱。我们比较解压缩数据的功率谱 $p'(k)$ 与原始数据的功率谱 $p(k)$，并要求所有 $k < 10$ 的最大相对误差不超过 1%。

指标6： 晕团搜寻器旨在从暗物质分布中寻找晕（过密区域），并分别输出所找到的每个晕的位置、网格单元数量和质量。具体而言，晕团搜寻算法 [48] 会从所有模拟数据中搜寻晕，并采用以下两个判据：（1）数据点的质量必须大于某个阈值（例如，整个数据集平均质量的 81.66×）才能成为晕团网格候选单元 [21], [47], [49], ；（2）某个区域内必须有足够多的晕团网格候选单元才能形成一个晕。对于解压缩数据，部分信息（晕的质量和网格单元数量）可能会偏离原始数据。

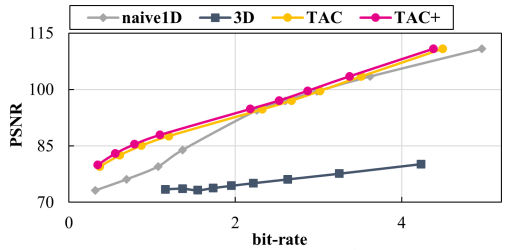
C. 率失真评估

我们首先评估所提出的压缩方法的率失真表现，并在不同模拟和压缩算法上将其与基线方法进行比较。

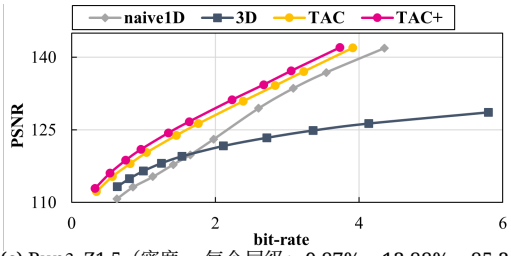
对于 Lor/Reg 算法，如图20、22和24所示，我们采用 SHE 的新方法 TAC+ （以粉色



（a）运行2_T3（最细层级密度 = 0.02%）



(b) Run2_T4（最细层级密度 = 3E-5）



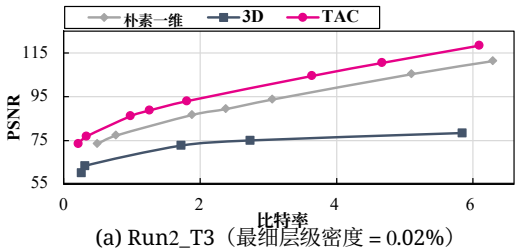
(c) Run3_Z1.5（密度 y 每个层级：0.87%、13.88%、85.25%）

图 22：使用 Lor/Reg算法，对 run2 和 run3 在不同时间步上的我们的方法与基线方法进行率失真比较。

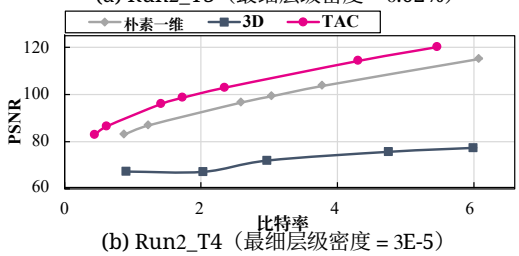
曲线表示）在 Nyx 和 IAMR 的所有数据集上都比不采用 SHE 的原始 TAC（黄色曲线）性能更好。然而，如图26所示，在 WarpX 中，TAC+ 并未优于 TAC。这可归因于 WarpX 的细化结构较为简单。与细化数百个区域的 Nyx 和 IAMR（如图 4a 所示）不同，WarpX 只细化一个大型矩形区域。因此，WarpX 的单位块大小是 Nyx 和 IAMR 的四倍。经过 TAC+ 分区后，单位块的数量大幅减少。这一减少抵消了 TAC+的优势；TAC+专为高效处理大量小型 AMR 数据块而开发。

此外，如图20至27所示，对于一维基线，在三个模拟的全部十个数据集以及两种压缩算法（即 Lor/Reg 和 Interp）下，我们的方法（包括 TAC+ 和 TAC）均优于一维基线。此外，我们的方法性能比一维基线更加稳定（即曲线更平滑）。我们还发现，在测试的数据上，zMesh略逊于一维基线，如图20和21所示，下一节将对此进行解释。

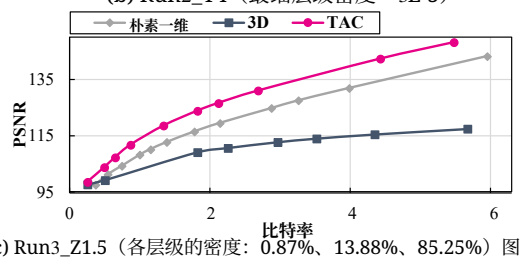
对于 3D 基线，我们观察到，如图20至图27所示，在所有数据集和压缩算法中，当最精细层的密度相对较低，或解压缩数据具有较高的峰值信噪比时，我们的方案性能都要好得多。然而，如图20和图21所示，当满足以下条件时，我们的方法无法全面优于 3D 基线：（1）AMR 数据仅有两个细化层级；（2）最精细层具有相对较高的密度；（3）解压缩数据具有较低的峰值信噪比/比特率。



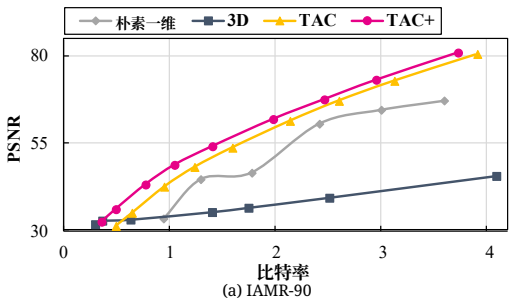
(a) Run2_T3（最细层级密度 = 0.02%）



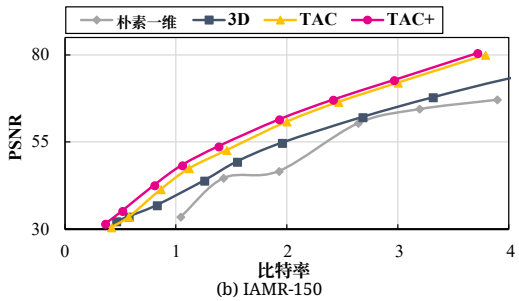
(b) Run2_T4（最细层级密度 = 3E-5）



(c) Run3_Z1.5（各层级的密度：0.87%、13.88%、85.25%）图 23：使用 Interp算法在 Nyx run2 和 run3 上对 TAC 与基线方法进行的率失真比较。



(a) IAMR-90



(b) IAMR-150

图 24：使用 Lor/Reg算法对瑞利-泰勒不同时间步上的方法与基线方法进行的率失真比较。

下一节将详细讨论 AMR 层级数量、最精细层的密度，以及解压缩数据的比特率/峰值信噪比如何影响 3D 基线、TAC+ 和 TAC 的性能。

D. 与基线方法的比较

在压缩时，zMesh 旨在利用各 AMR 层级之间的数据冗余（如引言所述），提升基于块的 AMR 数据集的平滑度。因此，如果树结构自适应网格细化数据（即我们测试的数据集）中不存在数据冗余，zMesh 就无法提升平滑度。图28b 展示了一个简单示例来说明这一点

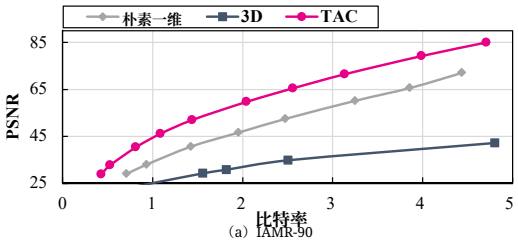


图25：使用 Interp算法对瑞利-泰勒模拟中不同时间步的 TAC（左上）和基线方法进行率失真比较。

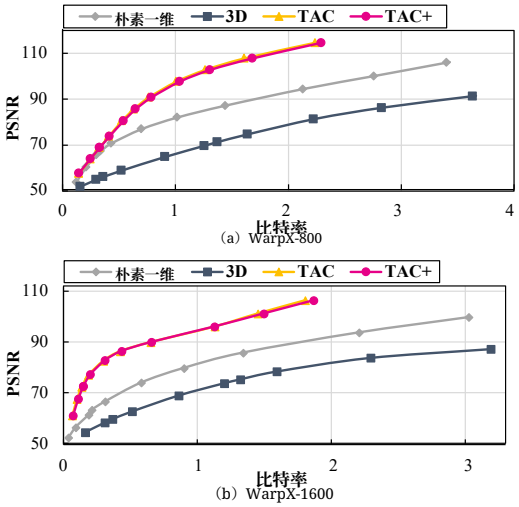


图26：使用 Lor/Reg 算法，对 WarpX 不同时间步上的我们的方法与基线方法进行率失真比较。

：较细层级的数据值更高，因为只有当网格的值超过某个阈值时，才会对其进行细化。对于基于块的自适应网格细化，当网格因值较高而需要细化时，该值仍会保留在该层级中，从而产生一个冗余值（即红色的 8）。如果采用原始的 Z 序排序，逐层遍历数据（如图28b所示），重排后的数据将出现三次显著的数值变化（即从 2 变为 8、从 8 变为 1，以及从 1 变为 9）。为解决这一问题，zMesh 根据二维数组的布局，同时遍历两个 AMR 层级。重排后的数据为“1-2-8-9-8-7-8-1”，仅出现两次数值显著变化（从 2 变为 8，以及从 8 变为 1）。因此，zMesh 可以提升基于块的 AMR 数据的平滑度。

然而，如图28a所示，对于树结构自适应网格细化数据（不保存冗余的“8”），与逐层单独压缩的一维基线相比，zMesh 在遍历两个自适应网格细化层级时会引入两处显著的数据变化（即从 2 变为 9，以及从 8 变为 1）。这解释了为什么 zMesh 略逊于一维基线。

考虑三维基线方法时，我们观察到，在以下情况下，它的效果略优于 TAC+：（1）AMR 数据只有两个细化层级；（2）数据的最精细层具有相对较高的密度；

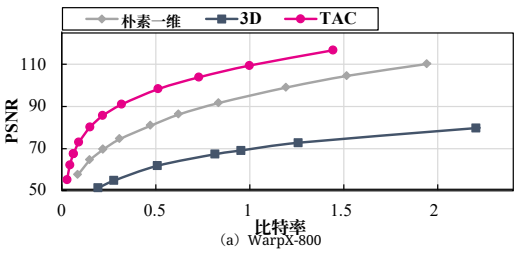
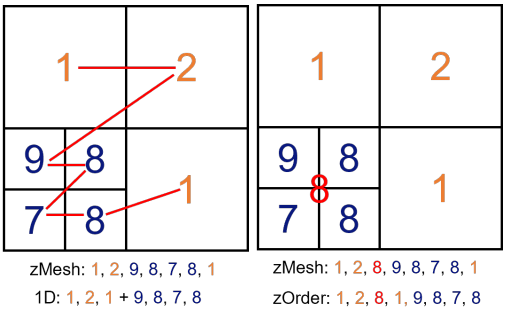


图27：使用 Interp 算法，对 WarpX 不同时间步上的 TAC+ 与基线方法进行率失真比较。



（a）基于树的AMR数据（b）基于块的AMR数据 图28：一个示例，展示一维基线、zMesh 和原始 z 序如何对简单的二维AMR数据进行重排，无论数据是否包含冗余。橙色：粗层级；蓝色：细层级；红色：冗余数据。

以及（3）解压缩数据具有较低的峰值信噪比/比特率。

现在，我们逐一解释各个观察点。如第 II-D 节所述，3D 基线的一个主要缺点是上采样过程产生了冗余数据。对于具有两个以上细化层级的 AMR 数据，这种冗余会更加明显，尤其是在最精细层的数据较为稀疏时。这些额外数据会增大数据大小，导致 3D 基线的压缩性能不够理想（观察点 1 和 2）。

相较于 TAC+ 和 TAC，3D 基线的数据局部性更好，平滑度也更高。这是因为它将整个数据集作为一个整体进行处理，而不对其进行分段。因此，与 TAC+ 和 TAC 相比，3D 基线具有更强的可压缩性；随着误差界增大，其比特率下降得更快，而峰值信噪比下降得更慢。因此，3D 基线在低比特率下表现更好（观察点 3）。

E. TAC/TAC+ 采用不同压缩算法

请注意，Lor/Reg压缩算法可以使用增强型TAC+，从而提高压缩性能。相反，对于Interp，我们没有使用TAC+，因为如第III-D节所述，它无法相较于TAC带来性能优势。这种差异的根本原因在于AMR 数据和

Lor/Reg算法的性质。AMR 数据和Lor/Reg算法都是基于块的，而Interp是全局的。

具体而言，压缩高维自适应网格细化数据的一大挑战在于需要进行分区，这可能会损害空间信息和数据平滑性。鉴于 Lor/Reg 也会将数据划分为数据块，因此它与自适应网格细化数据固有的分区方式天然契合。通过利用我们的预处理和优化技术，结合使用 SLE 的 TAC+ 的 Lor/Reg 能够有效解决由分区导致的局部数据平滑性和平滑性问题，因此尤其适用于自适应网格细化数据。相比之下，Interp 方法对分区后的自适应网格细化数据采用全局插值方法。由于分区，较小的数据块不可避免地会破坏全局插值所需的全局空间信息。虽然我们可以使用 OpST 和 AKDTree 来获得更大的分区数据块并提高空间局部性，但仍无法使插值预测与自适应网格细化数据实现完美兼容。

F. 自适应误差界下的后分析质量

表II：采用不同方法进行的晕团搜寻器分析。

	CR	平均相对质量差	平均相对单元差
3D 基线	188.7	2.7E-04	2.2E-03
TAC+ (1:1)	189.1	2.2E-04	2.1E-03
TAC+ (2:1)	**192.5**	**1.1E-04**	**9.0E-04**

在考虑逐层压缩时，我们的方法可以依据（1）后续分析指标、（2）粗层级的上采样率，以及（3）不同 AMR 层级之间的率失真权衡，为不同的 AMR 层级应用不同的误差界。现在，我们使用两种宇宙学专用的后续分析指标（即功率谱和晕团搜寻器）评估我们的方法，以展示自适应误差界方法的优势。我们选择数据集 run1-Z2 进行评估，因为在该数据集上，TAC+ 的性能与 3D 基线相近。

图 29 展示了在不同 AMR 层级之间进行率失真权衡的动机。随着细层和粗层的误差界增大，它们的比特率将趋于相近。这意味着，当误差界相对较大时，与压缩误差的增加相比，数据大小的减少并不显著（即两条曲线的斜率都较小）。因此，我们得出结论：当误差较大时，以数据质量换取大小缩减并不值得。

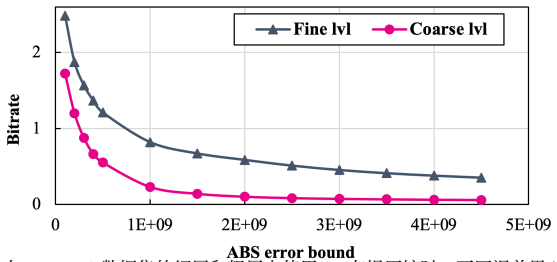


图29：在 Run1 Z2 数据集的细层和粗层上使用 SZ 有损压缩时，不同误差界对应的比特率。_

功率谱 图 30a 显示，在压缩比（几乎）相同的情况下，TAC+ （采用均匀误差界）的功率谱误差优于三维基线。

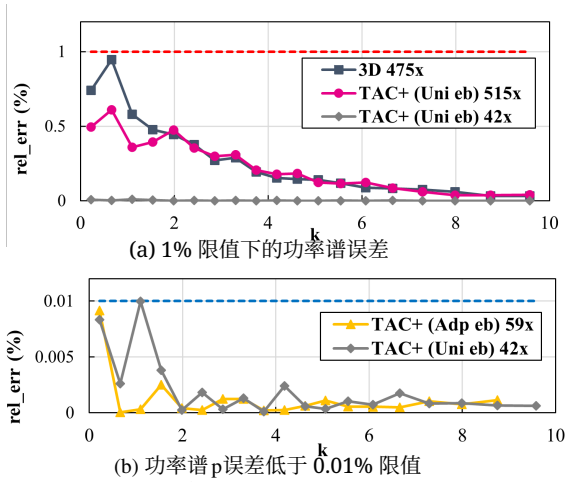


图30：run1-Z2 重子密度场上，3D 基线、TAC+（均匀误差界）和 TAC+ （自适应误差界）的功率谱相对误差。红色和蓝色虚线分别表示可接受功率谱误差的 1% 和 0.01% 限值。

还请注意，如图 30a 中的灰色曲线所示，在压缩比为42×时，TAC+ 造成的功率谱失真几乎可以忽略（低于 0.01%）。

现在，让我们按照本节开头提到的三个步骤，调整每个 AMR 层级的误差界。首先，需要在均匀分辨率数据上运行后分析指标——功率谱——以评估数据的全局质量。因此，对于均匀分辨率数据，细层和粗层的理想误差界配置/比例应为 1:1。

如前所述，AMR数据集的粗层级需要上采样，以统一分辨率。因此，粗层级的压缩误差也会随之放大，导致后续分析中的误差增大。因此，我们需要根据上采样率为粗层级设置更小的误差界。此处，Z2 粗层级的上采样率为2^3, , 使细层和粗层的理想误差界比变为 8:1。

最后，还需要根据前文所述的率失真权衡对这一 8:1 的比例进行调整。如图 29 所示，采用 8:1 的误差界比（例如，细层级为 4E+9，粗层级为 5E+8 ）时，细层级的误差界过大，导致率失真权衡效果不佳。因此，我们可以通过增大粗层级的误差界（以提高压缩比）并减小细层级的误差界（以增加压缩误差），来平衡两个层级，从而整体改善率失真表现。根据我们的实验，我们将误差界比从 8:1 调整为 3:1。如图30b所示，在功率谱误差相近的情况下，与均匀误差界相比，采用自适应误差界的 TAC+ 能够显著提高压缩比。

晕精细化 我们评估了使用 3D 基线、TAC+ （采用均匀误差界）和 TAC+（采用自适应误差界）识别出的三个最大晕的质量变化和单元数量变化，如表II所示。可以看出，与 3D 基线相比，采用均匀误差界的 TAC+ 能够获得更好的晕精细化分析质量。

与针对功率谱进行的误差界配置分析类似，

表 III：总体 Nyx 上采用不同方法并设置不同绝对误差界时的压缩/解压吞吐量（MB/s）

Comp Algo	EB_abs	运行1_z10				Run1 Z2_				运行2 T4_				运行3_Z1			
		一维	三维	TAC	TAC+	1D	3D	TAC	TAC+	1D	3D	TAC	TAC+	1D	三维	TAC	TAC+
Lor/Reg	1E+08	51	37	79	77	51	79	85	84	49	0.32	24	24	49	5.6	63	63
	1E+09	81	48	94	93	55	92	103	97	53	0.39	26	26	53	6.5	78	75
	1E+10	89	53	107	102	58	100	111	104	84	0.41	28	27	57	7.1	91	84
Interp	1E+08	74	35	80	\	73	73	90	\	67	0.27	24	\	68	4.9	70	\
	1E+09	82	42	91	\	82	81	95	\	77	0.32	25	\	78	5.8	77	\
	1E+10	90	46	97	\	87	85	104	\	84	0.33	26	\	87	6.3	87	\

表 IV：在 IAMR 上使用不同绝对误差界时，各种方法的总体压缩/解压吞吐量（MB/s）。

Comp Algo	EB_abs	IAMR 90_				IAMR_150			
		1D	3D	TAC	TAC+	1D	3D	TAC	TAC+
Lor/Reg	2E+03	50	5.0	67	66	51	28	76	78
	2E+04	52	5.2	73	71	50	30	85	86
	2E+05	54	5.6	79	78	53	31	94	94
Interp	2E+03	72	4.6	68	\	68	25	81	\
	2E+04	80	4.8	75	\	78	27	92	\
	2E+05	86	5.1	80	\	81	28	96	\

表 V：WarpX 上不同方法在不同绝对误差界下的总体压缩/解压吞吐量（MB/s）。

Comp Algo	EB_abs	WarpX_800				WarpX_1600			
		一维	三维	TAC	TAC+	1D	三维	TAC	TAC+
Lor/Reg	1E+06	49	15	97	85	47	8.9	102	89
	1E+07	52	18	107	92	50	11	126	103
	1E+08	52	19	115	98	53	14	159	121
Interp	1E+06	73	14	128	\	68	8.7	123	\
	1E+07	79	16	140	\	75	11	140	\
	1E+08	83	17	147	\	79	12	159	\

现在让我们调整晕团搜寻器中细层和粗层之间的误差界比。晕团搜寻器分析也需要以均匀分辨率数据作为输入。不过，与功率谱分析不同，晕团搜寻器分析更关注细层级中的高值点，因为只有高值数据点才符合晕团候选对象的条件，如第四节-B所述。请注意，这并不意味着我们可以直接丢弃值较小的粗层级数据，因为这些数据仍会影响数据集的平均值，而平均值也是晕团搜寻器的一个重要参数 [48]。因此，基于大量实验，我们将均匀分辨率数据的理想误差界比设为 1:2（即细层级与粗层级之比）。之后，考虑到 2^3，的上采样率，误差界比变为 4:1。最后，我们根据率失真权衡将该比值调整为 2:1。表II显示，采用自适应误差界的 TAC+ 在质量和单元数量方面的差异最小。

G. 时间开销评估

我们在具有不同误差界的数据集上评估总体吞吐量（包括预处理和（解）压缩）。如表 III、IV 和 V 所示，与 3D 基线相比，TAC+ 在 Nyx Run1 数据集上的吞吐量最高提升 $2\times$，在 Nyx Run2 数据集上提升 $75\times$，在 Nyx Run1 数据集上提升 $11.8\times$，在 IAMR 数据集上提升 $10\times$，在 WarpX 数据集上提升 $14\times$。这是因为 WarpX、IAMR、Nyx Run2 和 Nyx Run3 数据集在最精细层的密度低于 Run1 数据集，导致 3D 基线的冗余数据开销更高，这与我们在第 IV-D 节中的讨论一致。我们注意到，TAC+ 的表现优于所有测试数据集上的一维基线，唯独 Nyx Run2_T4 数据集除外，这是由于在较小数据集上，其数据分区时间（相对于总时间）较长。此外，三张表中的符号“\”表示，如第 IV-A 节所述，TAC+ 专门与 Lor/Reg 算法配对。

TAC+ 和 TAC 的总体吞吐量较高，这可归因于三种预处理策略的高效率。首先，GSP 预处理的时间复杂度为线性，因为只需处理一部分边界数据一次，即可计算出填充值。此外，填充仅限于特定的空区域，且只执行一次。其次，OpSt(+) 和 AKDTree(+) 的时间复杂度分别为$O(N^2 \cdot d)$ 和 $O(\frac{1}{3}N \cdot \log N)$，其中 N表示单位块数量，d 表示密度。鉴于每个单位块包含大量数据点，OpSt(+) 和 AKDTree(+) 都具有很高的效率。此外，通过采用第 III-E 节详述的混合压缩策略，我们可以在 OpSt(+) 和 AKDTree(+) 中选择速度更快的算法，进一步提升 TAC(+) 的性能。

此外，TAC+ 的吞吐量与 TAC 几乎相同。吞吐量略有下降，是因为 TAC+ 以更细粒度的方式压缩数据。在此评估中，我们排除了 zMesh，因为从理论上讲，由于额外的 Z 序排序，其速度比一维基线更慢；此外，根据我们的评估结果，其率失真表现也更差。

V. 结论与未来工作

总之，本文从系统层面利用 3D 压缩处理 AMR 数据。我们提出了三种预处理策略，可根据每个 AMR 层级的密度进行自适应调整。在相同数据质量损失下，与最先进的方法相比，我们的方法最多可将压缩比提高 4.9× 。借助逐层压缩方法，在相同压缩比下，我们能够将细层和粗层的误差界比分别调整为 3:1 和 2:1，从而分别改善功率谱和晕查找器分析。在未来的工作中，我们将把 TAC+ 应用于更多 AMR 模拟，并使用 OpenMP 在多核 CPU 上以及使用 CUDA 在 GPU 上提高其吞吐量。

致谢

本研究由 Triad National Security, LLC 的员工撰写。该公司根据与美国能源部（DOE）和国家核安全管理局（NNSA）签订的第 89233218CNA000001 号合同运营洛斯阿拉莫斯国家实验室。本研究得到百亿亿次计算项目（ECP）的支持，项目编号：17-SC-20-SC；该项目由美国能源部科学办公室和国家核安全管理局共同开展。本研究还得到美国国家科学基金会（NSF）项目 2303064、2247080、2311876 和 2312673 的资助。本研究使用了国家能源研究科学计算中心的资源。该中心是美国能源部科学办公室的用户设施，位于劳伦斯伯克利国家实验室，并根据第 DE-AC02-05CH11231 号合同运营。我们感谢劳伦斯伯克利国家实验室 NYX团队的 Zarija Luki´c 博士和 Jean Sexton 博士授权我们使用宇宙学数据集。本研究使用了开源粒子网格（PIC）代码 WarpX，该代码主要由美国能源部百亿亿次计算项目资助。

参考文献

[1] C. Burstedde、O. Ghattas、G. Stadler、T. Tu 和 L. C. Wilcox，“面向拍级计算机的自适应网格偏微分方程模拟”，Teragrid 会议论文集，第 8 卷，2008年。[2] A. Dubey、A. Almgren、J. Bell、M. Berzins、S. Brandt、G. Bryan、P. Colella、D. Graves、M. Lijewski、F. Löffler 等，“块结构自适应网格细化软件包中高级框架综述”，《并行与分布式计算杂志》，第 74 卷，第 12 期，第 3217–3227 页，2014年。[3] W. Zhang、A. Almgren、V. Beckner、J. Bell、J. Blaschke、C. Chan、M. Day、B. Friesen、K. Gott、D. Graves 等，“AMReX：一种用于块结构自适应网格细化的框架”，《开源软件杂志》，第 4 卷，第 37 期，第 1370–1370 页，2019年。[4] J. M. Stone、K. Tomida、C. J. White 和 K. G. Felker，“Athena++自适应网格细化框架：设计与磁流体力学求解器”，《天体物理学杂志增刊系列》，第 249 卷，第 1 期，第 4 页，2020年。[5] A. S. Almgren、J. B. Bell、M. J. Lijewski、Z. Luki´c 和 E. Van Andel，“Nyx：一种用于计算宇宙学的大规模并行自适应网格细化代码”，《天体物理学杂志》，第 765 卷，第 1 期，第 39 页，2013年。[6] B. Runnels、V. Agrawal、W. Zhang 和 A. Almgren，“利用块结构自适应网格细化和几何多重网格求解器实现大规模并行有限差分弹性力学”，《计算物理学杂志》，第 427 卷，第 110065 页，2021年。[7] S. Whitman、J. Brasseur 和 P. Hamlington，“使用百亿亿次级燃烧代码 PeleC 模拟钝体稳定火焰”，2018年。[8] K. Sverdrup、N. Nikiforakis 和 A. Almgren，“利用结构化自适应网格细化对时变黏塑性流体流动进行高度并行化模拟”，《流体物理学》，第 30 卷，093102，2018 9，第 p. ，[9] Nyx，https://github.com/AMReX-Astro/Nyx，2021年。[10] P. Deutsch，“Gzip 文件格式规范 4.3 版”，1996年。[11] Zstandard，http://facebook.github.io/zstd/，2020年。[12] S. W. Son、Z. Chen、W. Hendrix、A. Agrawal、W.-k. Liao 和 A. Choudhary，“面向百亿亿次计算时代的数据压缩综述”，《超级计算前沿与创新》，第 1 卷，第 2 期，第 76–88 页，2014年。[13] S. Di 和 F. Cappello，“基于 SZ 的快速误差有界有损高性能计算数据压缩”，载于 2016 年 IEEE 国际并行与分布式处理研讨会（IPDPS）。IEEE，2016年，第 730–739 页。[14] D. Tao、S. Di、Z. Chen 和 F. Cappello，“基于多维预测和误差控制量化，显著提升科学数据集的有损压缩效果”，载于 2017 年 IEEE 国际并行与分布式处理研讨会（IPDPS）。IEEE，2017年，第 1129–1139 页。[15] X. Liang、S. Di、D. Tao、S. Li、S. Li、H. Guo、Z. Chen 和 F. Cappello，“ 针对高压缩比优化的科学数据集误差控制有损压缩 ”，载于 2018 年 IEEE 大数据国际会议。IEEE，2018年。[16] P. Lindstrom，“定速率压缩浮点数组”，IEEE 可视化与计算机图形学汇刊，第 20 卷，第 12 期，第 2674–2683 页，2014年。[17] M. Ainsworth、O. Tugluk、B. Whitney 和 S. Klasky，“MGARD：一种用于浮点数据压缩的多层次技术”，载于 SC 超级计算 DRBSD-2 研讨会，2017年。[18] X. Liang、B. Whitney、J. Chen、L. Wan、Q. Liu、D. Tao、J. Kress、D. Pug-mire、M. Wolf、N. Podhorszki 等，“MGARD+：优化误差有界科学数据约简的多层次方法”，IEEE 计算机汇刊，第 71 卷，第 7 期，第 1522–1536 页，2021年。[19] R. Ballester-Ripoll、P. Lindstrom 和 R. Pajarola，“TTHRESH：多维可视化数据的张量压缩”，IEEE 可视化与计算机图形学汇刊，第 26 卷，2903，2020 9，2891，第 pp. –。[20] F. Cappello、S. Di、S. Li、X. Liang、A. M. Gok、D. Tao、C. H. Yoon、X.-C. Wu、Y. Alexeev 和 F. T. Chong，“科学数据集中浮点数据有损压缩的应用案例”，《国际高性能计算应用杂志》，2019年。[21] S. Jin、P. Grosset、C. M. Biwer、J. Pulido、J. Tian、D. Tao 和 J. Ahrens，“理解用于极大规模宇宙学模拟的基于 GPU 的有损压缩”，载于 2020 年 IEEE 国际并行与分布式处理研讨会（IPDPS）。IEEE，2020年，第 105–115 页。[22] P. Grosset、C. M. Biwer、J. Pulido、A. T. Mohan、A. Biswas、J. Patchett、T. L. Turton、D. H. Rogers、D. Livescu 和 J. Ahrens，“Foresight：对数据约简至关重要的分析”，载于 SC20：国际高性能计算、网络、存储与分析大会。IEEE，2020年

[23] T. Lu, Q. Liu, X. He, H. Luo, E. Suchyta, J. Choi, N. Podhorszki, S. Klasky, M. Wolf, T. Liu 等人，“理解和建模高性能计算科学数据上的有损压缩方案”，载于*2018*年*IEEE*国际并行与分布式处理研讨会（*IPDPS*）。IEEE，2018年，第348–357页。[24] A. H. Baker, H. Xu, J. M. Dennis, M. N. Levy, D. Nychka, S. A. Mickelson, J. Edwards, M. Vertenstein 和 A. Wegener，“评估数据压缩对气候模拟数据影响的方法”，载于第*23*届高性能并行与分布式计算国际研讨会论文集。ACM，2014年，第203–214页。[25] A. H. Baker, H. Xu, D. M. Hammerling, S. Li 和 J. P. Clyne，“迈向多方法途径：气候模拟数据的有损数据压缩”，载于高性能计算国际会议。Springer，2017年，第30–42页。[26] A. M. Gok, S. Di, Y. Alexeev, D. Tao, V. Mironov, X. Liang 和 F. Cappello，“Pastri：量子化学中双电子积分的误差有界有损压缩”，载于*2018*年*IEEE*国际集群计算会议（*CLUSTER*）。IEEE，2018年，第1–11页。[27] X.-C. Wu, S. Di, E. M. Dasgupta, F. Cappello, H. Finkel, Y. Alexeev 和 F. T. Chong，“使用数据压缩进行全状态量子电路模拟”，载于国际高性能计算、网络、存储与分析大会论文集，2019年，第1–24页。[28] H. Luo, J. Wang, Q. Liu, J. Chen, S. Klasky 和 N. Podhorszki，“zmesh：探索应用特征以提高自适应网格细化的有损压缩率”，载于*2021*年*IEEE*国际并行与分布式处理研讨会。IEEE，2021年，第402–411页。[29] L. Fedeli, A. Huebl, F. Boillod-Cerneux, T. Clark, K. Gott, C. Hillairet, S. Jaure, A. Leblanc, R. Lehe, A. Myers 等人，“在超大规模级超级计算机上通过突破性的网格细化粒子网格模拟，推动基于激光的电子加速器设计前沿”，载于*SC22*：国际高性能计算、网络、存储与分析大会。IEEE，2022年，第1–12页。[30] IAMR，https://github.com/AMReX-Codes/IAMR，2023年。[31] S. Jin, S. Di, F. Vivien, D. Wang, Y. Robert, D. Tao 和 F. Cappello，“通过原位任务调度隐藏面向高性能计算应用的压缩加速I/O”，载于*EuroSys 2024*，2024年。[32] G. K. Wallace，“JPEG静态图像压缩标准”，*IEEE*消费者电子学汇刊，第38卷，第1期，第xviii–xxxiv页，1992年。[33] D. Le Gall，“MPEG：一种面向多媒体应用的视频压缩标准”，*ACM*通信，第34卷，第4期，第46–58页，1991年。[34] K. Zhao, S. Di, M. Dmitriev, T.-L. D. Tonellot, Z. Chen 和 F. Cappello，“通过动态样条插值优化科学数据的误差有界有损压缩”，载于*2021*年第*37*届*IEEE*国际数据工程会议。IEEE，2021年，第1643–1654页。[35] X. Liang, Q. Gong, J. Chen, B. Whitney, L. Wan, Q. Liu, D. Pug-mire, R. Archibald, N. Podhorszki 和 S. Klasky，“通过多层分解实现科学数据的误差控制、渐进式和自适应检索”，载于国际高性能计算、网络、存储与分析大会论文集，2021年，第1–13页。[36] F. Wang, N. Marshak, W. Usher, C. Burstedde, A. Knoll, T. Heister 和 C. Johnson，“基于树的自适应网格细化数据的CPU光线追踪”，*Computer Graphics Forum*，第39卷，第1–12页，2020年6月。[37] G. Harel, J.-B. Lekien 和 P. Pébaÿ，“对AMR网格可视化的两项新研究成果：I. 极大规模二维网格的交互式渲染；II. 任意维度的新型选择过滤器”，2017年3月。[38] D. Wang, J. Pulido, P. Grosset, J. Tian, J. Ahrens 和 D. Tao，“使用AMReX框架分析数据约简技术对自适应网格细化应用可视化的影响”，载于国际高性能计算、网络、存储与分析大会*SC'23*研讨会论文集，2023年，第263–271页。[39] K. Zhao, S. Di, X. Liang, S. Li, D. Tao, Z. Chen 和 F. Cappello，“通过二阶预测和参数优化显著改进高性能计算数据集的有损压缩”，载于第*29*届高性能并行与分布式计算国际研讨会论文集，2020年，第89–100页。[40] D. Wang, J. Pulido, P. Grosset, J. Tian, S. Jin, H. Tang, J. Sexton, S. Di, K. Zhao, B. Fang 等人，“Amric：一种面向自适应网格细化应用、用于高效I/O的全新原位有损压缩框架”，载于高性能计算国际会议论文集

[41] J. Bentley，“用于关联搜索的多维二叉搜索树”，*ACM*通信，*1975*年*9*月，第*18*卷：第*509–517*页：插图，含参考文献。，第18卷，1975年1月。[42] D. Hoang、H. Bhatia、P. Lindstrom 和 V. Pascucci，“大规模粒子数据的高质量、低内存占用渐进式解码”，载于*2021*年*IEEE*第*11*届大数据分析与可视化研讨会（*LDAV*）。IEEE 计算机学会，2021年，第32–42页。[43] G. Cirio、G. Lavoué 和 F. Dupont，“数据驱动的渐进式网格压缩框架”，载于*GRAPP*，2010年，第5–12页。[44] O. Devillers 和 P.-M. Gandoin，“用于交互式传输的几何压缩”，*Visualization ’00* 论文集，2000年1月。[45] D. Wang、J. Pulido、P. Grosset、S. Jin、J. Tian、J. Ahrens 和 D. Tao，“TAC：优化三维自适应网格细化模拟的误差有界有损压缩”，载于第*31*届国际高性能并行与分布式计算研讨会论文集，2022年，第135–147页。[46] X. Liang、S. Di、D. Tao、Z. Chen 和 F. Cappello，“一种具有逐点相对误差界的有损数据压缩高效变换方案”，载于*2018*年*IEEE*国际集群计算会议（*CLUSTER*）。IEEE，2018年，第179–189页。[47] S. Jin、J. Pulido、P. Grosset、J. Tian、D. Tao 和 J. Ahrens，“通过细粒度速率-质量建模自适应配置宇宙学模拟的原位有损压缩”，载于第*30*届国际高性能并行与分布式计算研讨会论文集，2021年，第45–56页。[48] M. Davis、G. Efstathiou、C. S. Frenk 和 S. D. White，“冷暗物质主导的宇宙中大尺度结构的演化”，《天体物理学杂志》，第292卷，第371–394页，1985年。[49] B. Fang、D. Wang、S. Jin、Q. Koziol、Z. Zhang、Q. Guan、S. Byna、S. Krishnamoorthy 和 D. Tao，“表征存储故障对HPC应用的影响：方法与见解”，载于*2021*年*IEEE*国际集群计算会议（*CLUSTER*）。IEEE，2021年，第409–420页。



Daoce Wang 是印第安纳大学布卢明顿分校智能系统工程专业的博士生。他于2018年获得电子科技大学计算机科学学士学位，并于2020年获得佛罗里达大学计算机科学硕士学位。他的研究兴趣包括科学数据约简、自适应网格细化模拟和并行算法。邮箱：daocwang@iu.edu。



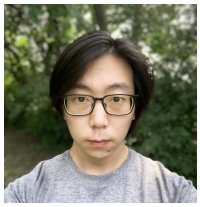
Jesus Pulido 是洛斯阿拉莫斯国家实验室规模化数据科学团队的研究科学家。Pulido于2019年获得加利福尼亚大学戴维斯分校计算机科学博士学位。Pulido专长于数据分析、数据约简、可视化、高性能计算、小波和多分辨率方法。他在图像传感器、天文学、湍流和宇宙学应用方面拥有经验。邮箱：pulido@lanl.gov。



Pascal Grosset 是洛斯阿拉莫斯国家实验室规模化数据科学团队的科学家。他的主要研究兴趣包括大规模数据分析与可视化，以及数据约简。他于2016年在犹他大学获得计算机专业博士学位，研究方向为图形学与可视化，研究重点是大规模可视化。电子邮箱：pascalgrosset@lanl.gov。



Sian Jin 是天普大学的助理教授。他于2023年获得印第安纳大学计算机工程博士学位，并于2018年获得北京师范大学物理学理学学士学位。他的研究兴趣包括高性能计算、数据压缩、神经网络和并行计算。他在多个主要期刊和国际会议上发表了数篇论文，包括 SC、PPoPP、VLDB、EuroSys、HPDC、IPDPS 和 ICDE。电子邮箱：sianjin@iu.edu。



Jiannan Tian 是印第安纳大学布卢明顿分校智能系统工程专业的博士候选人。他的研究兴趣包括科学数据有损压缩与误差分析，以及以GPU为中心的计算。他目前的项目包括开发GPU加速压缩算法，以及对有损压缩进行系统设计优化。电子邮件：jti1@iu.edu。



Kai Zhao 是佛罗里达州立大学计算机科学系助理教授。他于2022年获得加利福尼亚大学河滨分校计算机科学博士学位，并于2014年获得北京大学学士学位。他的研究兴趣包括高性能计算、科学数据管理与约简，以及弹性机器学习。电子邮件：kzhao@cs.fsu.edu。



James P. Ahrens于1989年获得马萨诸塞大学阿默斯特分校计算机科学理学学士学位，并于1996年获得华盛顿大学西雅图分校计算机科学博士学位。他是洛斯阿拉莫斯国家实验室的高级科学家。他的主要研究兴趣包括可视化、计算机图形学、数据科学和并行系统。他发表了100多篇经过同行评审的论文，也是ParaView的创始人兼设计负责人。ParaView是一款开源可视化工具，旨在处理超大型数据。ParaView广泛用于科学可视化，并已应用于全球各地的超算中心和科研中心。他担任 IEEE计算机学会可视化与图形技术委员会（TCVG）主席。电子邮箱：ahrens@lanl.gov。



Dingwen Tao 是印第安纳大学布卢明顿分校的副教授，并负责领导高性能数据分析与计算实验室。他于2018年获得加利福尼亚大学河滨分校计算机科学博士学位，并于2013年获得中国科学技术大学数学理学学士学位。他获得过多项奖项，包括美国国家科学基金会职业奖（2023年）、亚马逊研究奖（2022年）、Meta研究奖（2022年）、R&D 100奖获奖者（2021年）、IEEE计算机学会高性能计算技术委员会（TCHPC）青年学者卓越奖（2020年）、美国国家科学基金会CRII奖（2020年）和IEEE CLUSTER最佳论文奖（2018年）。他现任《IEEE并行与分布式系统汇刊》副编辑。他是IEEE和ACM高级会员。电子邮箱：ditao@iu.edu。