TAC: Optimizing Error-Bounded Lossy Compression for Three-Dimensional Adaptive Mesh Refinement Simulations

Daoce Wang (daoce.wang@wsu.edu)
Jesus Pulido (pulido@lanl.gov)
Pascal Grosset (pascalgrosset@lanl.gov)
Sian Jin (sian.jin@wsu.edu)
Jiannan Tian (jiannan.tian@wsu.edu)
James Ahrens (ahrens@lanl.gov)
Dingwen Tao (dingwen.tao@wsu.edu)
Background: Adaptive Mesh Refinement

- **Adaptive Mesh Refinement**
  - Increase resolution in regions of most interest
  - Reduce computational and storage overhead
  - Result in hierarchical AMR data with different resolutions
  - One of the most widely used frameworks for HPC applications

- **Example of AMR**
  - The mesh will be refined when the value meets the refinement criteria (i.e., greater than the threshold)
  - The grid structure changes with the universe’s evolution
  - The red boxes indicate different resolutions within one AMR level

https://www.cttc.upc.edu/?q=node/165
Different type of AMR

Tree-based AMR
- Tree-based AMR organizes the grids as leaves on the tree and has no redundant data across different level
- Tree-based AMR can be more complex and time consuming to perform visualization and analysis

Patch-based AMR
- Patch-based AMR saves the data that will be refined at the fine level in the coarse level redundantly
- The redundant coarse data will not be used in post analysis and vis
- We focus on patch-based AMR and discard the redundant coarse data while doing the compression

http://cucis.ece.northwestern.edu/projects/DAMSEL

AMReX: Building a Block-Structured AMR Application
Motivation: Why Compression

- Even with AMR, the size of data generated by apps could still be prodigious
  - One Nyx AMR dataset ($\frac{1}{2} \times 2048^3$ mesh points in the coarse level; $\frac{1}{2} \times 4096^3$ in the fine level) → 1.8 TB
  - Running the simulation 5 times with 200 snapshots dumped per simulation → 1.8 PB

- Trend of Supercomputing Systems
  - The compute capability is developed much faster than storage and bandwidth: a widening gap
    (1) between compute unit and storage bandwidth (PF–SB), or
    (2) between main memory size and storage bandwidth (MS–SB)

| supercomputer    | year | class | PF   | MS   | SB     | MS/SB | PF/SB |
|------------------|------|-------|------|------|--------|-------|-------|
| Cray Jaguar      | 2008 | 1 PFLOPS | 1.75 PFLOPS | 360 TB | 240 GB/S | 1.5k   | 7.3k  |
| Cray Blue Waters | 2012 | 10 PFLOPS | 13.3 PFLOPS | 1.5 PB  | 1.1 TB/S | 1.3k   | 13k   |
| Cray CORI        | 2017 | 10 PFLOPS | 30 PFLOPS  | 1.4 PB  | 1.7 TB/S* | 0.8k   | 17k   |
| IBM Summit       | 2018 | 100 PFLOPS | 200 PFLOPS | >10 PB** | 2.5 TB/S | >4k    | 80k   |

PF: peak FLOPS * when using burst buffer ** counting only DDR4

Source: F. Cappello (ANL)
Background: Lossy Compression

- **Lossy compression on scientific data**
  - Offers much **higher compression ratios** than lossless compression by trading a little bit of accuracy
  - Traditional lossy compressors (e.g., JPEG) are designed for images (integer) → bad performance on **scientific data** (floating-point data)
  - New generation of lossy compressors:
    1. **SZ** (Prediction based), nice compression ratio
    2. **ZFP** (Transform based), high throughput
    3. **TThresh** (HOSVD based), works nice in 3d but slow

---

*Error-Controlled Lossy Compression Optimized for High Compression Ratios of Scientific Datasets*
*Fixed-Rate Compressed Floating-Point Arrays*
*TTHRESH: Tensor Compression for Multidimensional Visual Data*
*COMET: A Novel Memory-Efficient Deep Learning Training Framework by Using Error-Bounded Lossy Compression*
Basic AMR Compression

- Challenges
  - Compared to non-AMR data, the structure of AMR data is more comprehensive.
  - The data of each level are stored separately in 1D, could be reshaped & convert to uniform resolution & combined for vis/post-analysis.

\[
\begin{align*}
\text{lvl}_0\text{.bin} & : (0A, 0B, 0C, 0D) \\
\text{lvl}_1\text{.bin} & : (1A, 1B, 1C)
\end{align*}
\]
Basic AMR Compression

- **1D Baseline Compression**
  - Compress the 1d directly → lose almost all the *spatial information*

- **2/3D Baseline Compression**
  - Compress in 2/3d with an up-sampled coarse level → *redundant data*

*No matter which method is used, we are faced with the problem of either low locality or redundant data*
SOTA AMR Compression: zMesh

- **An alternative solution of the 1d baseline**
  - **Smooth** (preprocess) the 1D data by **reordering** to help compression
  - Puts the points neighbored in the 2D layout closer in the 1D array

![Diagram](image)

- **Limitation**
  - Cannot apply different error bound for different lvls, different AMR lvls will have different “importance” based on the need for post-analysis.
  - Compress the data in 1D, can not fully utilize spatial information of high dimension data
Overview of TAC

- Compress Each Level Separately in high dimension
  - Each level contain empty regions that decrease the data smoothness and increase the data size

- Our Hybrid Pre-process Strategies
  - Three pre-processing strategies that can adapt based on the density of each AMR level
    1. Optimized Sparse Tensor Representation (OpST) for low-density level
    2. Adaptive k-D Tree (AKDTree) for medium-density level
    3. Ghost-Shell Padding (GSP) for high-density level

Visualization of data distributions of an example AMR dataset (finest level)
OpST for Low-density Data

- Naïve Sparse Tensor Representation (NaST)
  - Partition $\rightarrow$ Linearize blocks $\rightarrow$ Remove empty blocks $\rightarrow$ Pass to SZ $\rightarrow$ Reconstruct
  - Needs a small unit-block size to effectively remove the empty regions (e.g., $16^3$ vs $512^3$) $\rightarrow$ high proportion of boundary data $\rightarrow$ low compression performance
OpST for Low-density Data

- **OpST: larger sub-block**
  - Partition → Use Dynamic Programming to initiate an array $BS$ to save the size of the maximum square whose bottom-right corner is that unit block → Extract the big sub-block → Update $BS$ → Pass to SZ after done extraction

A 2D example of OpST. The subblocks are extracted according to $BS$. E.g., a 2-by-2 sub-block $B_0$ is extracted according to $BS_1$ [2] [1].
OpST vs NaST

- OpST can significantly reduce the overall compression error

Visual (one slice) of compression errors of two approaches using SZ based on Nyx “baryon density” field

OpST (CR = 233.8, PSNR = 76.9 dB)
AKDTTree for Medium-density Data

- Address high overhead issue of OpST for denser data
  - Time complexity of OpST: $O(N^2d)$, $N$ is the unit block number and $d$ is the density
  - Time complexity of AKDTTree: $O(\frac{1}{3}NlogN)$

- Remove empty regions and extract sub-blocks
  1. Partition
  2. Use a tree to represent the data, each node is associated with a sub-block
  3. Adaptively split each sub-block from the middle among one of the dimension
  4. Keep splitting a node until it is full or empty
  5. Collect all the leaf nodes and send them to the compressor

2D Example of AKDTTree

Select the dimension which can maximize the difference of the numbers of non-empty unit blocks of the two children
AKDTree for Medium-density Data

- **Adaptive splitting**

1. Categorize nodes into three different types: "cube" (1:1:1), "flat" (2:2:1), and "slim" (1:2:2)
2. Divide cube node $d$ into eight oct-blocks, $s_1,\ldots, s_8$ → get the counts of non-empty unit blocks $c_1,\ldots, c_8$ of $s_1,\ldots, s_8$ → decide along which dimension to split
3. For the flat node $d_1$, we can reuse $c_1,\ldots, c_4$ to decide how to split
4. Simply split the slim node $d_{11}$ along x-axis
5. This process (i.e., cube nodes → flat nodes → slim nodes) will be looped until the node is empty or full

Only count every three step (i.e., only for the “cube” nodes) → $O(\frac{1}{3}N\log N)$
GSP for High-density Data

- Not much room for removing empty regions for dense data
  - OpST and AKDTree will hurt the data locality/smoothness
  - Pad zeros into the few empty regions → higher error at the boundary

Visualization of data distributions of an example AMR dataset (coarse level)

Zero Filling (CR = 156.7, PSNR = 32.8 dB)
Ghost Shell Padding (GSP)

- Partition → pad empty unit block using the average of its non-empty neighbors’ boundary data values.
- For empty unit blocks have more than one non-empty neighbors → use the avg value of all its neighbors for padding.

2D Example of GSP. Non-empty blocks are in navy blue; padded blocks are in light blue/red; padded blocks based on more than one non-empty neighbors are in red.

GSP (CR = 161.3, PSNR = 33.5 dB)
Hybrid Compression Strategy

- Adaptively choose a best-fit pre-process strategy
  - We have: OpST (low-density), AKDTree (medium-density), and GSP (high density)
  - Use **two data-density thresholds** to determine when to use OpST, AKDTree, or GSP

- First threshold $T_1$ for switching between OpST and AKDTree
  - OpST and AKDTree have almost same compression performance in terms of bit-rate and PSNR
  - Time cost of OpST **increases linearly** with **data density**

$T_1 = 50$

Time overhead of OpST and AKDTree on different datasets with different densities.
Hybrid Compression Strategy

- Second threshold $T_2$ for switching between AKDTree and GSP
  - When the density is low, AKDTree is better; when the density gets higher, GSP gradually outperforms AKDTree
  - AKDTree and GSP have similar compression performance when the density is around 60% $\Rightarrow T_2 = 60%$

Compression performance comparison (PSNR vs Bitrate) of GSP, OpST and AKDTree on six datasets with different densities
Evaluation

- **Experimental Setup**
  - Real-word application: Nyx cosmology simulation based on AMReX
  - Datasets: 7 datasets generated by 2 runs with different numbers of AMR levels, simulating a region of 64 megaparsecs (Mpc)
  - Platform: two 28-core Intel Xeon Gold 6238R processors and 384 GB DDR4 memory

| Dataset | # Levels | Grid Size of Each Level (Fine to Coarse) | Density of Each Level (Fine to Coarse) |
|---------|----------|----------------------------------------|---------------------------------------|
| Run1_Z10 | 2        | 512, 256                               | 23%, 77%                              |
| Run1_Z5  | 2        | 512, 256                               | 58%, 42%                              |
| Run1_Z3  | 2        | 512, 256                               | 64%, 36%                              |
| Run1_Z2  | 2        | 512, 256                               | 63%, 37%                              |
| Run2_T2  | 2        | 256, 128                               | 0.2%, 99.8%                           |
| Run2_T3  | 3        | 512, 256, 128                          | 0.02%, 0.56%, 99.42%                  |
| Run2_T4  | 4        | 1024, 512, 256, 128                    | 3E-5, 0.02%, 2.2%, 97.7%              |

Our tested datasets

Visual of baryon density field of z10 (early timestep)

z5 (later timestep)
Evaluation

- **Evaluation on Rate-distortion**
  - Outperforms naïve 1D baseline & zMesh (up to 3.3x)
  - Perform much better than 3D baseline when
    1. finest level has a relatively **low density**, or
    2. decompressed data has a **high PSNR**

- **Discussion on Comparison with Baselines**
  - 3D baseline works better when finest level is dense
    - Dense finest lvl \(\rightarrow\) similar to **non-AMR dataset**
      \(\rightarrow\) no need to use AMR compress strategies
  - zMesh cannot improve the smoothness if there is **no data redundancy** in the AMR datasets
Evaluation

- **Evaluation on Time Overhead**
  - Up to **75x** faster than 3D baseline on the Run2 datasets and **2.4x** faster on the Run1
    - Due to Run2 has higher overhead of redundant data for the 3D baseline
  - Throughput degrades on the small datasets
    - Due to a relatively heavy launching time compared to the overall time on the small datasets

| $E_{abs}$ | Run1_Z2 | Run1_Z3 | Run1_Z5 | Run1_Z10 | Run2_T2 | Run2_T3 | Run2_T4 |
|----------|---------|---------|---------|---------|---------|---------|---------|
| 1E+08    | 169     | 94      | 97      | 166     | 90      | 94      | 161     | 76      | 99      | 160     | 40      | 95      | 152     | 17      | 76      | 143     | 2.4     | 60      | 125     | 0.4     | 30      |
| 1E+09    | 219     | 115     | 121     | 213     | 120     | 127     | 208     | 109     | 123     | 208     | 63      | 117     | 193     | 27      | 91      | 184     | 3.9     | 66      | 159     | 0.5     | 32      |
| 1E+10    | 259     | 125     | 135     | 256     | 125     | 136     | 253     | 117     | 137     | 250     | 65      | 135     | 242     | 30      | 102     | 229     | 4.0     | 72      | 197     | 0.5     | 34      |

Overall compression/decompression throughput (MB/s) of different approaches with different absolute error bounds
Evaluation

- Evaluation on Post-analysis Quality with Adaptive Error Bound
  - TAC can apply different error bounds to different AMR levels based on (1) the post-analysis metrics, (2) the up-sampling rates of coarse levels, and (3) the rate-distortion trade-off between different AMR levels
  - Power spectrum (PS) eb ratio: (1) 1:1, PS focus on the global quality → (2) 8:1, up-sample rate is $2^3$ → (3) 3:1
  - Halo finder (HF) eb ratio: (1) 2:1 HF focus on finer data → (2) 4:1 → (3) 2:1

We compare the PS $p'(k)$ of decompressed data with the original $p(k)$ and accept a maximum relative error within 1% (red dashed line) for all $k < 10$. The mass change, and the number of cells change for the biggest halo identified using the 3D baseline, TAC with uniform and, TAC with adaptive error bound.
Conclusion & Future Work

**Conclusion**
- Propose TAC, an error-bounded lossy compression for 3D AMR data
- Propose three pre-processing strategies that can adapt based on the density of each AMR level
- Improve the compression ratio compared to the STOA approach by up to 3.3x under the same data quality loss
- Tune the error-bound ratio of fine and coarse levels for better post analysis quality

**Future work**
- Apply our hybrid compression approach to more AMR simulations.
- Address the issue of low throughput on small AMR datasets.
Thank you!
Any questions and ideas are welcomed

Contact: Dingwen Tao: dingwen.tao@wsu.edu
Daoce Wang: daoce.wang@wsu.edu