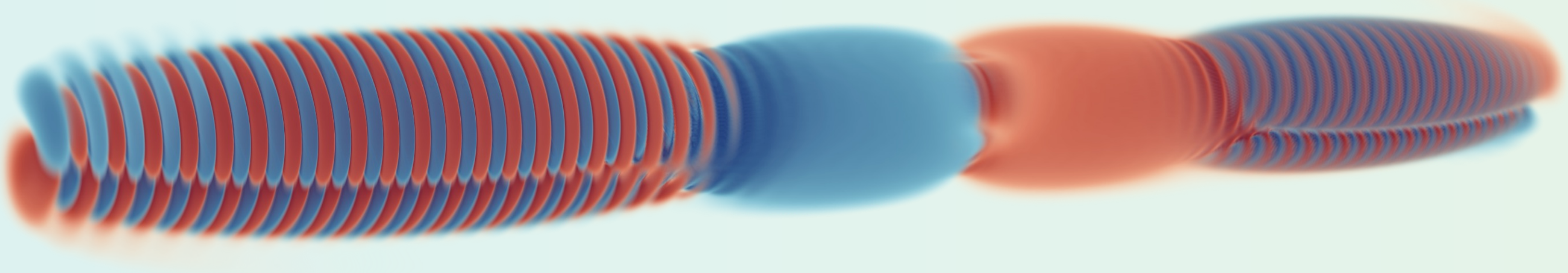
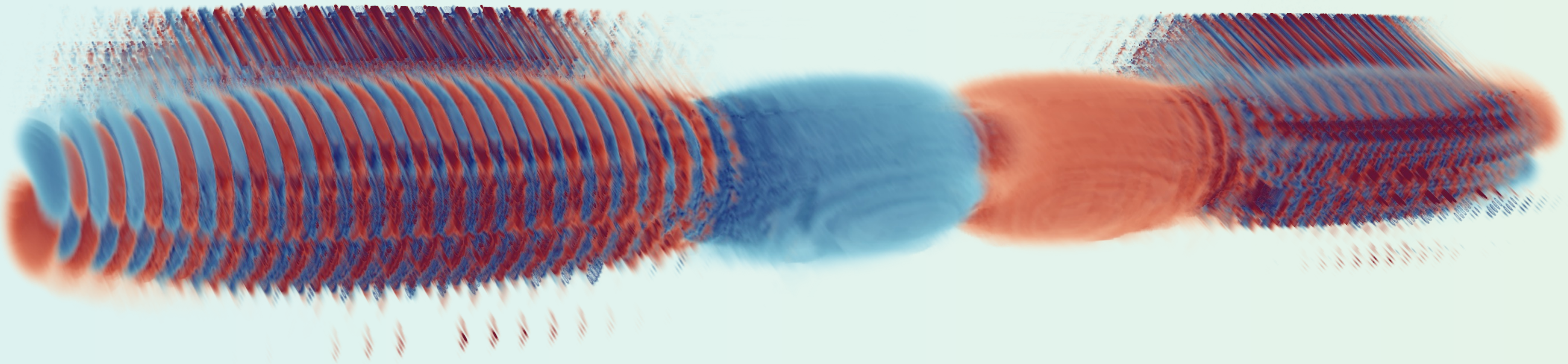
AMRIC: A Novel In Situ Lossy Compression Framework for Efficient I/O in Adaptive Mesh Refinement (AMR) Applications

Daoce Wang, Jesus Pulido, Pascal Grosset, Jiannan Tian, Sian Jin, Houjun Tang, Jean Sexton, Sheng Di, Kai Zhao, Bo Fang, Zarija Lukić, Franck Cappello, James Ahrens, Dingwen Tao

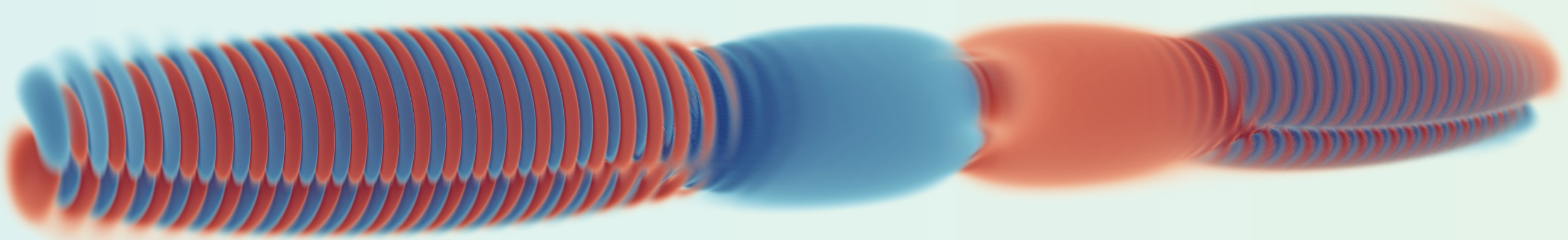
Better AMR Compression



Original Data



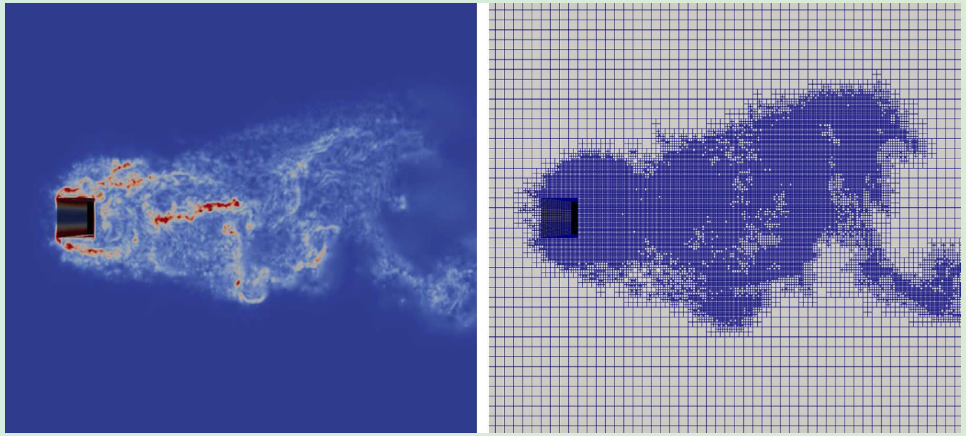
AMReX
Compression Ratio = 19



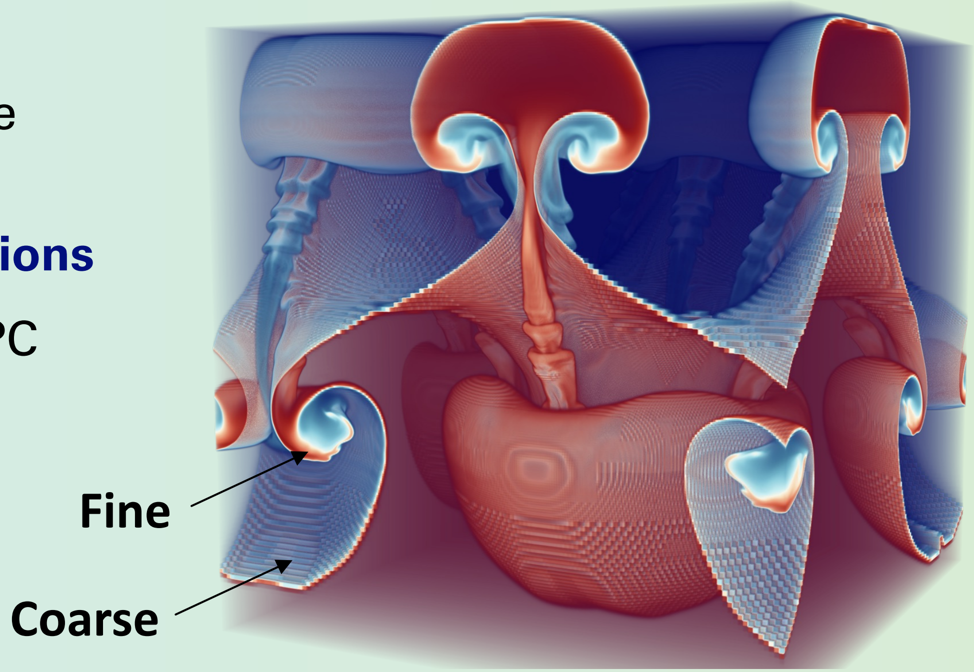
Our AMRIC
Compression Ratio = 24

Background: AMR

https://www.cttc.upc.edu/?q=node/165

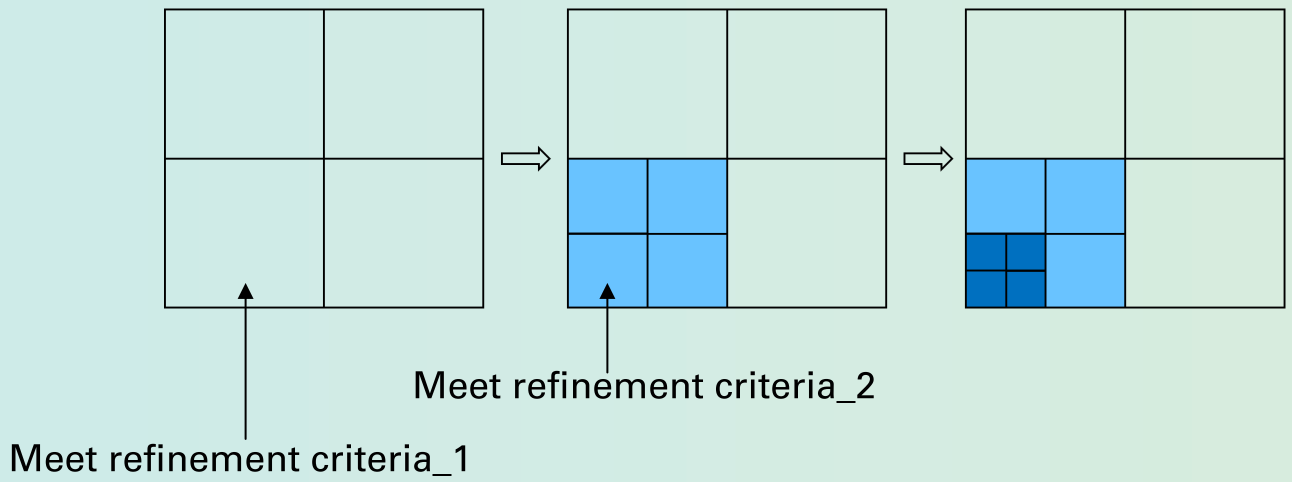


- Introduction to AMR
  - Each mesh represents a value of an area.
    - Smaller mesh → higher resolution
  - Change the mesh (**spatial resolution**) based on the level of refinement needed by the simulation, use **finer mesh** in **"more important"** region.
  - Achieve the desired accuracy as well as increase computational and storage savings.
  - Result in hierarchical data with **different resolutions**
  - One of the most widely used frameworks for HPC applications



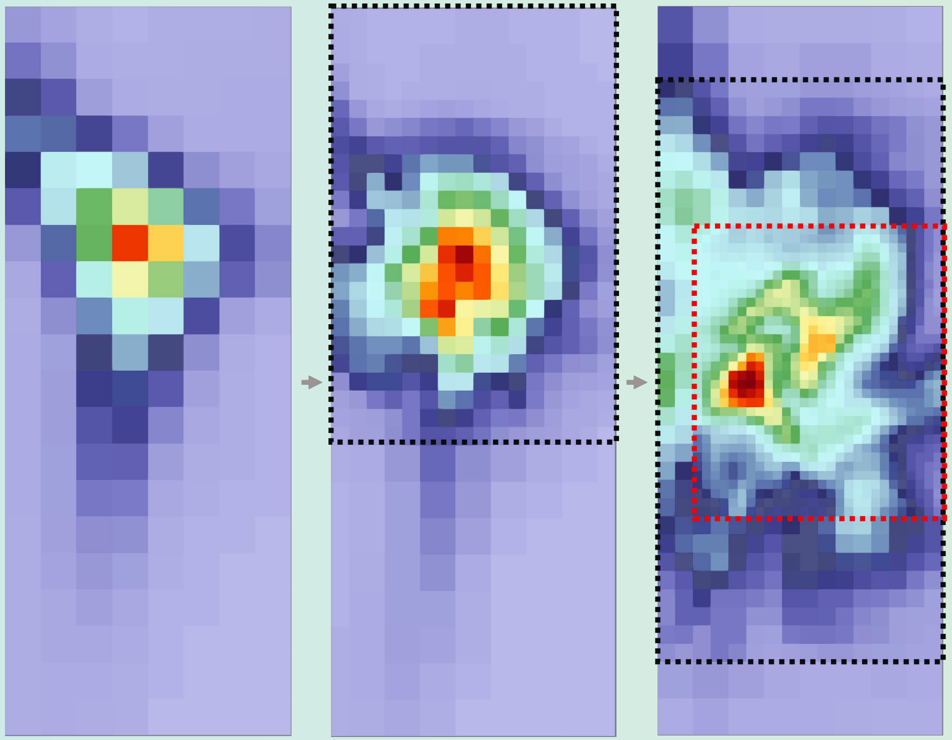
Background: AMR

- Example of AMR
  - A mesh will be refined when its value (e.g., density/velocity) meets refinement criteria (i.e., greater than the threshold)



Background: AMR

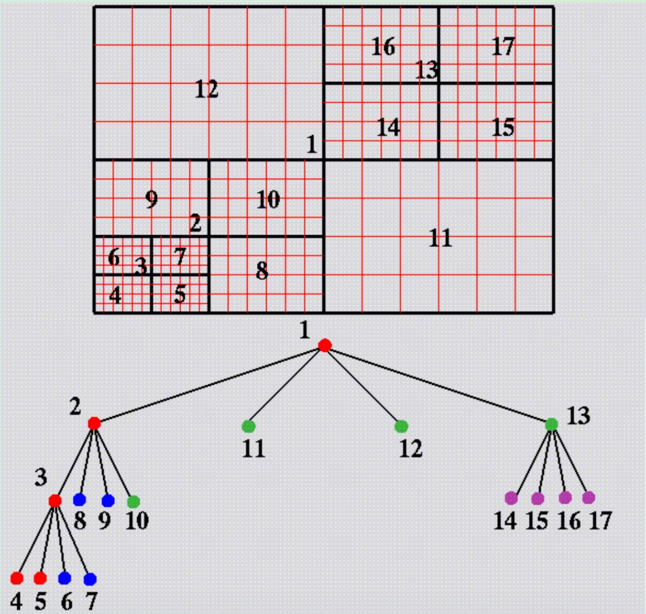
- Example of AMR
  - The grid structure changes with the universe's evolution
  - The dashed boxes indicate **different resolutions** within one timestep



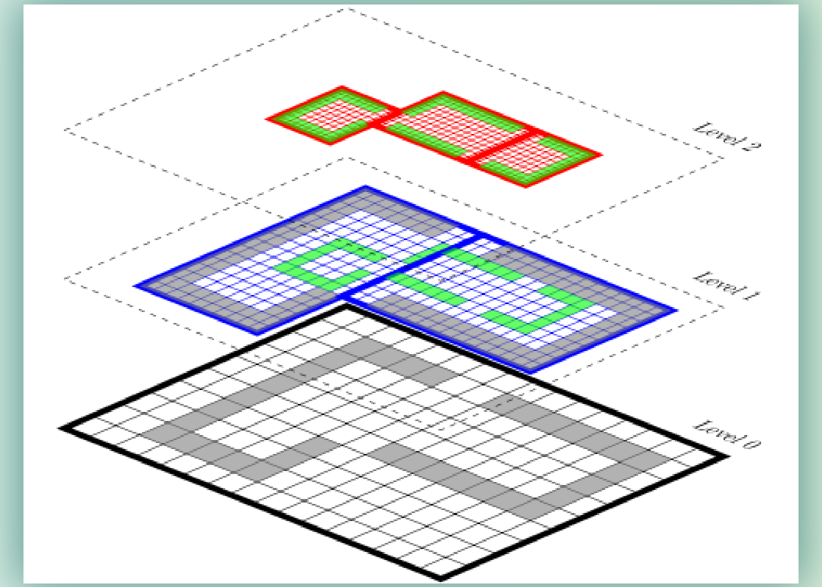
Vis of three key timesteps (zoom in) of a cosmology simulation

Different Types of AMR

- Tree-based AMR
  - Organizes the grids as leaves and **has no redundant** data across different levels
  - More complex and time-consuming to perform visualization and analysis
- 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 **AMReX**
  - We 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

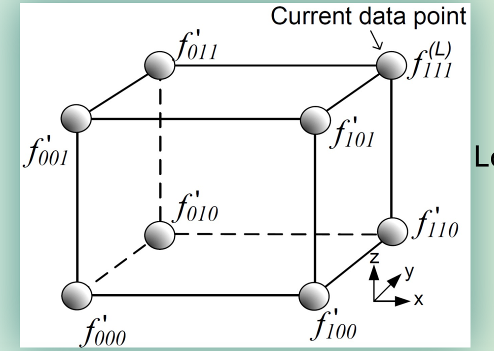
- Even with AMR, the size of data generated by apps could still be prodigious
  - E.g., Nyx non-AMR dataset 4096^3 * 10 = 5TB (only for one snapshot)
    - AMR dataset: 50% full resolution, 50% half resolution → 2.8TB
    - Will take a single node at Summit for 2.8TB/2.1 GB/s = 22 mins
- Trend of Supercomputing Systems
  - The compute capability is developed much faster than storage bandwidth: a widening gap
    - between compute unit and storage bandwidth (PF–SB), or
    - 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**
FRONTIER	2023	1 exaFLOPS	1680 PFLOPS	39 PB	~7.5 TB/S	**5.2K**	**224k**

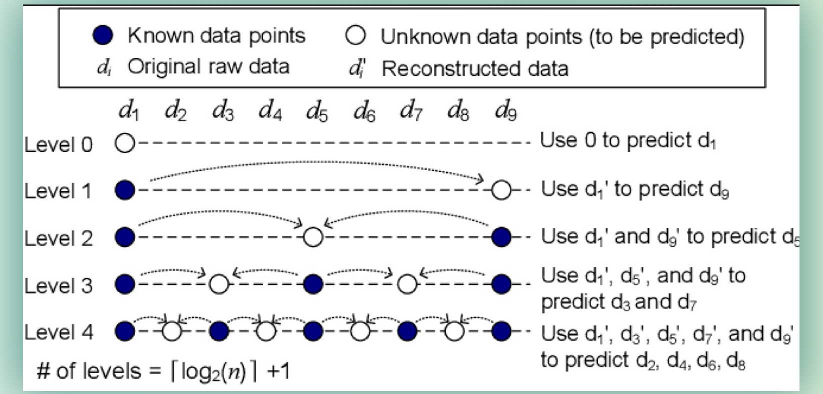
PF: peak FLOPS   * when using burst buffer   ** counting only DDR4   *Source: F. Cappello (ANL)*

Background: Compression

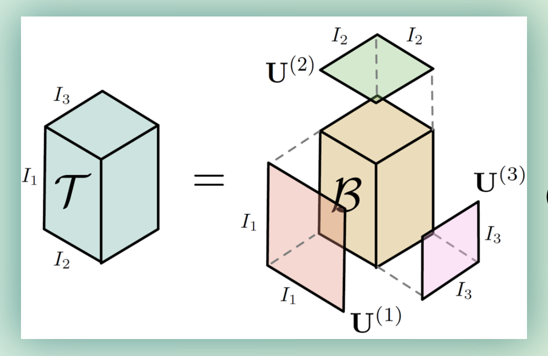
- Lossy compression on scientific data
  - Offers much **higher compression ratios** than lossless compression by trading a little bit of accuracy
  - Traditional lossy compressors (JPEG) are designed for **images** (int) → bad performance on **scientific data** (floating-point data)
  - New generation of lossy compressors:
    - SZ (Prediction based), nice compression ratio
      - SZ-**L**or/**R**eg (faster); SZ-Interp (higher ratio)
    - ZFP (Transform based), high throughput
    - TThresh (HOSVD based), works nice in 3d but slow



Lorenzo predictor (SZ-Lor/Reg)



Spline interpolation (SZ-Interp)



HOSVD (TThresh)

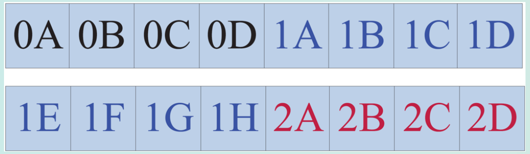
Error-Controlled Lossy Compression Optimized for High Compression Ratios of Scientific Datasets

Fixed-Rate Compressed Floating-Point Arrays

TTHRESH: Tensor Compression for Multidimensional Visual Data

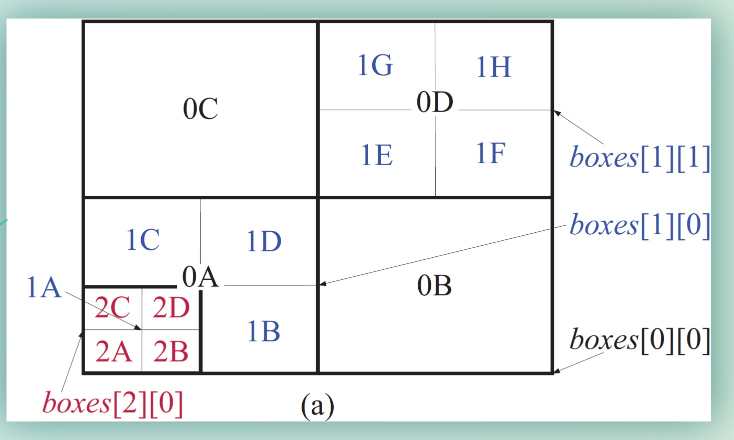
SOTA AMR Compression

- zMesh [Luo et al., IPDPS’21]
  - Preprocess and leverage the data redundancy across different AMR levels.
  - Compress the 2/3D data in 1D → cannot leverage higher-dimension compression



original 1d data

reordered 1d data



2d AMR data layout

- TAC [Wang et al., HPDC’22]
  - Improve zMesh’s compression quality through adaptive 3D compression
    - **3.3x** higher compression ratio under the same data distortion

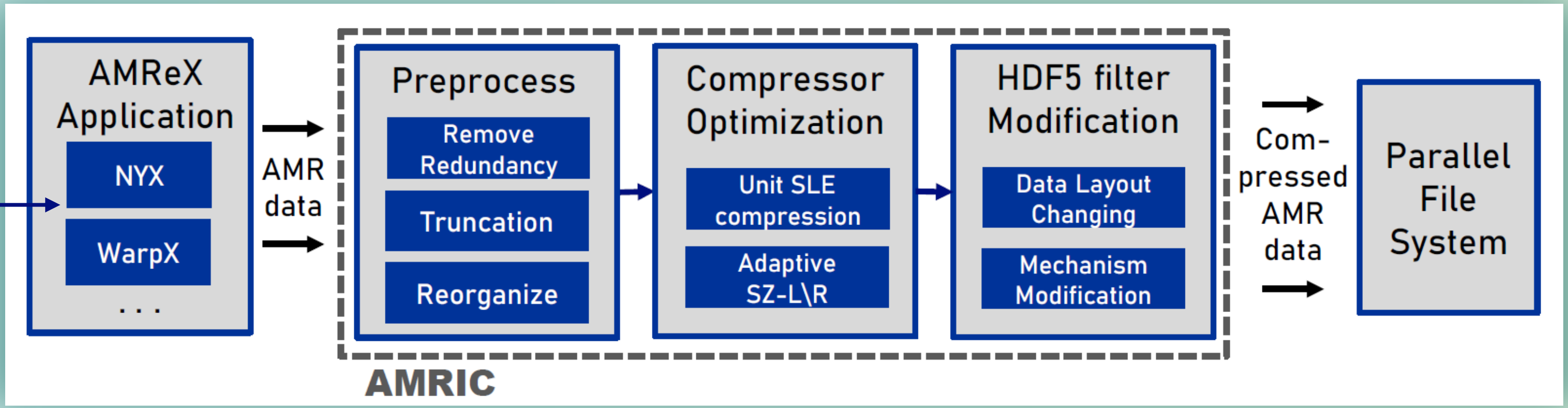
Use in situ Compression with Simulation

- zMesh, TAC are not suitable for in situ compression
  - zMesh requires **extra communication** for reordering in parallel scenarios
  - TAC requires the reconstruction of the entire domain's hierarchy
    - Result in significant **overhead** for in parallel scenarios
- In situ compression could save time & enhance I/O efficiency
  - Compressing data during the application's runtime
    - Offline: app **write ori data** to disk → SZ **read ori data** → SZ **write compressed data** to disk
    - In situ: app **directly writes compressed data** to disk
- AMReX framework supports in situ AMR data compression (SZ)
  - Only compresses the data in 1D → low **compression quality**
  - Cannot effectively utilize the HDF5 filter → low **CR** and **I/O performance**

AMRIC

- AMRIC: First **In Situ** Lossy Compression Framework for AMR Applications
  - Improve both **I/O efficiency** and boost **compression perf** for AMR applications

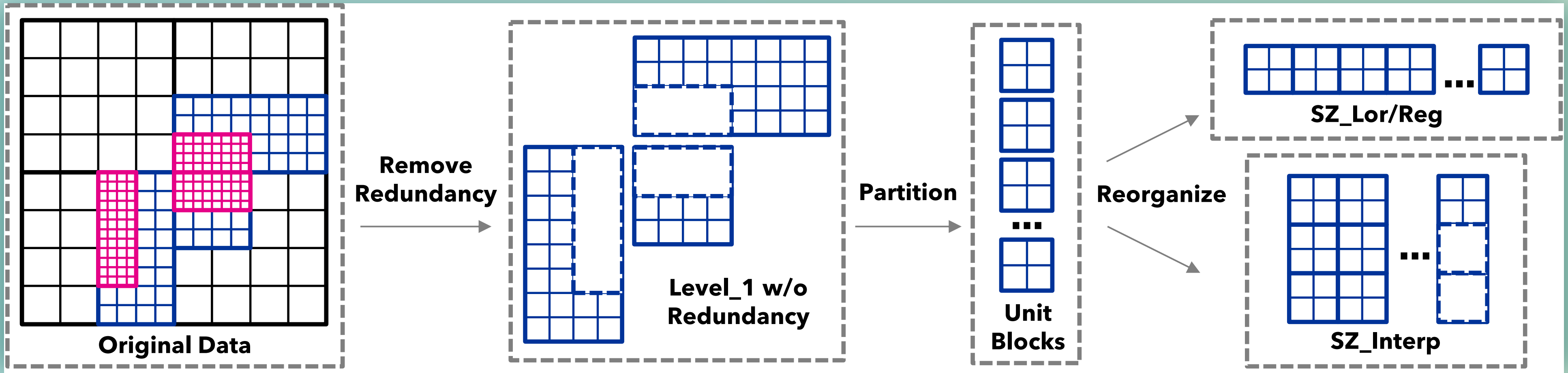
Integrated into AMReX and can be easily deployed to AMReX's app



- Overview
  - Compression-oriented preprocessing workflow for 3D AMR data compression
  - Optimize the state-of-the-art SZ lossy compressor's efficiency on AMR data
  - Overcome the gap between the HDF5 and AMR applications

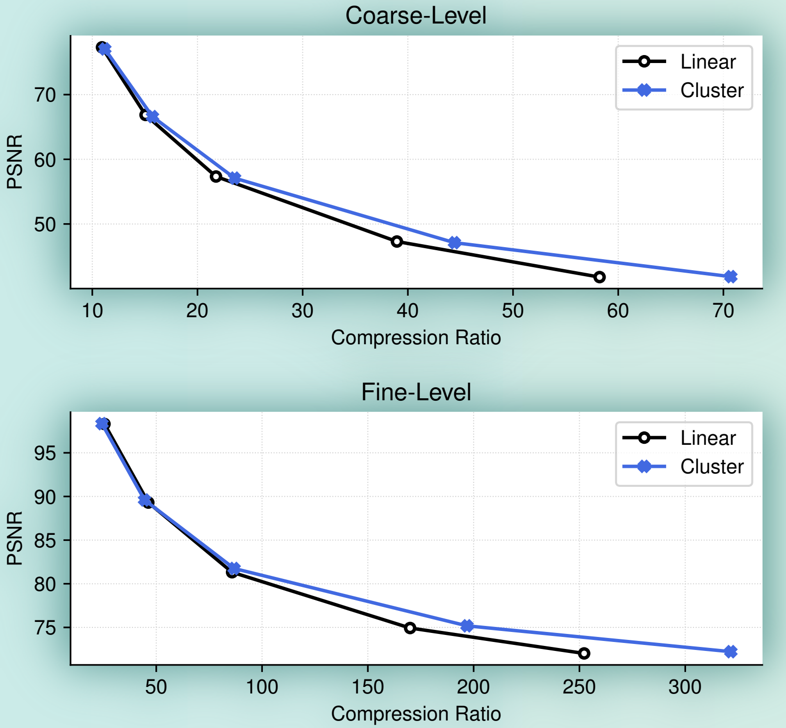
Prepossessing of AMR Data

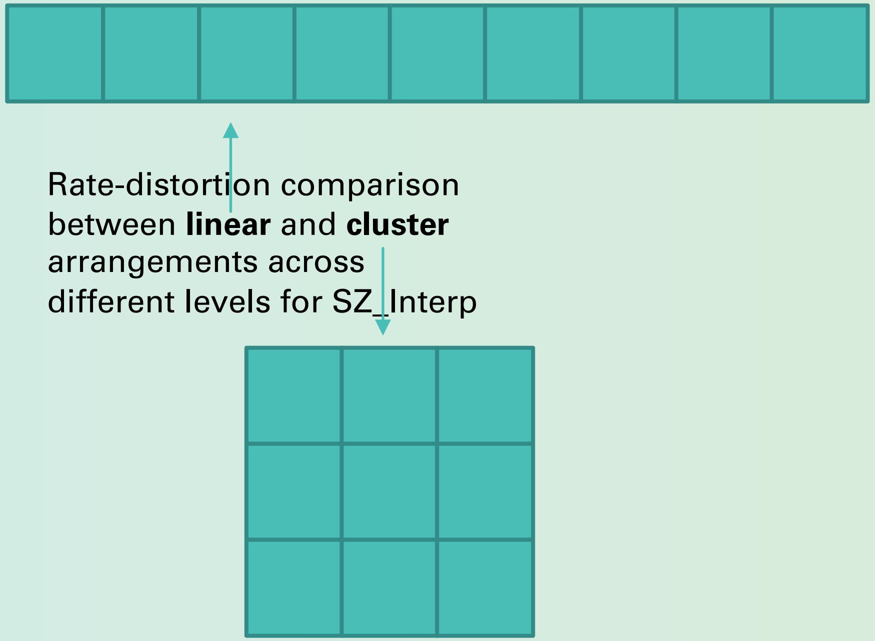
- Remove redundancy
  - The coarsest lvl_0 overlaps with the finer lvl_1; lvl_1 overlaps with the finest lvl_2
  - The redundant coarse regions can be removed to **save space**
- Challenge: irregular shapes for 3D compression
  - Use **uniform partition** to rearrange the boxes into a collection of unit blocks
  - **Reorganizing** blocks based on different compressors to improve compression performance



Prepossessing of AMR Data

- For SZ_L/R, we simply linearize the partitioned unit blocks for **high-speed**
- For SZ_Interp, we **cluster** the unit blocks for better compression performance
  - SZ_Interp will perform **global** interpolation for all 3 dimensions of the **entire dataset**
  - Cluster the unit blocks to **balance** the interpolation process across multiple dimensions

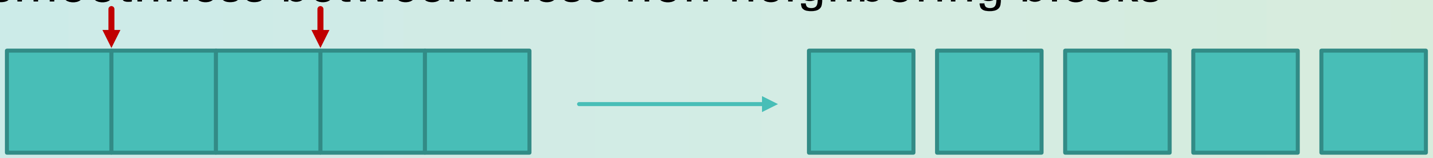




Rate-distortion comparison between **linear** and **cluster** arrangements across different levels for SZ_Interp

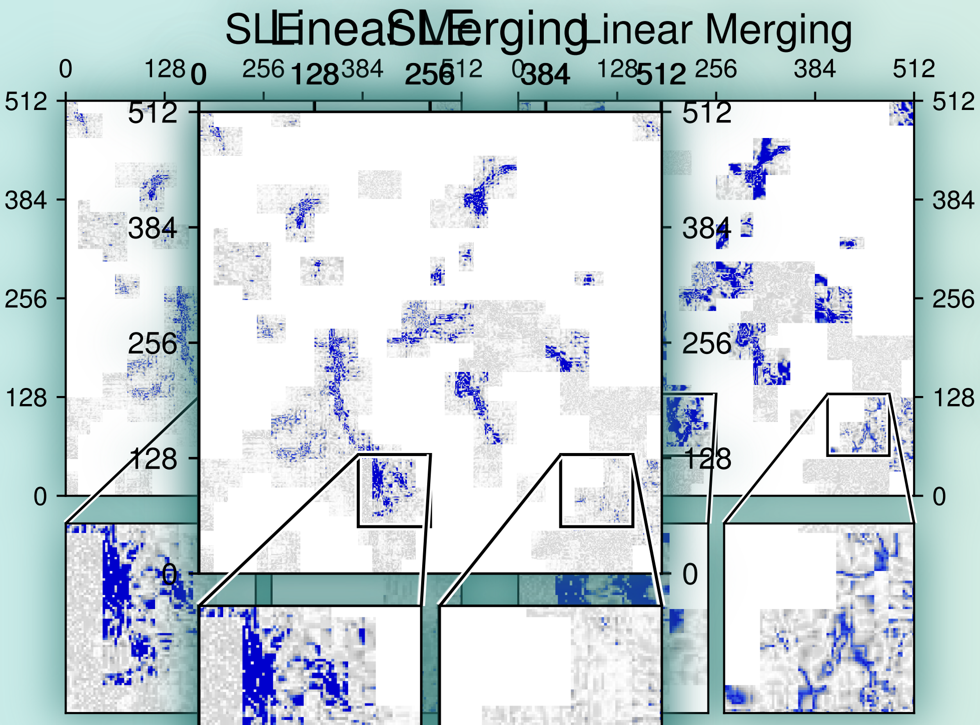
Optimization of SZ-L/R: (1) SLE

- Challenge 1: **Low prediction accuracy** on AMR data
  - Merged unit blocks may not be adjacent in the original dataset, resulting in poor data smoothness between these non-neighboring blocks

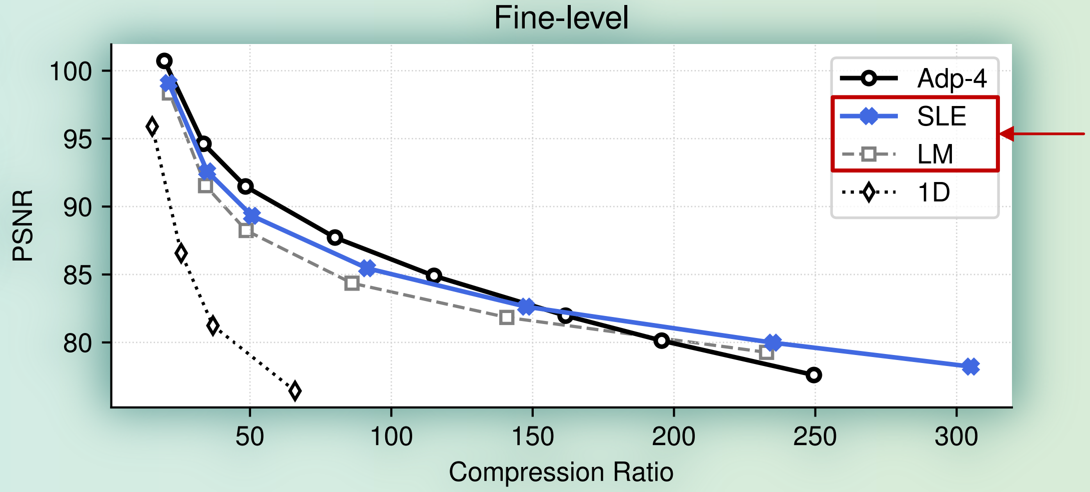


  - Directly compress each unit block individually?
    - No, since SZ will use lots of Huffman trees to encode these blocks separately
    - Result in low encoding efficiency → **low compression ratio**
- Solution 1: Improve prediction using unit Shared Lossless Encoding (**SLE**)
  - **Separate prediction** of unit blocks while **encoding** them **together**
    - Each unit block is predicted and quantized individually
    - The quantization codes from each unit block are combined to create a **shared Huffman tree** and then encoded

Optimization of SZ-L/R: (1) SLE



Vis comp ression errors of
unit SLE ear merging
(right, CR ompression error

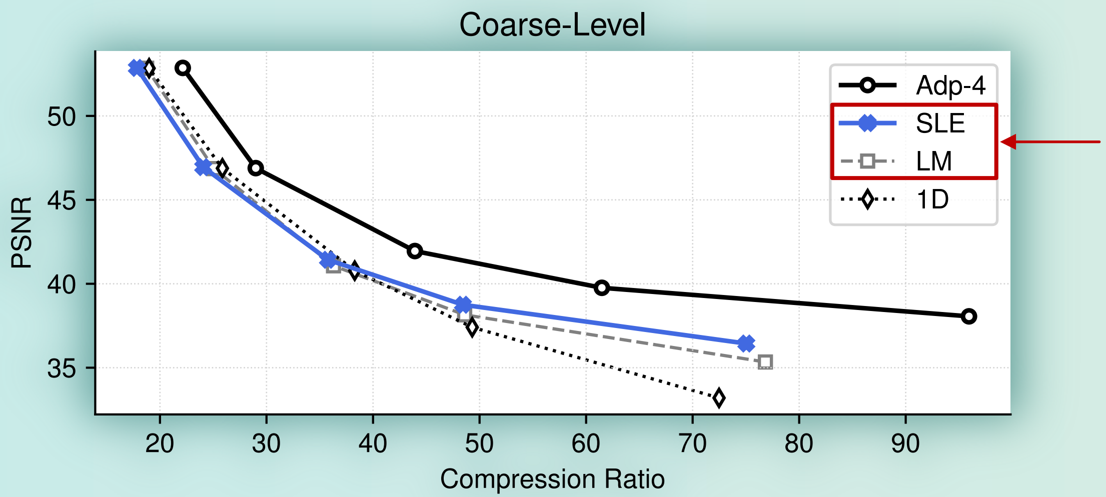


Rate-distortion comparison between LM (original SZ) and SLE

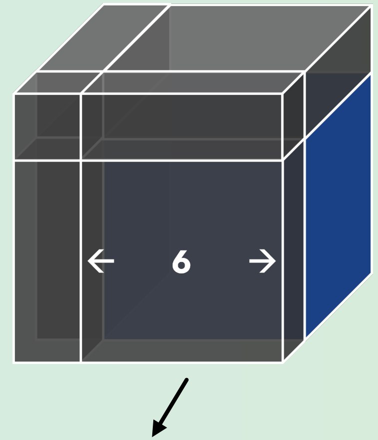
- Solution 1: Improve prediction using unit Shared Lossless Encoding (**SLE**)
  - **Separate prediction** of unit blocks while **encoding** them **together**
    - Reduces overall **compression error** → improvement in rate-distortion

Optimization of SZ: (2) Adaptive SZ-L/R

- Challenge 2: SLE may produce **undesirable residues**
  - SZ_L/R compressor will truncate the input data into 6×6×6 blocks
    - When using unit SLE, the SZ_L/R will further partition unit blocks using 6×6×6 block
    - The unit block size of AMR data is 2^n
  - Might produce blocks that are **difficult to compress**



When unit block size = 8, SLE cannot dominate baseline

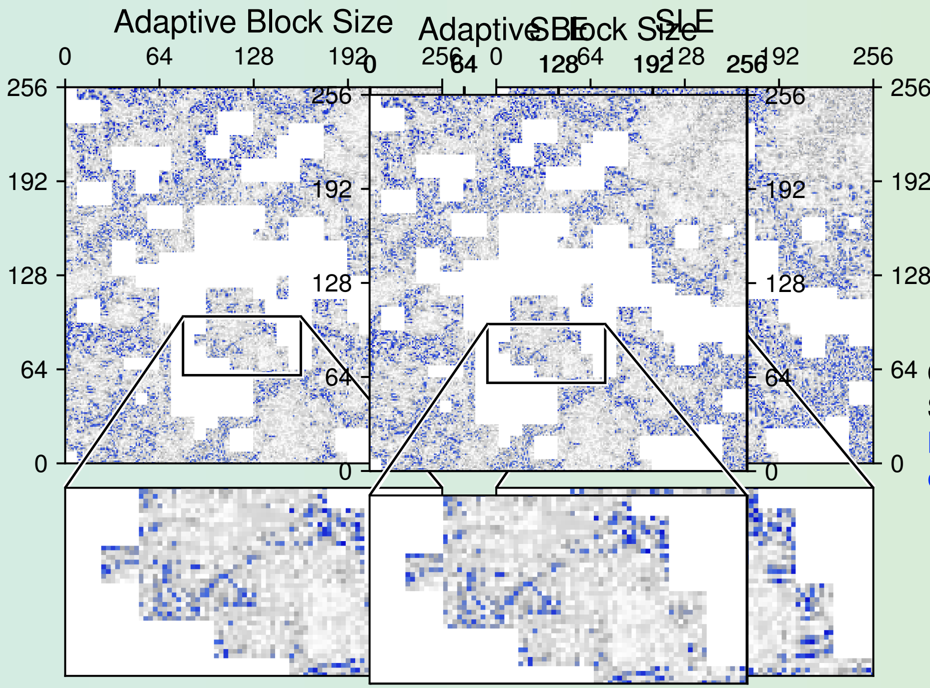


Example of the **original partition** of SZ_L/R on a unit block with the size of 8×8×8; the gray boxes represent data that are **difficult to compress**

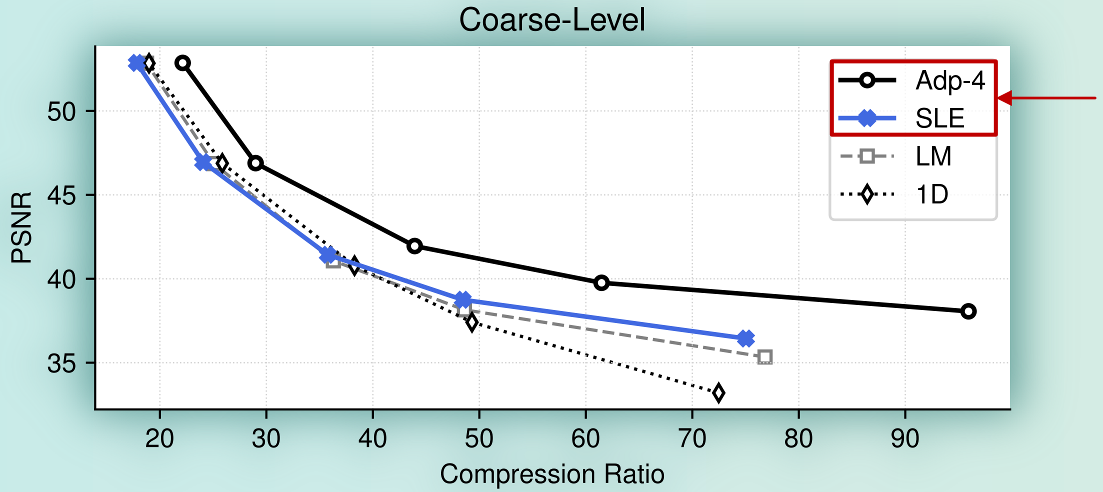
Optimization of SZ: (2) Adaptive SZ-L/R

- Solution 2: SZ_L/R with adaptive block size

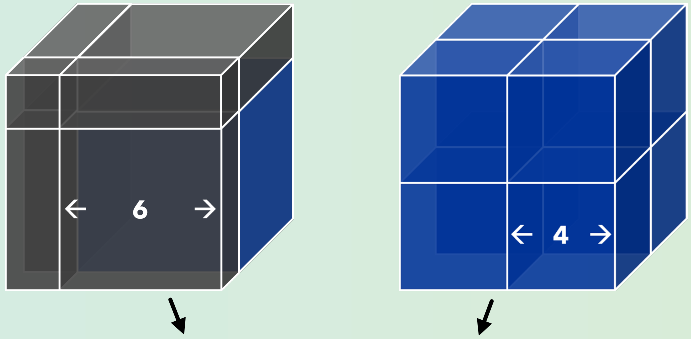
$$\text{SZ\_BlkSize} = \begin{cases} 4\times4\times4, & \text{if unitBlkSize mod } 6 \le 2; \\ 6\times6\times6, & \text{if unitBlkSize mod } 6 > 2; \\ 6\times6\times6, & \text{if unitBlkSize} \ge 64 \end{cases}$$



Compression errors of **SLE** and **Adp block size**
Bluer means **higher compression error**



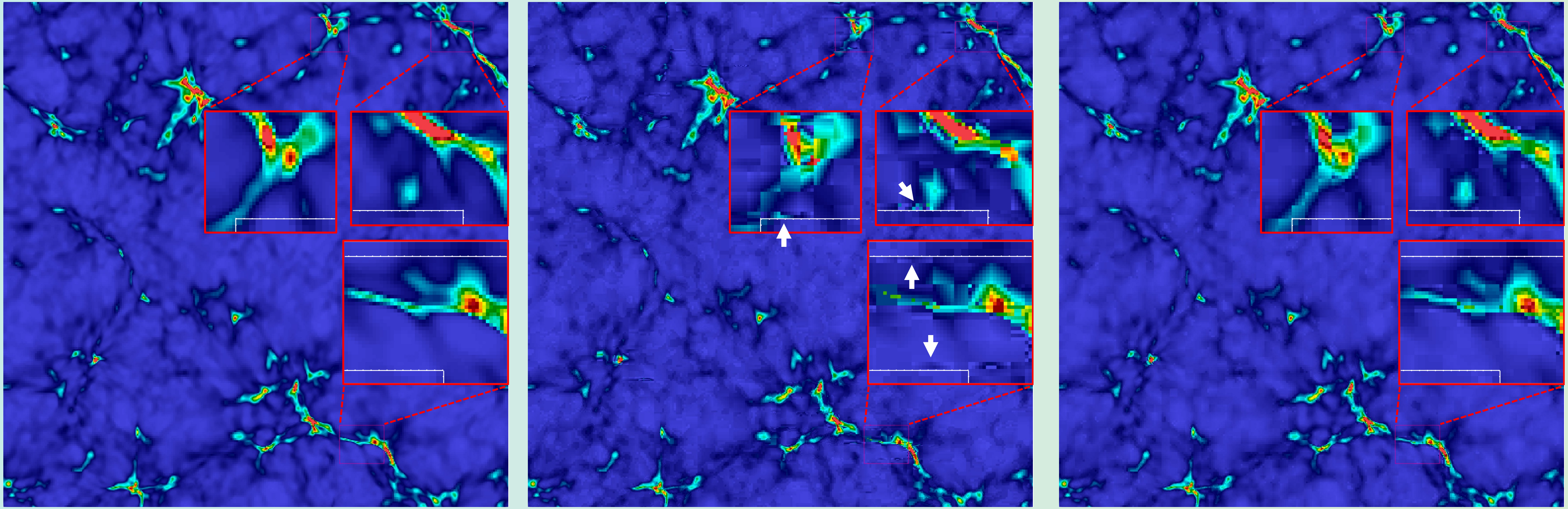
When unit block size = 8, SLE cannot dominate baseline



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

Optimization of SZ: Overall Vis Improvement

- AMRIC's optimized SZ_L/R notably enhances the **visualization quality** of AMR data



Original data | Original SZ_L/R, **CR =51.7** | AMRIC SZ_L/R, **CR =53.2**

Vis comparison (one slice) of ori data and decompressed data produced by original SZ_L/R and AMRIC's SZ_L/R on Nyx. **Warmer colors** indicate **higher values**. The **white lines** denote the **boundaries** between AMR levels.

H5 Compression Filter Modification

- HDF5 natively supports data compression filters such as H5Z-SZ
  - Allows **chunked data** to pass through compression filters on the way to the disk
  - Data can be compressed using a compression filter during the write operation

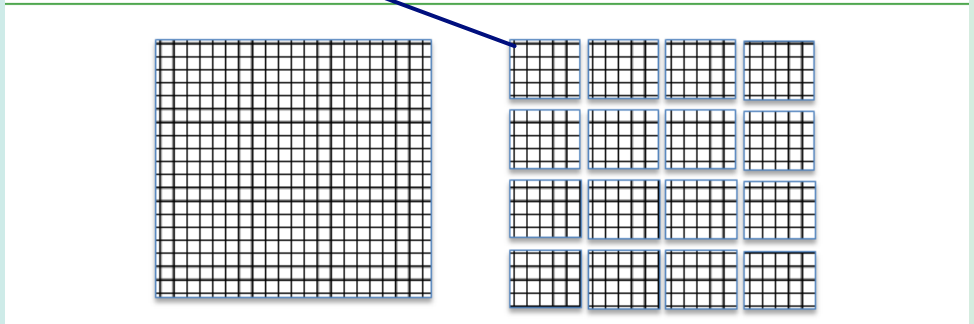
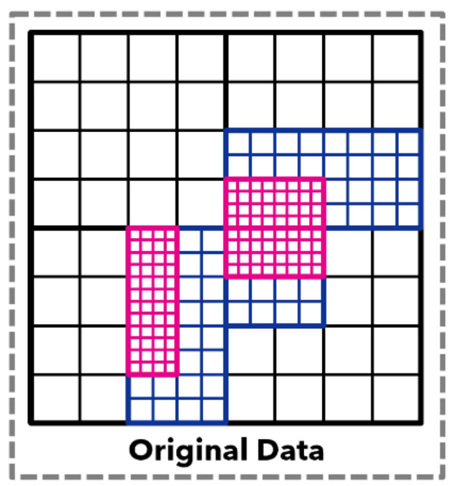


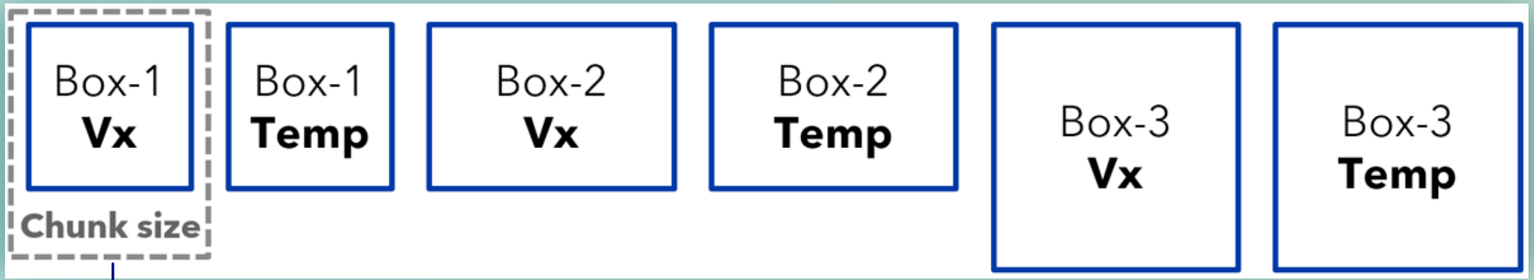
Figure 1: Data array is logically split into equally sized chunks each of which is stored separately in the file.

- Challenge: How to select the optimal chunk size for compression filters
  - Too small → low compression ratio & I/O perf
  - **We want a large chunk size!**
  - **Feature of AMR data is perverting us to use a large chunk**
    - Change AMR data layout, modify filter mechanism

Load Imbalance for AMR Data

- Challenge 1: AMR **data layout issue** for multiple fields
  - AMReX divides each AMR level's domain into boxes
  - Each box typically contains data from multiple fields.
    - Data of **one field** in different boxes are stored **separately**
  - Need to set a **chunk size** for the compression filter
    - The compression filter then processes the data chunk by chunk
  - Result in small chunk size thus **low compression ratio** & **I/O performance**



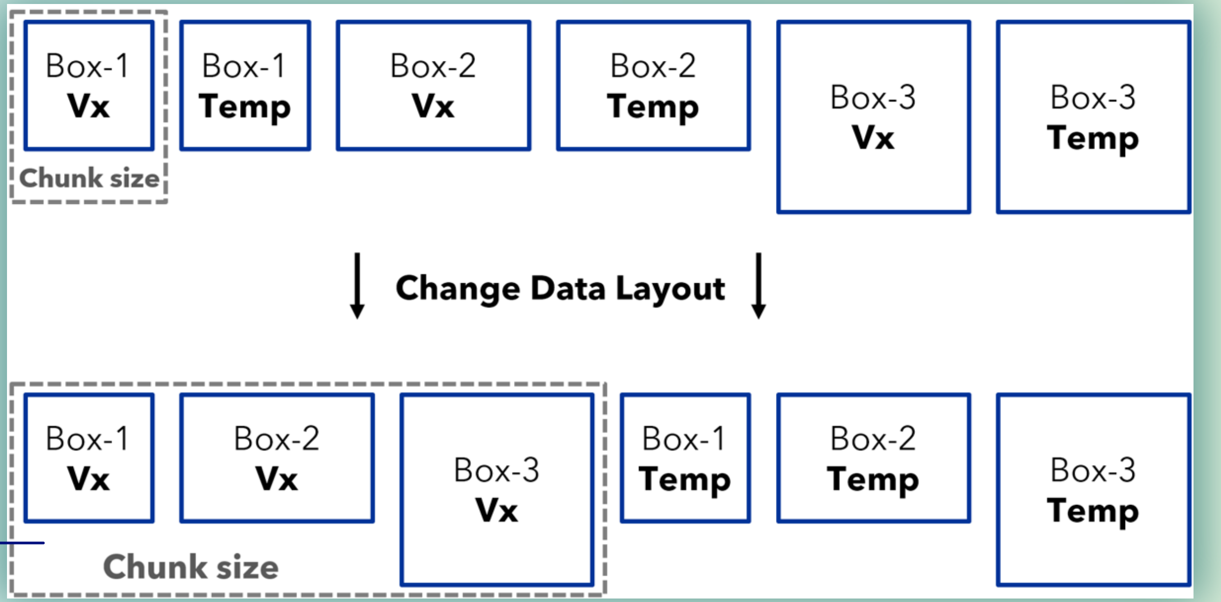


HDF5 chunk size cannot exceed the size of the smallest box (i.e., Box-1)

Load Imbalance for AMR Data

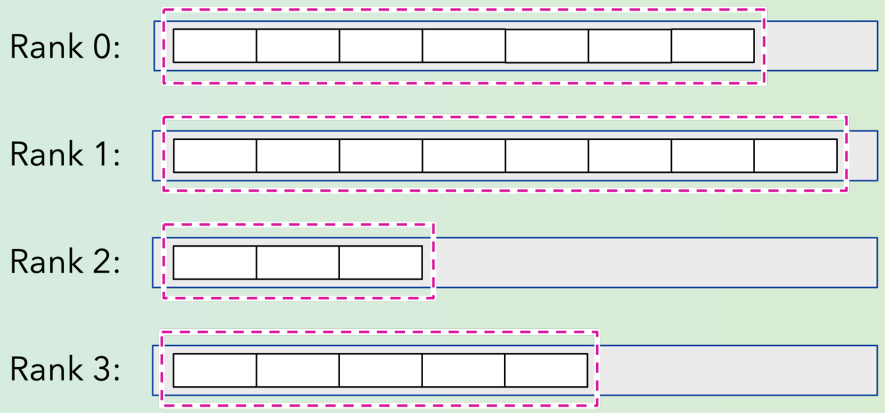
- Solution 1: Change data layout
  - Alter the loop access order when reading the data into the buffer, adds minimal overhead
    - Rather than reorganizing the buffer itself
  - Increase chunk size thus enhance both the **compression** and **I/O performance**

Larger chunk size that can compress the entire physical field together



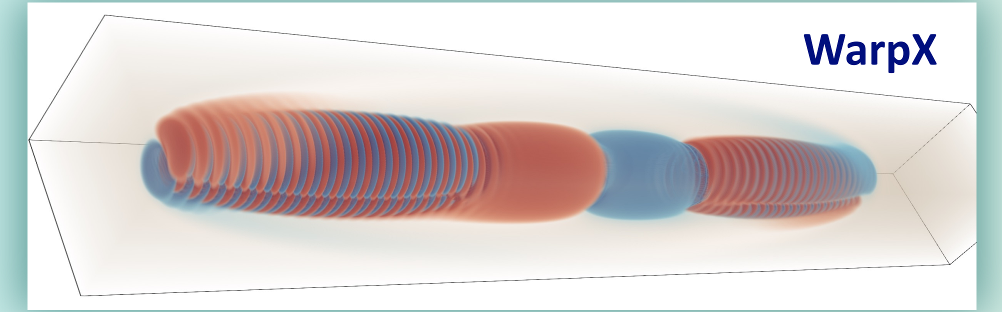
Modification of HDF5 Compression Filter

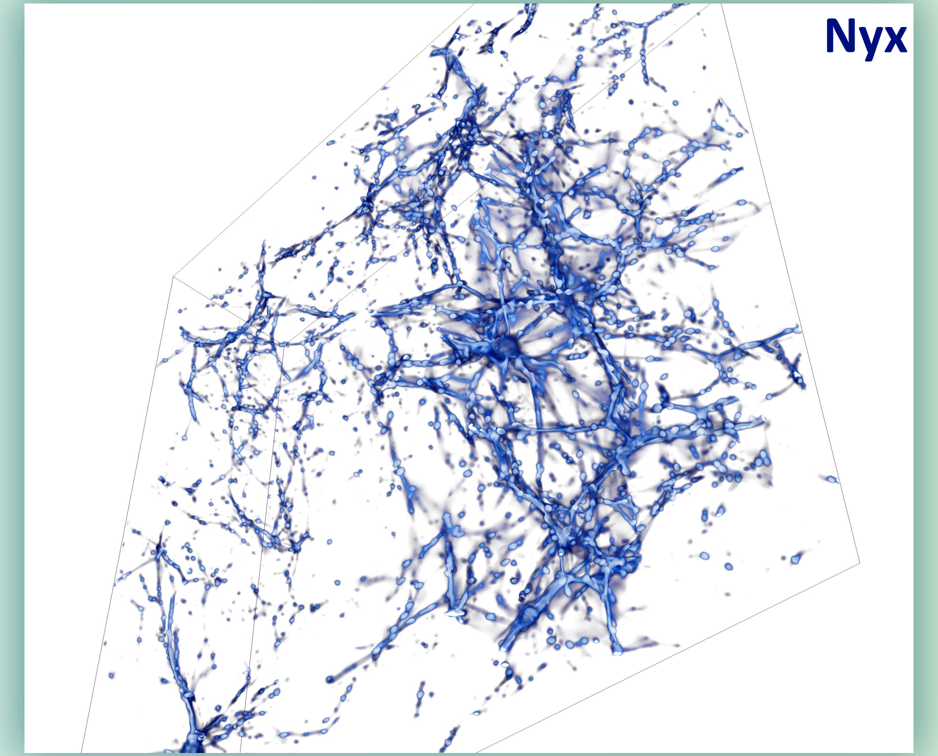
- Challenge 2: entire HDF5 dataset must use the same chunk size
  - Difficult to select an **optimal global chunk size** in **parallel scenario**
    - Data size on each MPI rank vary
  - Simply set $chunk\ size\ =\ \max(data\ size)$?
    - No, result in size overhead
  - Let each rank write its data to its own dataset?
    - No, result in **serial write** due to the collective write
- Solution 2: Modify the HDF5 filter mechanism
  - Still use $chunk\ size\ =\ \max(data\ size)$
  - But we will provide the real data size (the dashed pink box) of each rank to the filter



Evaluation Setup

- Two real-world AMR applications:
  - **Nyx** cosmology simulation
  - **WarpX** Particle-In-Cell (PIC) simulation
- Test platform: Summit supercomputer
  - Use up to 128 nodes and 4096 CPU cores;





Runs	#AMR Lvls	#Nodes (#MPI ranks)	Grid size of each level (coarse to fine)	Density of each lvl (coarse to fine)	Data size (per timestep)	Error Bound (AMRIC & AMReX)
WarpX_1	2	2 (64)	256*256*2048, 512*512*4096	98.04%, 1.96%	12.4 GB	1E-3, 5E-3
WarpX_2	2	16 (512)	512*512*4096, 1024*1024*8192	98.04%, 1.96%	99.3 GB	1E-3, 5E-3
WarpX_3	2	128 (4096)	1024*1024*8192, 2048*2048*16384	98.96%, 1.04%	**624 GB**	1E-4, 5E-4
Nyx_1	2	2 (64)	256*256*256, 512*512*512	98.6%, 1.4%	1.6 GB	1E-3, 1E-2
Nyx_2	2	16 (512)	512*512*512*, 1024*1024*1024	96.67%, 3.23%	12 GB	1E-3, 1E-2
Nyx_3	2	128 (4096)	1024*1024*1024, 2048*2048*2048	98.3%, 1.7%	97.5 GB	1E-3, 1E-2

Evaluation on Compression Ratio

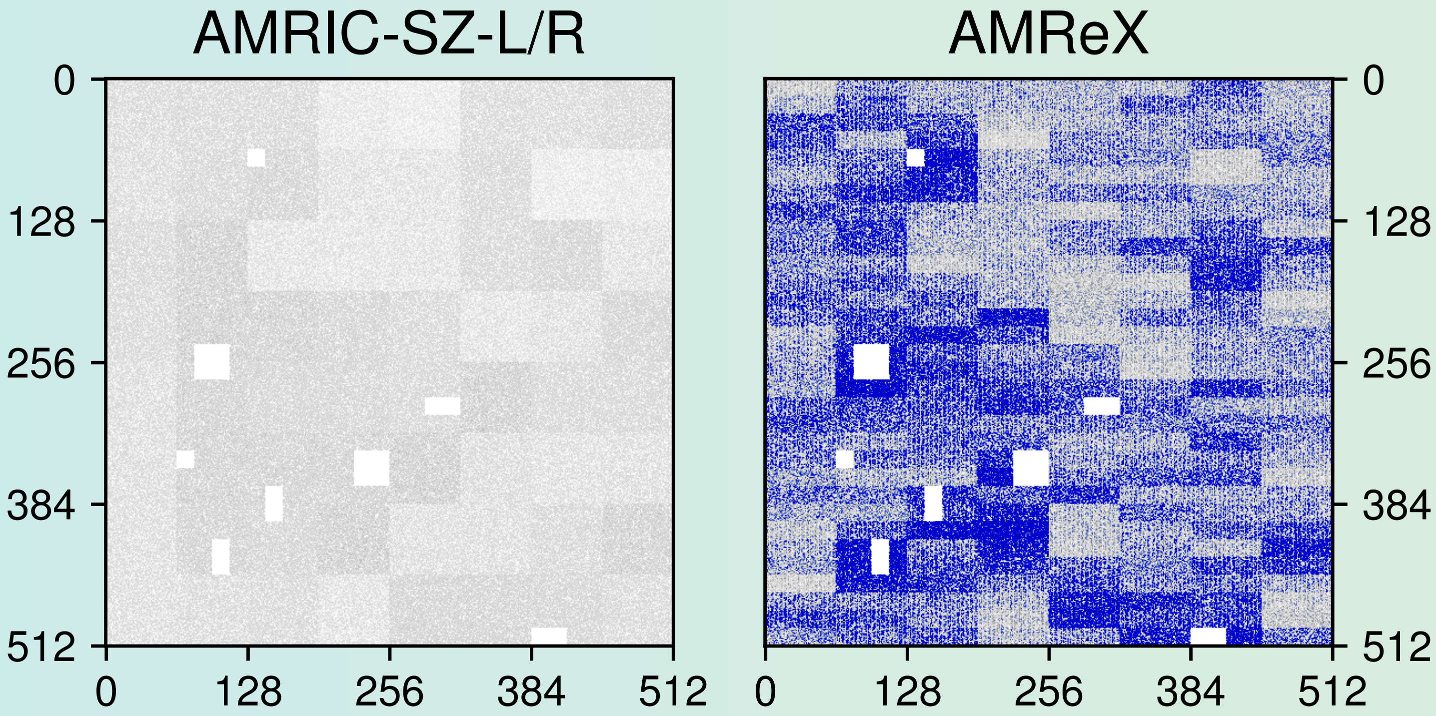
- Up to **81×** CR improvement over the ori AMReX's compression solution

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

Comparison of compression ratio with AMReX's original compression and AMRIC

Evaluation on Reconstruction Data Quality

- Data quality of AMRIC is also **higher** than that of AMReX



Vis (one slice) of compression errors of our AMRIC (left) and AMReX's compression (right) on Nyx. **Bluer means higher compression error**

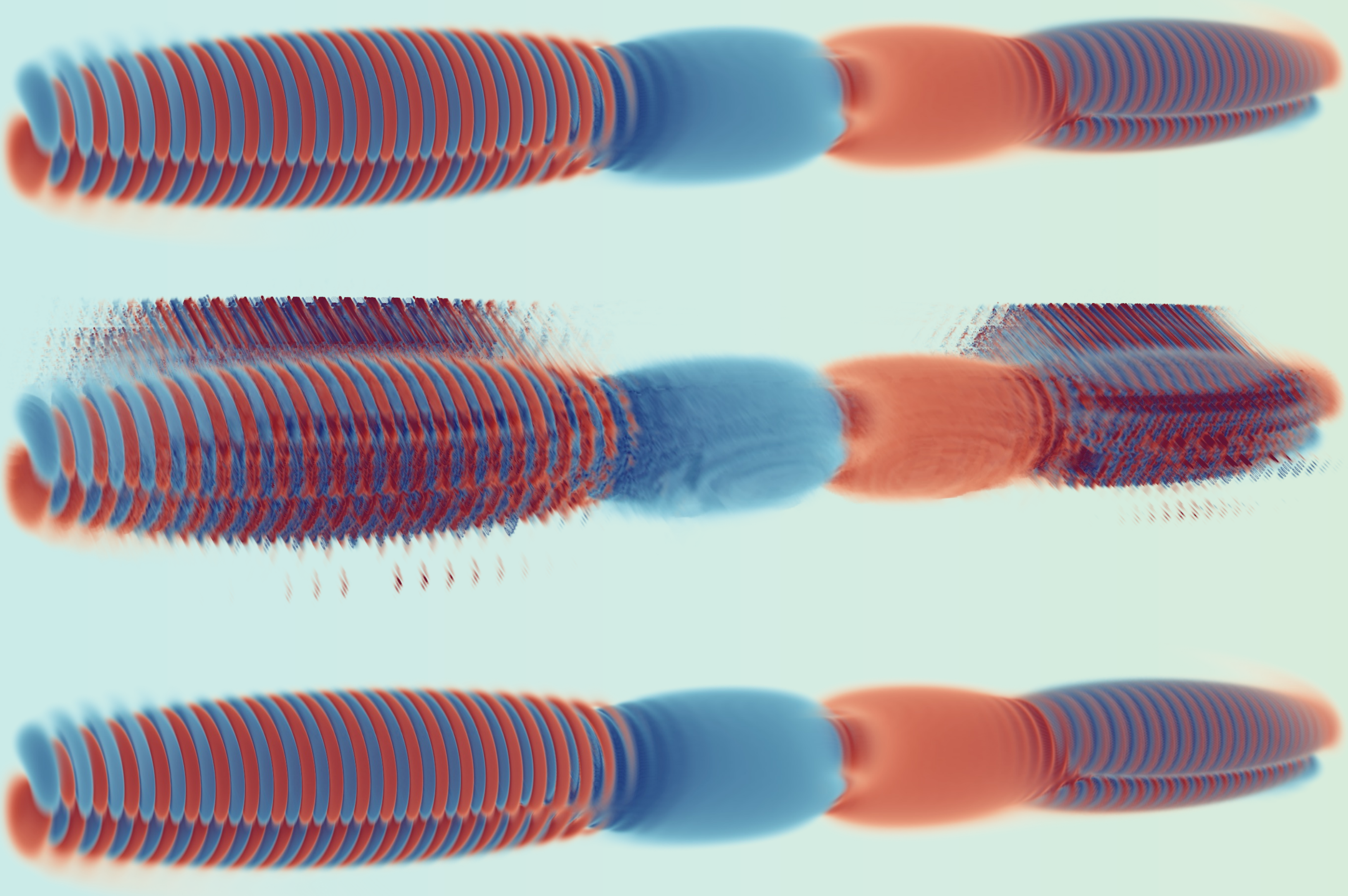
Evaluation on Reconstruction Data Quality

- Data quality (PSNR) of AMRIC is also **higher** than that of AMReX

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

Comparison of reconstruction data quality (in **PSNR**) with AMReX's original compression and AMRIC

Evaluation on Reconstruction Data Quality



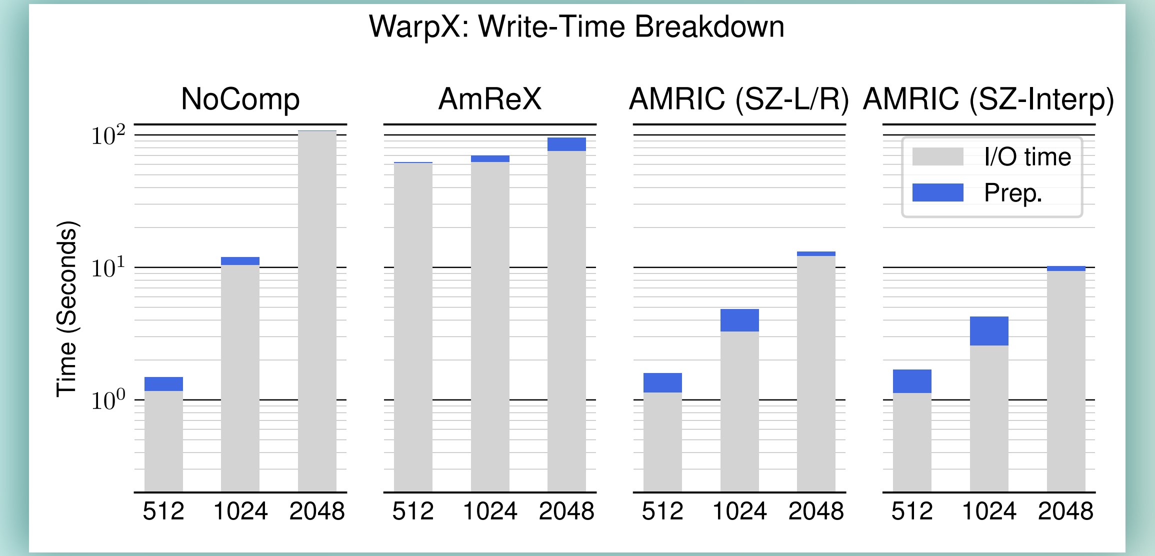
Original Data

AMReX
Compression Ratio = 19.0

Ours
Compression Ratio = 23.9

Evaluation on I/O Time

- Up to **10.5×** I/O performance improvement over the non-compression solution.
- Up to **39×** over the original AMReX's compression solution



Conclusion

- Propose insitu AMR compression AMRIC and integrating it into the AMReX
  - Design a compression-oriented in situ **pre-processing** workflow for AMR data
  - **Optimize** the SOTA SZ lossy **compressor** for AMR data
  - Efficiently utilizing the **HDF5 compression filter** on AMR data
- Compared to the non-compression sol: up to **10.5× I/O** performance improvement
- Compared to AMReX's compression sol: up to **39× I/O** improvement over & up to **81× compression ratio** improvement with better data quality

Thank you!

Questions 🤔 are welcome!

Contact: daocwang@iu.edu

ditao@iu.edu