LDRD Report - Variable precision computing

Combining IDX and zfp

At the beginning of this project, two prominent data reduction schemes in the literature are: IDX (resolution-based) and zfp (precision-based). We conducted several experiments to quantify the I/O tradeoffs among different methods to combine IDX and zfp. The most difficult challenge in combining these two schemes lie in the fact that coarse-resolution IDX data is not coherent, while zfp requires coherent data samples to achieve good compression. Conversely, coherent samples which facilitate good compression in zfp do not provide a good spatial coverage. Our work in this area includes:

- Studied two methods of combining zfp compression with IDX indexing, namely "block compression" and "brick compression". "Block compression" involves first organize the data samples in HZ order according to IDX, then compress each consecutive chunk of samples with zfp. This method incurs no I/O overheads for low-resolution queries but results in poor compression. The "brick compression" instead group each 4x4x4 block of samples into one unit, compress the unit with zfp, then organize these units in HZ order. This results in much better compression but incurs high I/O overheads at low resolution.
- To avoid the high I/O overheads of "brick compression", we have to devise a method to use all decompressed samples within a brick, but this introduces a problem of interpolating unevenly spaced data points: samples in the same brick are contiguous but samples in subsampled bricks are far apart. To tackle this problem we studied the interpolation method based on bi-Laplacian constraints. Although this method can work to some extent, we concluded that it is too unstable and too slow for our purposes.
- Proposed a novel idea in which each compressed 4x4x4 zfp block is separated into three "resolution levels": the DC coefficient, the 7 linear coefficients, and 56 higher-order coefficients. This approach avoids I/O overheads of the brick compression scheme, while retaining its high compression efficiency. We implemented a prototype for the previous idea and showed that the idea is scalable on supercomputers, is comparable to zfp in terms of reconstruction quality at low bit rates, and is able to achieve very high compression ratios (~2000x) while retaining reasonable data quality.

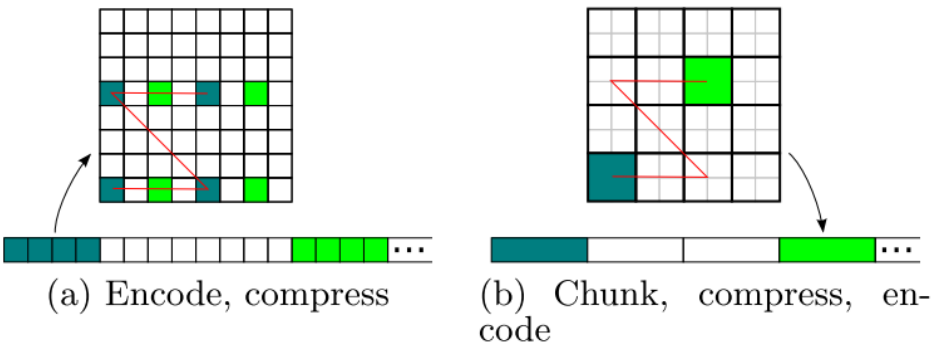


Figure 1. (a) "Block compression" versus (b) "block compression"

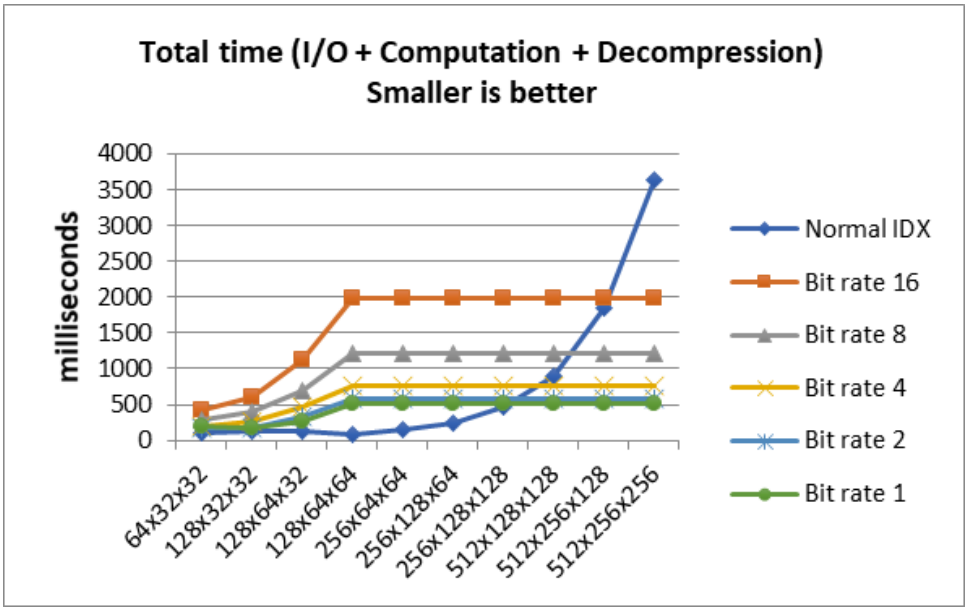


Figure 2. Brick compression at multiple bit rates against vanilla IDX for data reading at multiple resolution levels of a 3D, double-precision combustion simulation grid (the finest resolution is 512x256x256). We can see that in terms of data reading time, the brick compression scheme only beats regular IDX at low bit rates, and on relatively fine resolution levels.

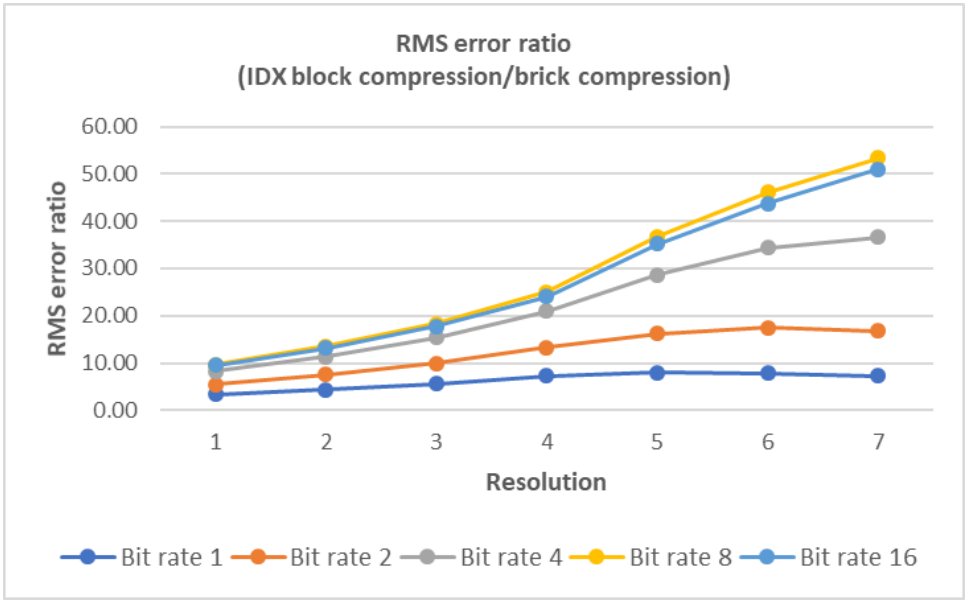


Figure 3. The block compression scheme incurs much higher error, especially for coarse resolution levels, compared to brick compression.

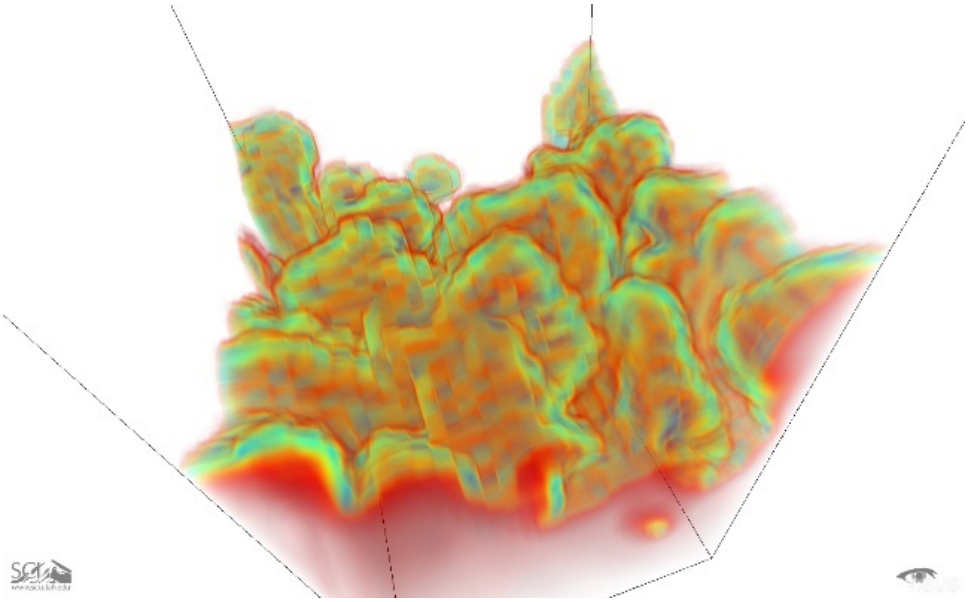


Figure 4. With brick compression, if no "smart" interpolation is used, simply placing 4x4x4 blocks into the low-resolution grid will result in blocky rendering artifacts.

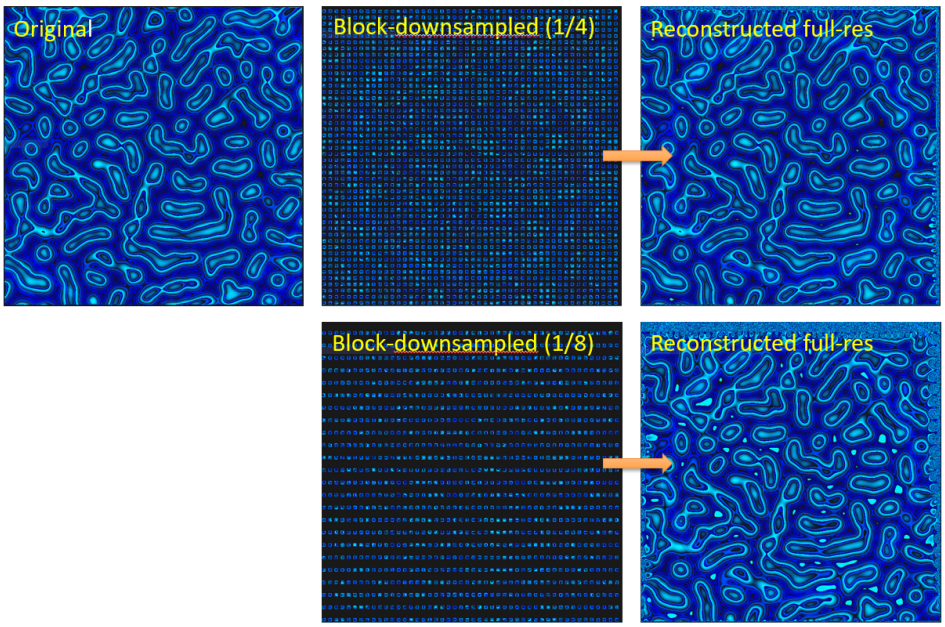


Figure 5. For this Miranda simulation (density field), we were able to approximate the function fairly well at 1/4 the final resolution, but the quality degrades quickly at 1/8 the final resolution. Furthermore, the linear system is highly ill-conditioned, the boundary cannot be reconstructed at all, and the system takes a long time to solve.

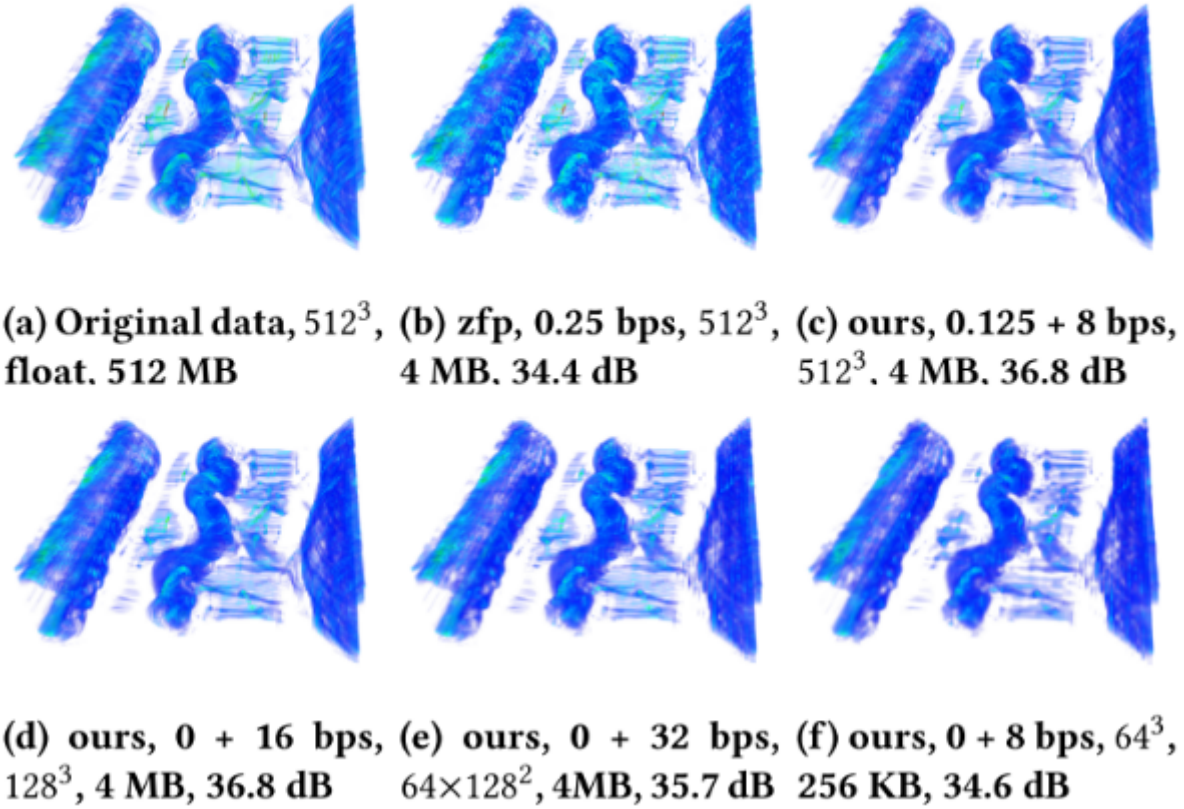


Figure 6. Comparison of different compression strategies at high ratios. CDF5/3 wavelet is used for downsampling. The output grids are upsampled to the original resolution using piece-wise constant upsampling when necessary. Our compression scheme is able to achieve good PSNR values at low bit rates (c and d). In (f) we further downsample the average field to a resolution of 64x64x64, by decompressing only the wavelet subbands up to that resolution, to achieve a 2000x compression ratio, while showing that it is still able to meaningfully reconstruct the data. Note that this is only possible through the combination of zfp compressionand wavelet downsampling. At this ratio, zfp alone in fixed accuracy mode reports a PSNR of 27.3 dB, while compression using wavelet alone (by keeping only the lowest-resolution subband) reports a PSNR of 33.5 dB, neither of which produces a rendering of reasonable quality.

Scaling I/O and in-situ analysis with reduced-resolution and compressed data

We performed several experiments to demonstrate the scalability of our data reduction techniques on supercomputers, as well as the feasibility of using such data for scientific analysis. Our contributions are:

- Implemented a distributed prototype for separated compression of the DC coefficients and the higher-order coefficients produced by the zfp transform, and demonstrated that at high core counts, the cost of compression is negligible compared to the communication and I/O costs.
- Made aggregation of coarse-resolution data scalable to up to 32K processes, with two optimization techniques: localized aggregation and domain partitioning, both of which constraint the communication to local regions of the network, which is key to scalability.
- Establish a direct link between HZ order (used by IDX) and the ordering imposed by traditional multi-level wavelet transforms. This means that the wavelet-based hierarchy can leverage our

infrastructures built for scalable I/O of IDX files.

- Showed that good results can be obtained for common topological data analysis techniques performed on data subsampled up to 4x in each dimension, and that the saving in time far outweight the loss of accuracy in analysis results.

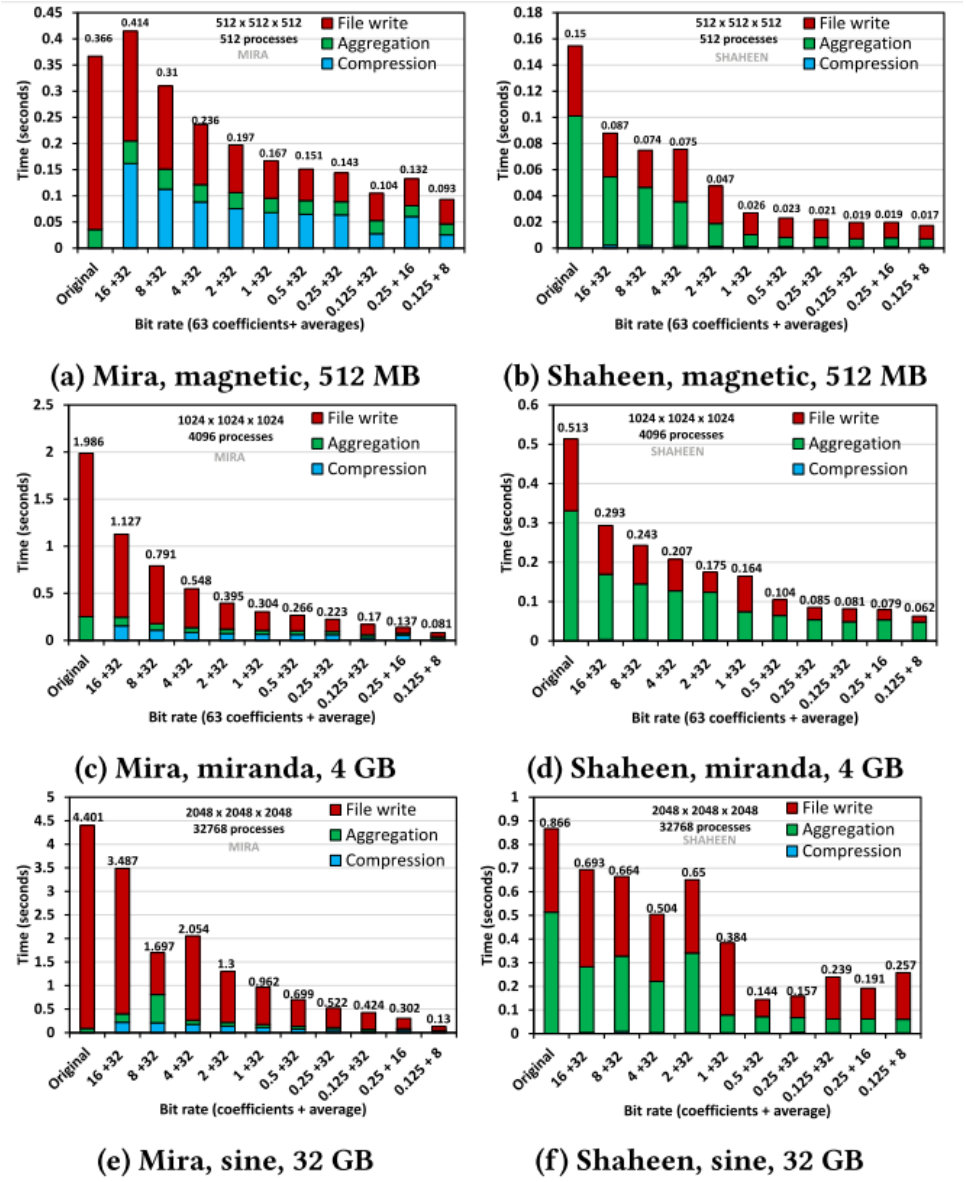


Figure 7. Weak scaling experiments with compression at different bit rates. Bit rate for the detail field varies from 0.125 to 16 and from 8 to 32 for the average field (32 menas uncompressed). Two levels of wavelet transform are applied to the average field prior to compression. At higher process counts, the cost of compression is negligible thanks to compression exhibiting a perfect weak scaling behavior compared to aggregation and I/O. Compression time is more dominant on Mira compared to Shaheen, likely due to a computationally weaker core. At core counts 512, 4096 and 32768, we observe best-case I/O speedups of 4x, 24x and 34x respectively on Mira and 9x, 8x and 6x respectively on Shaheen.

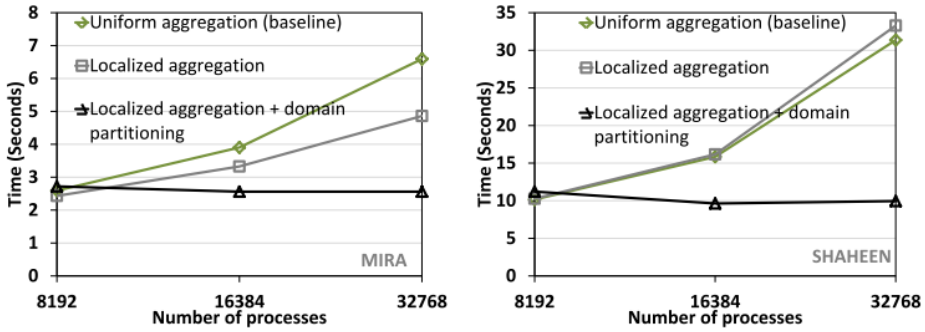


Figure 8. Efficacy of localized aggregation and domain partitioning. Per-process resolution is 32x32x32, 16 variables, using doubles. In domain partitioning, number of partition is 1 (8192), 2 (16384) and 4(32768). We compare our baseline aggregation scheme that uses uniform distribution of aggregators, with a second implementation that uses localized aggregation alone, and a third one that uses both localized aggregation and domain partitioning. It can be seen that the domain partitioning is necessary on both machines to get scalable performance. Localized aggregation brought no improvement on Shaheen II, as opposed to a 20% improvement on Mira. This can be attributed to the differences in network topologies of the two machines.

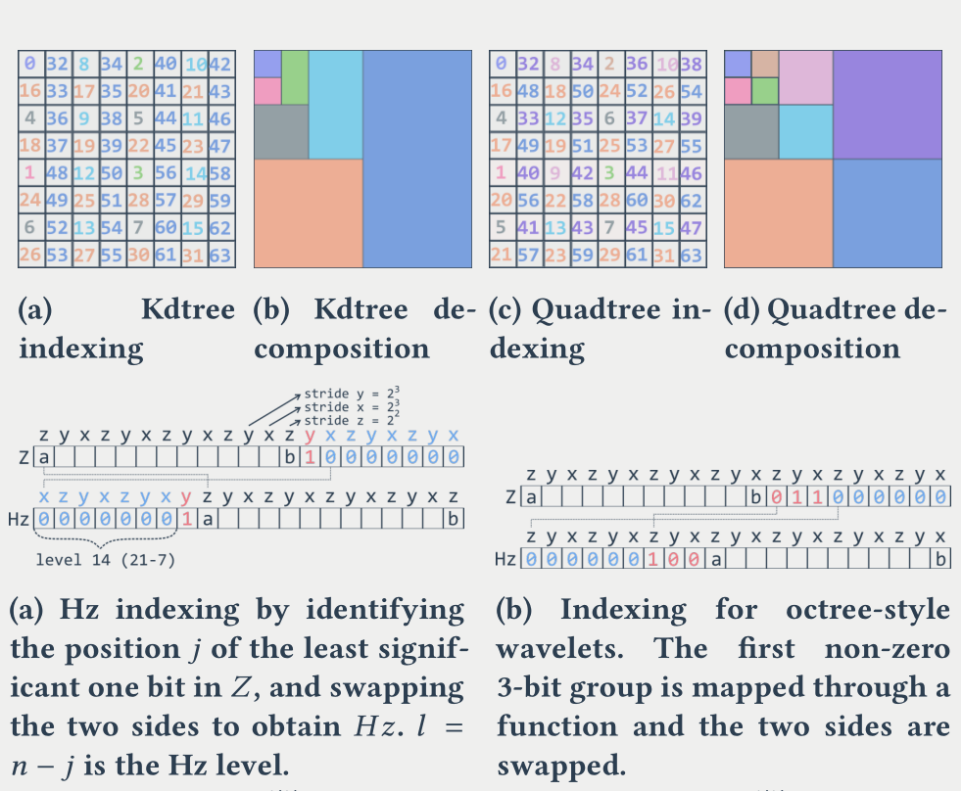
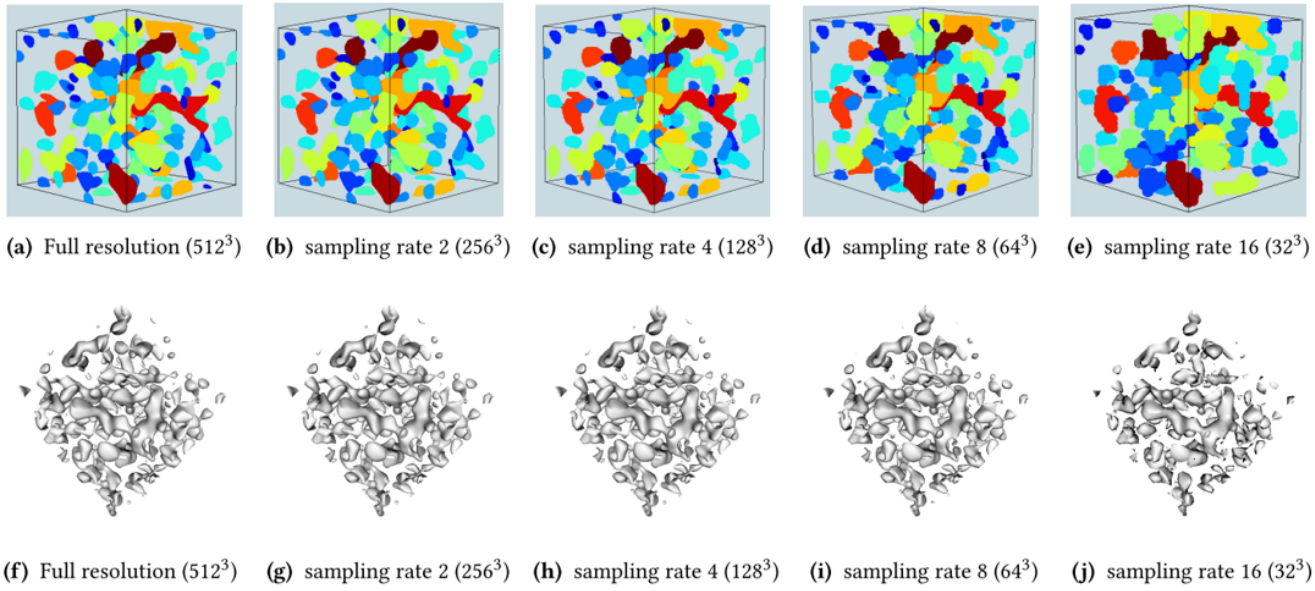


Figure 9. Correspondence between HZ indexing and wavelet's subband decomposition (top), and how to compute the indices (bottom), for both kdtree-style (left) and quadtree-style (right) variants.



(a) Full resolution (512^3) (b) sampling rate 2 (256^3) (c) sampling rate 4 (128^3) (d) sampling rate 8 (64^3) (e) sampling rate 16 (32^3)

(f) Full resolution (512^3) (g) sampling rate 2 (256^3) (h) sampling rate 4 (128^3) (i) sampling rate 8 (64^3) (j) sampling rate 16 (32^3)

Figure 10. Parallel merge tree (PMT) analysis algorithm ran on 512x512x512xfloat32 data set, on Tesla cluster (512 Xeon X5550 2.67GHz Processors) at the SCI institute of the University of Utah. We fixed the core count to 64, while changing the sampling rate from 1 (512x512x512) to 16 (32x32x32). PMT generates a merge tree, which is then used to perform segmentation. We observe that visually there is almost no loss in feature when down sampling from 512x512x512 to 128x128x128, after that we start to see a discernible loss of features. For example, at sampling rate of 8 and 16 we can see fewer features at top left corner of the dataset (d and e). We also observe a significant improvement (close to a factor of 8) in execution time as we go from full resolution (512x512x512) to 1/8th resolution (256x256x256). The rate of improvement slows down with higher sampling rate, mainly because total time starts to get dominated by the communication phase.

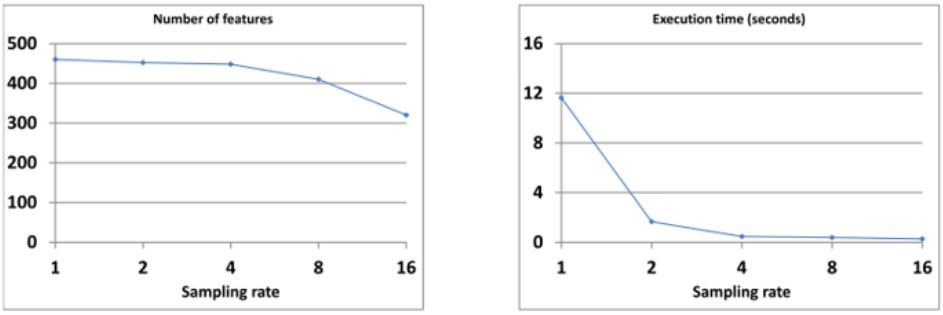


Figure 11. Actual number of features at all sampling rates. It can be seen that we incur a total loss of 140 features (460 at full data vs. 320 at sampling rate of 16) while going down in execution time from 11.5 seconds to 0.27 seconds. Overall, we observe that the time to completion for the PMT algorithm reduces significantly with sub-sampling, hence, also making it possible for them to be run in in-situ mode. Similar to PMT, we also observe small, describable differences among the iso-surface extracted at different sampling rates.

Resolution-precision tradeoffs for data analytics

Mixed-resolution and mixed-precision bit streaming in the wavelet domain

An alternative approach to combining zfp and IDX is the wavelet transform. Instead of IDX subsampling, the wavelet transform provides a hierarchy of resolution. Compression comes from the fact that in practice, the wavelet transform results in very small fine-level coefficients, which means the leading zero bits of these coefficients can be encoded more efficiently. Furthermore, the numbers of leading zeros of neighboring coefficients on the same subband are coherent. Assuming the wavelet coefficients are quantized to 16-bit integers, 16 wavelet subbands, and 32x32x32 block size for the purpose of encoding the leading zeros, the overhead of encoding the number of leading zeros is approximately 10 KB in 2D and 512 KB in 3D.

We combine this idea of encoding the number of leading zero bits for blocks of coefficients with a progressive streaming strategy where each bit is weighted based on its position and the associated wavelet basis function's squared norm (referred to below as "by importance static"). More important bits are streamed first. Comparing this scheme with traditional schemes such as "by levels" (where wavelet coefficients are streamed from coarse to fine), or "by bit planes" (where the coefficients are streamed from higher-ordered to lower-ordered bit planes), our "skip leading zeros" scheme results in much better PSNR.

We showed that our "skip leading zero" scheme also outperforms fixed-rate zfp compression in 2D, and is competitive with zfp in 3D, in terms of compression ratio. In 2D, we showed that both zfp and SLZ underperformed compared to JPEG2000. In the end, we decided on a hybrid scheme which combines the strengths of zfp, SLZ and JPEG2000: compressing individual blocks of wavelet transformed coefficients on each subband using zfp. The wavelet transform used by JPEG2000 and SLZ is responsible for decorrelating the data, while zfp's fast encoder can take advantage of any correlation left among neighboring wavelet coefficients. We showed later that this scheme beats JPEG2000 in both compression ratio and compression time, for both 2D and 3D.

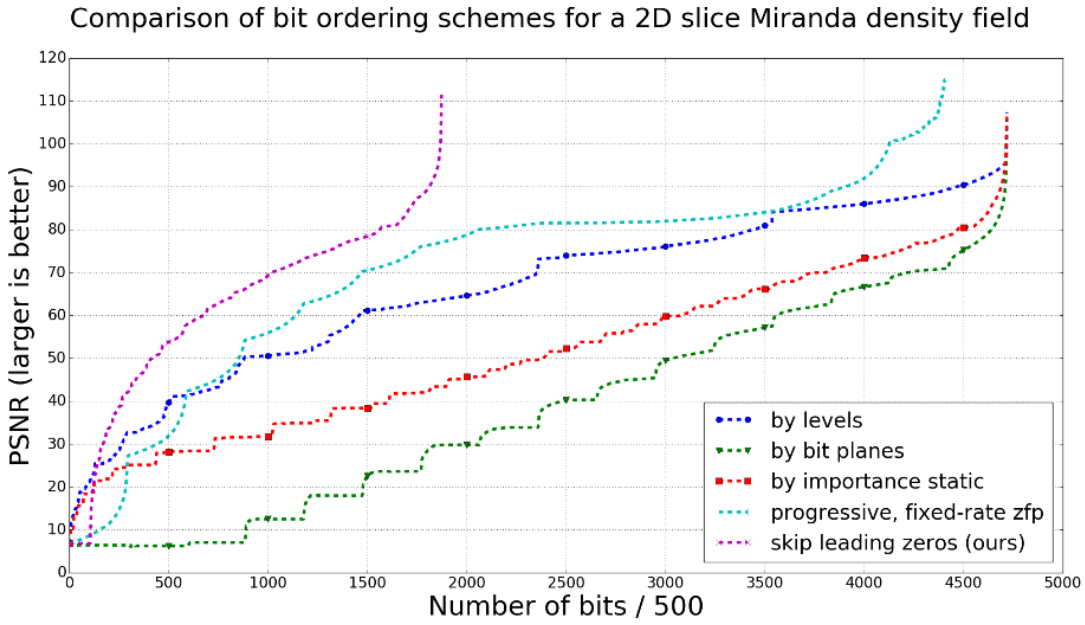


Figure 12. Comparisons of several bit streaming strategies for a 2D data set.

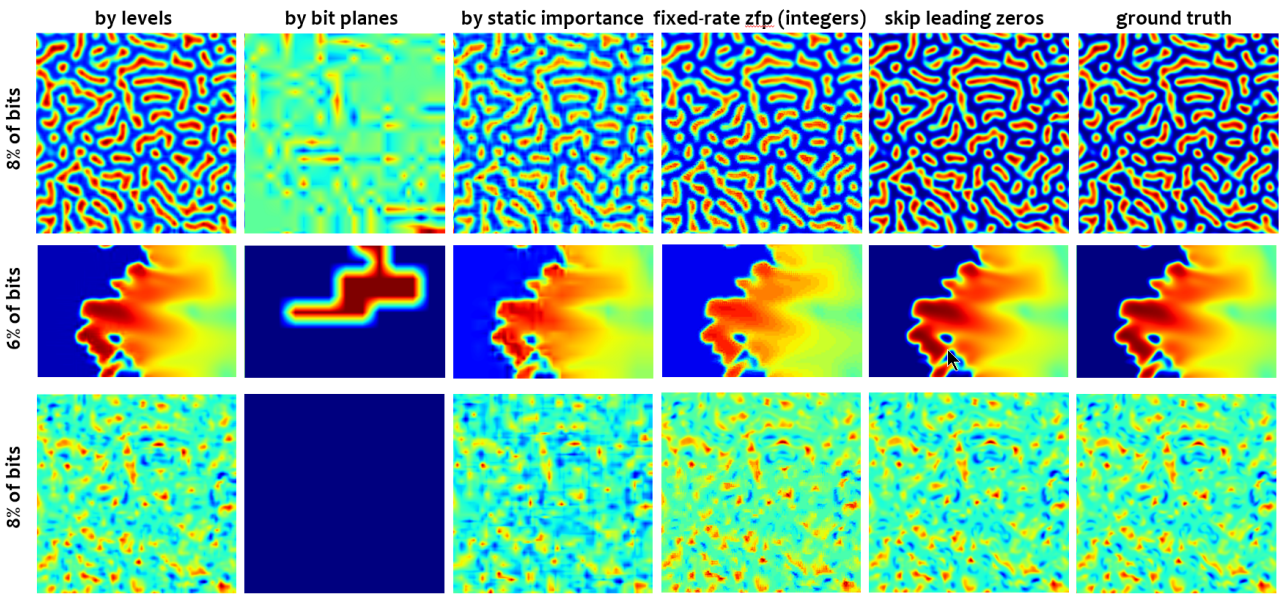


Figure 13. Comparing reconstructed data visually for all the streams (by levels, by bit planes, by static importance, fixed-rate zfp, and skip leading zeros). It can be seen that at 8% of the data size, our skip leading zeros scheme results in near-identical data quality compared to the ground-truth.

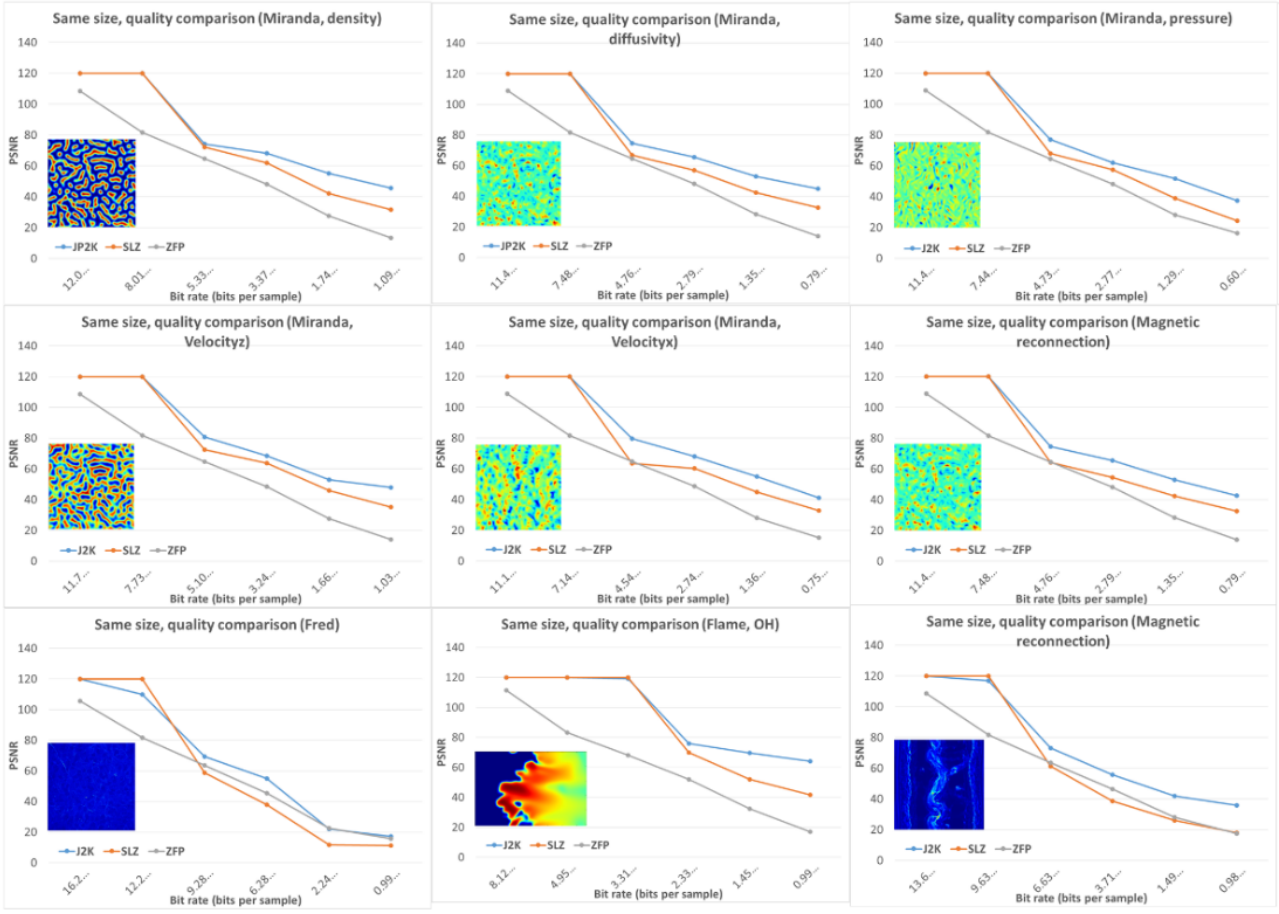


Figure 14. We compare SLZ, ZFP and JPEG2000, a state-of-the-art 2D compression algorithm. We compress the several 2D data sets at different bit rates, and compare the data quality in terms of PSNR (higher is better). JPEG2000 uses sophisticated entropy encoding, thus produces better compression at low bit rates compared to SLZ and ZFP. SLZ, on the other hand, is a relatively simple encoding scheme and thus can be implemented easily and very likely will result in faster encoding and decoding speed.

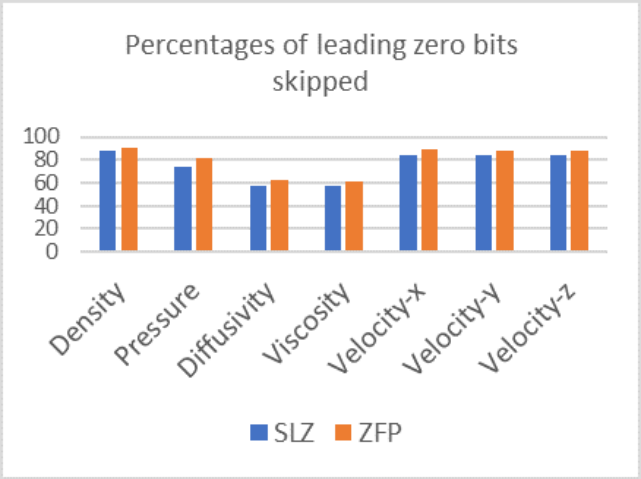


Figure 15. Percentage of leading zero bits skipped, by SLZ and ZFP. This comparison shows that SLZ is comparable to ZFP in terms of compression efficiency, averaged over seven 3D fields of a Miranda run.

Resolution-precision tradeoffs in the wavelet domain

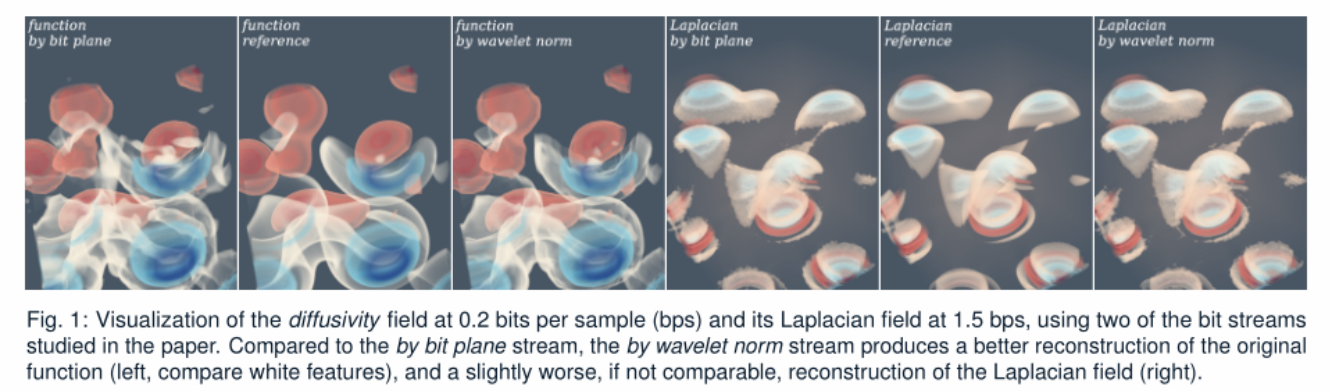


Figure 16. Visualization of the diffusivity field at 0.2 bits per sample and its Laplacian field at 1.5 bps, using two of the bit streams studied in the paper. Compared to the by bit plane stream, the by wavelet norm stream produces a better reconstruction of the original funciton (left, compare white features), and a slightly worse, if not comparable, reconstruction of the Laplacian field (right).

There currently exist two dominant strategies to reduce data sizes in analysis and visualization: reducing the precision of the data, e.g., through compression, or reducing its resolution, e.g., by subsampling. We explored the additional gains that could be achieved by combining both strategies. In particular, we present a common framework that allows us to study the trade-off in reducing precision and/or resolution in a principled manner. We represent data reduction schemes as a progressive stream of bits, and study how various bit orderings such as by resolution, by precision, etc. impact the resulting approximation error across a wide range of test data and analysis tasks. Furthermore, we compute streams optimized for different tasks, to serve as lower bounds on the achievable error. Scienti□c data management systems can use our results as guidance on how to store and stream data to make ef□cient use of the limited storage and bandwidth in practice. Our contributions include:

- A framework that allows systematic studies of the resolution-versus-precision tradeoffs for common data analysis and visualization tasks. The core idea is to represent various data reduction techniques as bit streams that improve data quality in either resolution or precision in each step. We can thus compare these techniques fairly, by comparing the corresponding data streams.

- Empirical evidence that jointly optimizing resolution and precision can provide signi□cant improvements on the results of analysis tasks over adjusting either independently. We present a diverse collection of data sets and data analysis tasks, and also show how different types of data analysis might require substantially different data streams for optimal results.
- A greedy approach that gives estimations for lower bounds of error for various analysis tasks. In addition, we also identify practical streams that closely approximate these bounds for each task, using a novel concept called stream signature, which is a small matrix that captures the essence of how a bit stream navigates the precision-versus-resolution space.
- A visualization tool that lets user interactively explore different streaming strategies and gain insights into each.

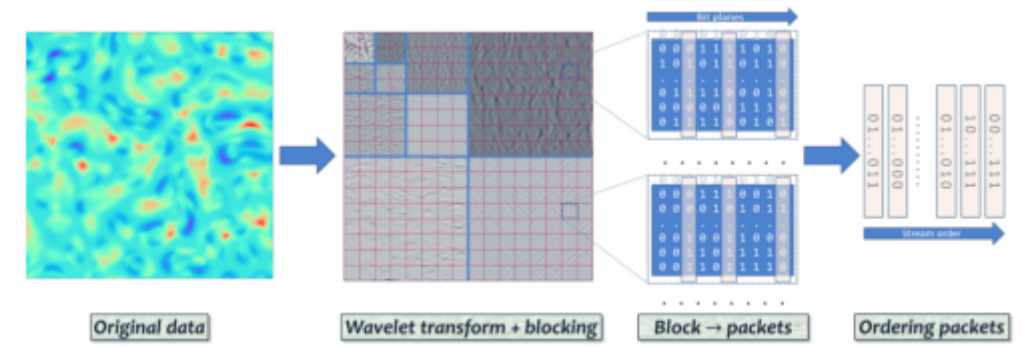


Figure 17. Our data streaming model. The input is a regular grid of floating-point samples; the output is a stream of packets. A packet consists of bits from the same bit plane, from a block of negabinary wavelet coefficients. Different data reduction schemes generate different streams. The wavelet subbands are separated by blue lines in the second image, with the coarsest subband at the top left corner.

We present the first empirical study to demonstrate that combining reduction in precision and resolution can lead to a significant improvement in data quality for the same data budget, and that different tasks might prefer different resolution-versus-precision trade-offs. For example, whereas computing histograms requires high precision, computing derivatives benefits more from higher resolution, and function reconstruction and isosurface extraction require a suitable mix of the two. We also show that common reduction techniques, e.g., those based on wavelet levels and coefficient magnitudes, do not perform well in the presence of entropy compression.

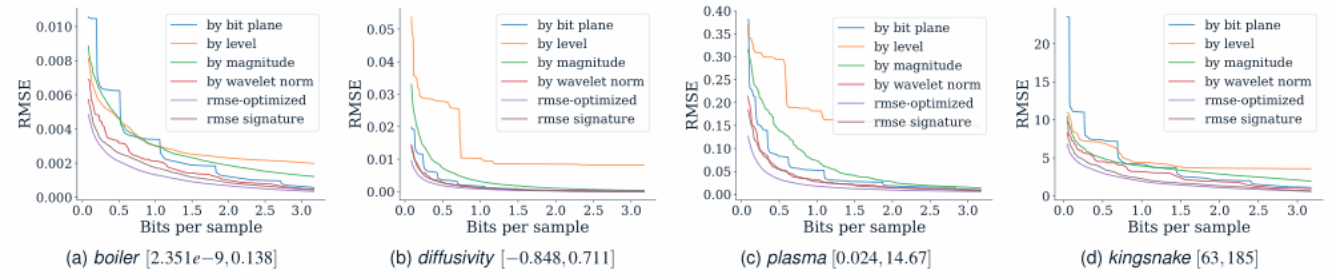


Figure 18. Root-mean-square error (RMSE) of reconstructed functions for different streams and data sets; lower RMSE is better. The streams are truncated at both ends to highlight the differences, without omitting important information. The numbers in bracets are the ranges of original data samples.

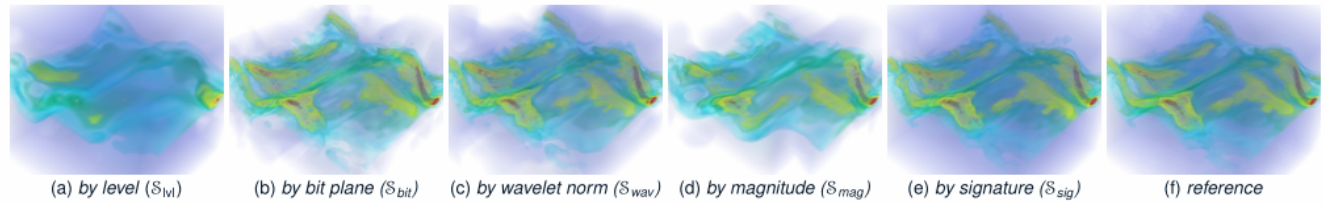


Figure 19. Volume rendering of a 64x64x64 region of plasma data set at 0.2 bps. Level-based stream captures the background (purple-blue) well, whereas bit-plane-based stream captures the fine details better. Wavelet-norm-based stream combines the strength of both. Signature-based stream, however, produces the most accurate rendering in overall.

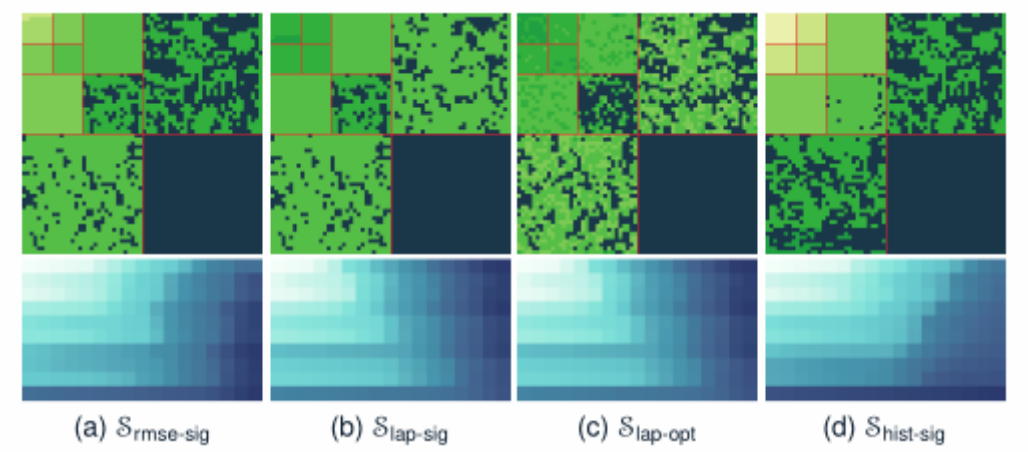


Figure 20. (Top) bit distribution across subbands at 1.6 bps for signature-based streams, and (bottom) corresponding stream signatures. Subbands are separated by red lines. The color of each pixel indicates the bit plane at which the corresponding coefficient currently is. Brighter greens corresponds to more precision. Both the top and the bottom rows show that histogram signature allocates more bits to the lower subbands, while Laplacian signature prefers to stream bits from higher subbands. RMSE signature is somewhere in the middle.

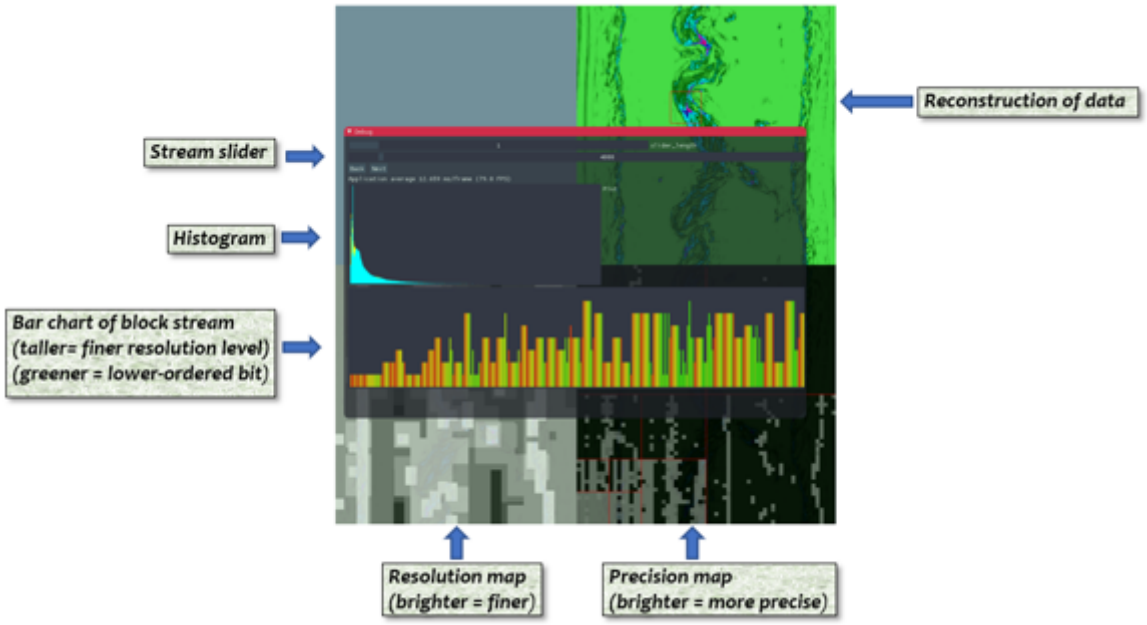


Figure 21. Our bit stream visualization tool.

Compared to data-independent streams, signature-based streams often perform better because they can adapt to the data. They are also amenable for implementation, since a signature is negligibly small and thus can be precomputed and stored during preprocessing. Here, we consider signature-

bases streams to be the best possible stream that could be realized. Improvements on these streams in the resolution-versus-precision space are likely possible, but they are unlikely to be significant. Given these assumptions and the fact that signature-based streams in most cases provides very similar results to bit-plane-based or wavelet-norm-based streams, the additional effort (and potential overheads) for task-dependent bit orderings is unlikely to be beneficial.

This leaves a significant gap between the best data-independent streams and the optimal stream. Our experiments suggest that the majority of these differences can be attributed to spatial adaptivity. The prototypical example is isosurface computation, where the optimal streams can skip all regions that do not affect any portion of the surface. It may be worthwhile in future work to investigate solutions to spatial adaptivity to significantly improve the performance of data-independent streams.

Resolution-precision tradeoffs for IDX-based hierarchies

For IDX-based multiresolution hierarchies (as opposed to wavelets), we proposed a linear programming algorithm to determine the best resolution and precision level for each spatial block, given a data budget. Using this algorithm, we computed and compared different query paths (in the resolution/precision 2D space) for different quantities of interest.

We hypothesized that different queries will result in very different refinement patterns. Surprisingly, many queries (such as RMSE and segmentation) result in almost identical data streams suggesting that there are only few classes of streams. This result may be applicable when designing a static stream order for local data (such as in data blocking) where depending on the query class a particular order would be selected.

In general, our findings here for IDX-based hierarchies agree with our results for the wavelet-based hierarchies explored in our IEEE TVCG paper.

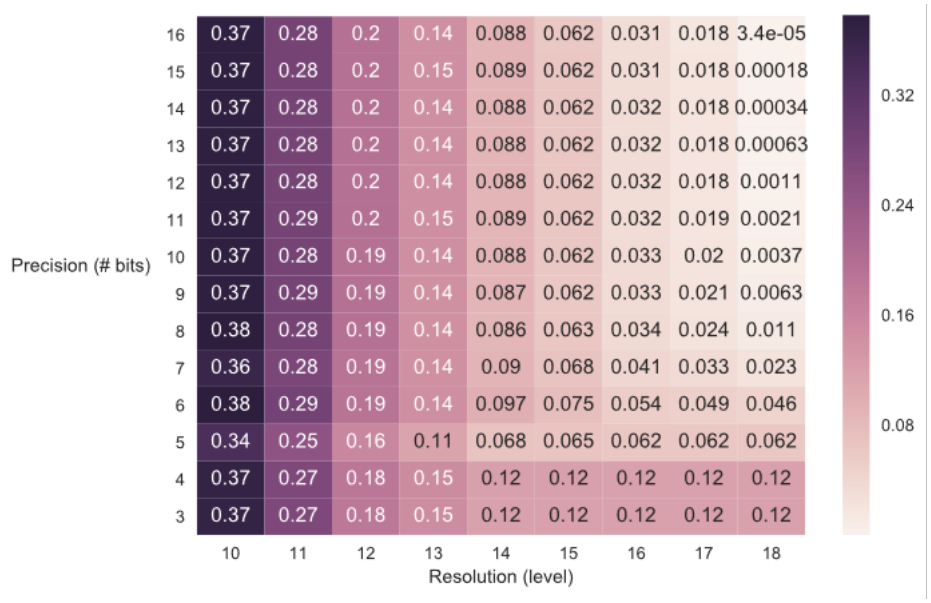


Figure 22. Error matrix that on x-axis has the resolution levels, on y-axis precision levels, and the values inside its cells are the individual errors at that particular resolution/precision with respect to the full resolution/precision dataset. We tile the dataset and vary resolution within each tile independently. For each tile we compute the error matrix and then construct an integer linear program (ILP) to find the optimal bit distribution given a bit budget.

Improvements in wavelet transform and wavelet compression

Tile-based wavelet transform

We implemented tile-based wavelet transform, where the transform happens one tile at a time, instead of a big transform done on the whole volume. The goal is to reduce the potentially huge memory requirement when processing very big volumes which might not even fit in main memory. The tiles are of fixed size (e.g. 32x32x32 in 3D), but each tile is padded by one voxel in each dimension to make sure the transform computes the same values as the global transform does. The new transform is mostly the same as before (we use lifting), but special treatment is required at the voxels at cell boundaries so that we do not double-count the values in these voxels.

We have tested the algorithm it on a volume of size 384x384x256xfloat64, and observed not only a reduction in memory usage but also a reduction in transform time: the new approach takes 700ms, while the traditional approach takes 1900ms to transform this volume. The faster speed is likely due to the better use of the cache (a tile can usually fit in L2 cache).

Finally, the implementation uses dynamically scheduled tasks (each tile corresponds to a task) that can easily scale (with minimal code changes) from desktop computers, where each task maps to a thread, to supercomputers, where tasks can be scheduled to run remotely.

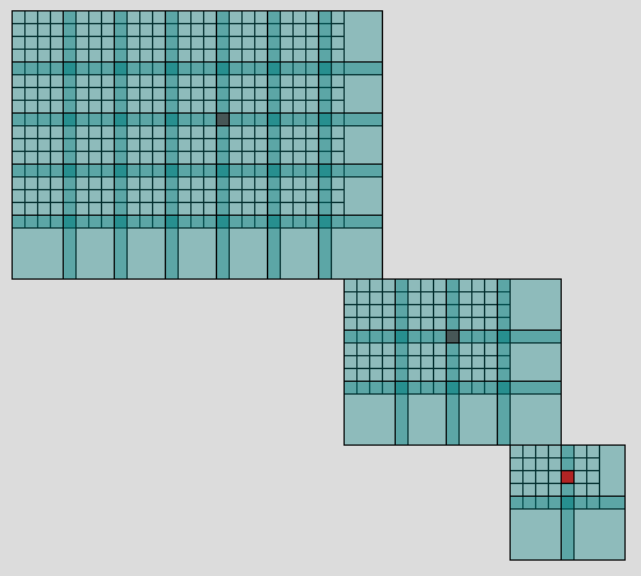


Figure 23. Visualizing our tile-based transform. The bright red pixel marks the currently selected pixel on the second level, while the dark red pixels each corresponds to the selected pixel on the next coarser level. Here we show only the LL subband for each level; in 2D there are three more subbands, namely LH, HL and HH. On each level, a tile depends on a few neighboring tiles on the previous, finer level. This number depends on the location of the tile as well as on the subband. A regular tile on subband 0 will have 64 dependencies in 3D and 16 in 2D.

Distributed wavelet transform

We implemented a distributed wavelet transform (for CDF5/3 wavelet), meant for building a resolution hierarchy out of the DC coefficients. We showed that the algorithm scales perfectly to 64K processes on the Mira supercomputer.

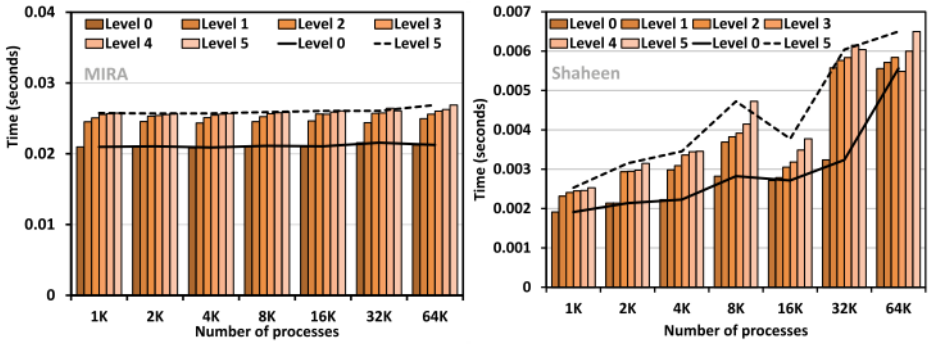


Figure 24. Weak scaling results of wavelet computation on Mira and Shaheen. Per-process load: 64x64x64xfloat32 (1 MB). Total amount of data varies from 1 GB at 1024 processes (512x512x512) to 64 GB at 65536 processes (4096x2048x2048). Number of wavelet levels in each dimension varies from 1 to 6. At each core count, the time taken to compute wavelet coefficients becomes larger with increasing levels, however, with a diminishing rate of growth. This is because total amount of stencil data needed by any process decreases inversely with the square of the level and the computation involved decreases inversely to the cube of the level. Also, Mira exhibits perfect scaling as opposed to Shaheen where total time taken to compute a wavelet level increases with core counts. We note that the parallel wavelet transform is rougly 150 times faster that actual I/O on both Mira and Shaheen.

Linear-lifting based extrapolation at the boundary

In the lifting scheme for the wavelet transform, the w-lift phase predicts the values at odd-indexed vertices. When the length of the (1D) input signal is even, the wavelet coefficient at the last vertex cannot be predicted correctly since an adjacent neighbor is missing. Common solutions to this limitation include assuming the adjacent neighbor to be zero, or mirroring the function by duplicating the penultimate value. However, both these approaches result in functions that are discontinuous or nonsmooth at the boundary, exaggerating the magnitude of the last wavelet coefficient, which result in unnecessary refinement in and significantly (and needlessly) inflate the memory footprint.

We introduced linear-lifting approach to extend the input function at the boundary. In particular, in the w-lift step if the function length is even, we extrapolate the data linearly at the boundary, to maintain smoothness across the boundary, because the last wavelet coefficient (in place of b) becomes zero. Our linear-lifting approach intersperses the linear extrapolation steps (introducing at most one extrapolated value at each step) with the lifting steps across hierarchy and across spatial dimensions.

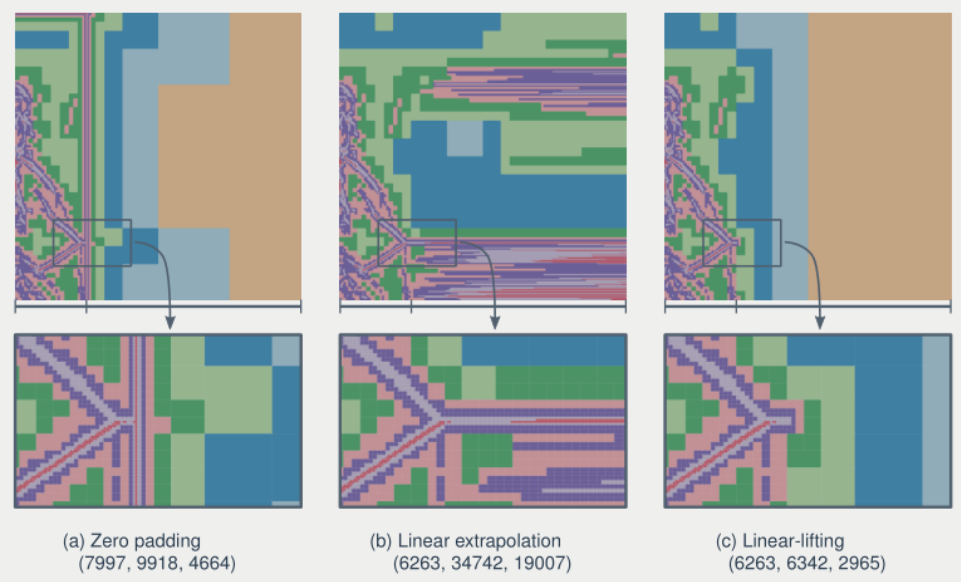


Figure 25. Comparison of different extrapolation modes for a function defined on [256×1024] domain to [1025×1025] (denoted by marks on the horizontal line). The data is colored with respect to the hierarchy levels of cells. Zero padding introduces artificial discontinuities at the boundary of the input domain (notice the red streak of finest-level cells), resulting in unnecessary refinement. Linear extrapolation maintains smoothness near the boundary, but may create discontinuities farther out from the original domain. Linear-lifting ensures smoothness in the entire domain, preserves the wavelet coefficients inside the original domain, and suppresses their values in the extrapolated region. The associated metrics are number of (cells, leaf nodes, internal nodes).

Wavelet compression using zfp

We compared the performance of zfp and EBCOT (the JPEG2000 wavelet encoder) in both speed and compression ratio. We carefully augmented the source code of OpenJPEG to measure only the relevant pieces of code for its encoder/decoder, to ensure as much as possible an apple-to-apple comparison with zfp.

The results show that zfp is about 10x to 20x faster than EBCOT for both encoding and decoding wavelet coefficients. Furthermore, the compressed data size is about 10% smaller with zfp. These results help justify our decision for using zfp as the compressor of choice for our file format.

```
-- Velocityz (384 x 384 x float64)
JP2K:    131,529  |     54 |     21
ZFP :    108,562  |      6 |      5

-- Diffusivity (384 x 384 x float64)
JP2K:    132,622  |     32 |     19
ZFP :    114,702  |      7 |      5

-- Density (384 x 384 x float64)
JP2K:    116,727  |     29 |     20
ZFP :     97,481  |      6 |      5

-- Pressure (384 x 384 x float64)
JP2K:    149,048  |     30 |     20
ZFP:     125,522  |     12 |      6

-- Viscosity (384 x 384 x float64)
JP2K:    132,486  |     27 |     11
ZFP :    114,572  |      6 |      5

-- Flame (256 x 256 x float64)
JP2K:     18,567  |     13 |     10
ZFP :     19,243  |      3 |      2

-- Turbulence (256 x 256 x float32)
JP2K:     58,364  |     12 |      9
ZFP :     53,414  |      3 |      2

-- Magnetic reconnection (512 x 512 x float
JP2K:    310,117  |     60 |     46
ZFP :    278,220  |     11 |     10

-- NEURONS (512 x 512 x uint16)
JP2K:    360,350  |    104 |     63
ZFP :    339,052  |     12 |     11
```

```
-- Velocityz (384 x 384 x 256 x float64)
JP2K: 24,100,307  | 10,787 |  9,756
ZFP : 17,442,531  |    735 |    525

-- Diffusivity (384 x 384 x 256 x float64)
JP2K: 14,550,350  |  7,879 |  6,973
ZFP : 15,236,876  |    760 |    429

-- Density (384 x 384 x 256 x float64)
JP2K: 18,324,131  |  9,368 |  8,149
ZFP : 16,848,121  |    685 |    490

-- Pressure (384 x 384 x 256 x float64)
JP2K: 19,989,981  |  9,917 |  9,140
ZFP:  16,834,358  |    727 |    485

-- Viscosity (384 x 384 x 256 x float64)
JP2K: 14,544,463  |  7,690 |  6,886
ZFP : 15,226,242  |    649 |    414

-- Flame (512 x 256 x 256 x float64)
JP2K: 16,943,701  |  8,329 |  7,081
ZFP : 14,924,359  |    577 |    416

-- Turbulence (256 x 256 x 256 x float32)
JP2K: 25,633,403  |  9,393 |  8,772
ZFP : 22,368,699  |    442 |    405

-- Magnetic reconnection (512 x 512 x 512 x float32)
JP2K: 252,801,471 | 77,294 | 67,874
ZFP : 217,135,890 |  2,614 |  2,949

-- NEURONS (512 x 512 x 512 x UInt16)
JP2K: 176,348,108 | 62,718 | 55,073
ZFP : 142,012,486 |  2,168 |  2,201
```

Figure 26. zfp versus JPEG2000 as wavelet encoders in (left) 2D and (right) 3D. The first column is the size of the compressed data, the column is the encoding time, and the third column is the decoding time.

Fast zfp decoding with AVX2 instructions

We made the serial bit transposer of zfp faster with the use of AVX2 instructions. This results in 2x improvement in speed for the decoder. We later learned of another approach to transposing the bit matrix, using recursion, which is even faster. We are investigating the potential use of SIMD instructions to improve the recursive bit transposer.

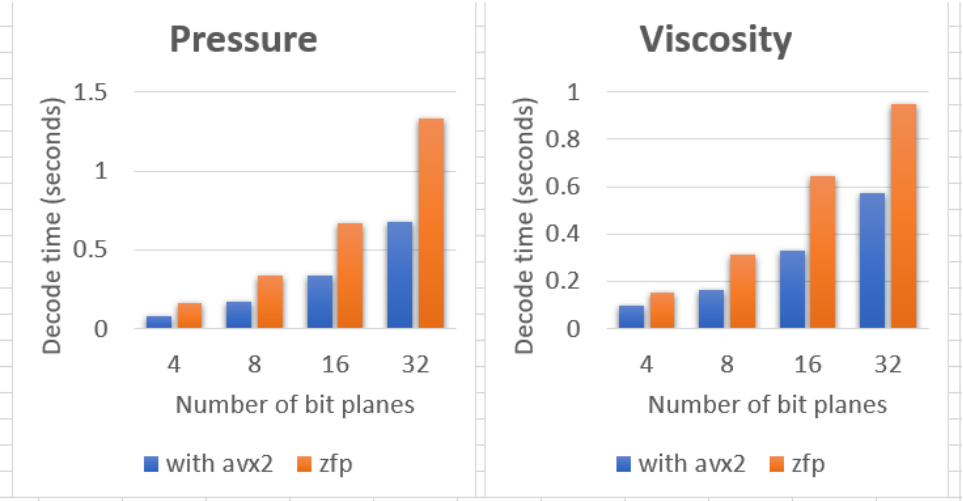


Figure 27. Timing comparison of serial zfp decoder vs AVX2 variant.

File format supporting local queries that improve resolution and/or data precision

Compact in-memory adaptive-resolution data structure

We have devised a compact in-memory representation for adaptive-resolution and adaptive-precision data. It consists of a grid of blocks, where the blocks can have different resolutions. At the finest resolution, every block will occupy a fixed-size region (e.g. 64x64x64 samples). New samples are added in the same way that a wavelet hierarchy works: an "odd" sample will be inserted between two "even" samples, doubling the size of the grid in each dimension.

To accommodate adaptive precision, we assume that samples within a block share the same precision, but across blocks the precision can vary. As a result, we store per-block samples compressed using zfp in fixed precision mode. We keep pointers to all compressed blocks, and allow a block to be totally empty, for example, to represent empty space in the volume.

We have implemented a visualization prototype for this structure in 2D, as shown below.

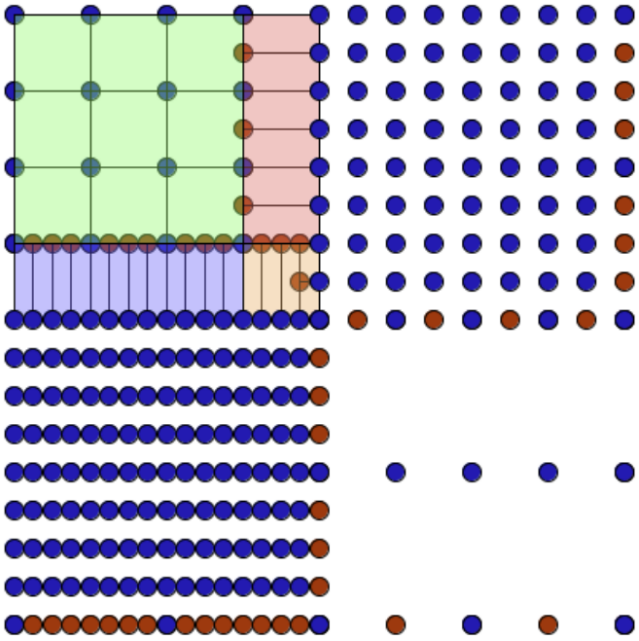


Figure 28. Four neighboring blocks are shown, of size 4x4 (top-left), 8x8 (top-right), 16x8 (bottom-left), and 4x2 (bottom-right) respectively. Note that we only store blue samples in green regions; the pink, purple, and orange regions are "bridges" between blocks, and are reconstructed on the fly as needed. Red samples are not stored, they are interpolated from blue ones as needed. Samples within a block (and within a colored region) always form a regular grid.

Compared to a fully adaptive grid, our block-based data structure has limited spatial adaptivity. If a block is too big, we will waste memory when representing fine-scale features. On the other hand, a too-small block size will be less efficient in representing large but coarse structures, as well as limit our freedom in resolution reduction. We choose 64x64x64 as the ideal block size.

File format schematic

To allow multi-resolution data reading, we transform the original data using linear B-spline wavelets. This transform partition the domain into a set of subbands, each corresponds to a resolution level. We further partition each subband into a set of non-overlapping, fixed-size tiles. For example, each tile can be of size 32 x 32 x32. We compress each tile independently using zfp, with a modification to allow interleaving of compressed bits across zfp blocks (which are of size 4 x 4 x 4). Higher-order bits are compressed before lower-order ones. Compressed bits are stored in chunks of the same size. For example, a chunk can be 4KB in size. The chunks are our smallest unit of I/O, to amortize disk access overheads. In this way, each tile is compressed into potentially several chunks. The following figure demonstrates our compression and chunking scheme.

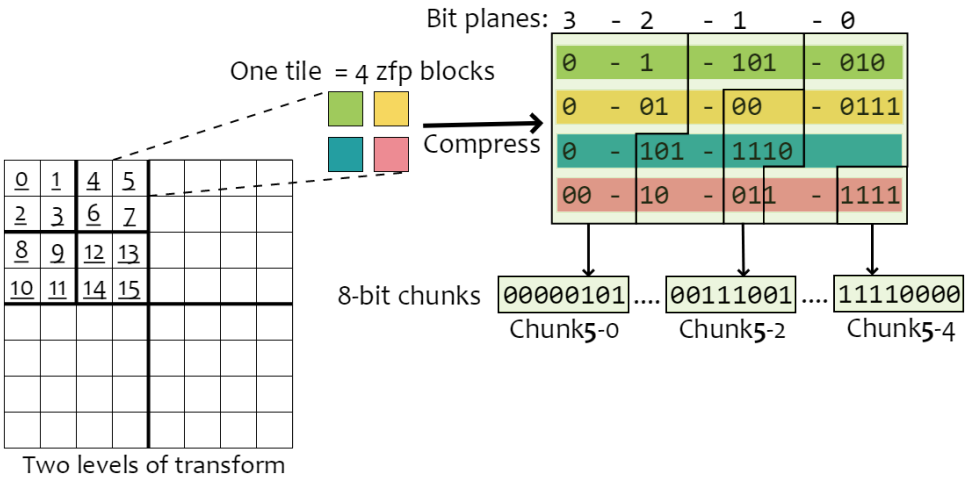


Figure 29. The forming of data chunks.

The chunks are further grouped into files. Our research has shown that having either too few or too many files result in scaling issues on today supercomputers. Therefore we let the user choose the number of tiles to be grouped together in a file. Tiles are first grouped into groups of seven, across all subbands, the rationale being the seven tiles all cover the same spatial region when transformed from the wavelet back to the original domain. In 2D, tiles are processes in groups of three instead of seven (see the figure below, noting that tiles 4, 8, and 12 form a group, as do tiles 5, 9, and 13, etc).

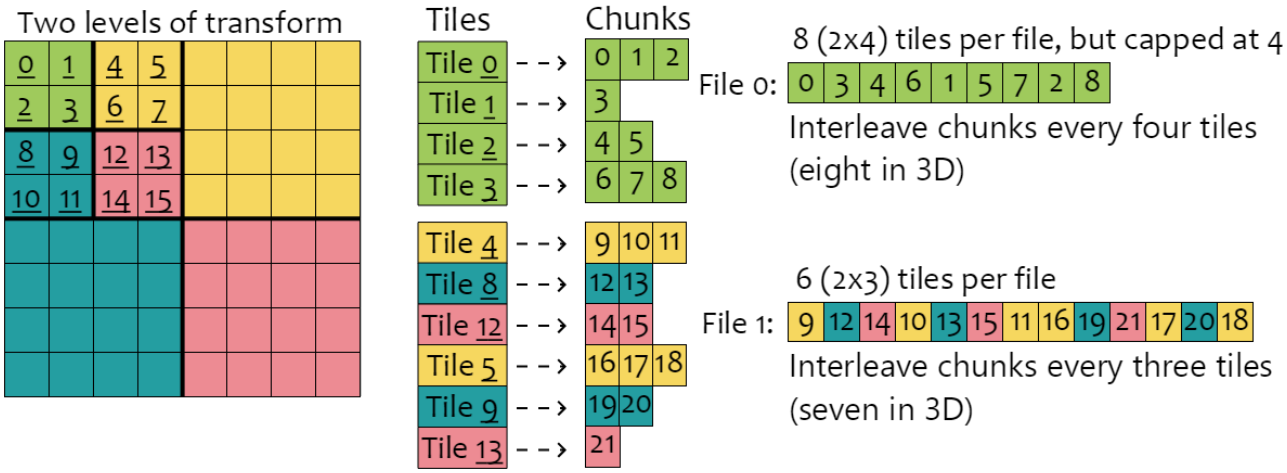


Figure 30. Grouping of chunks into files.

A file will consist of all the chunks belonging to K groups of 7 tiles. An exception is for the coarsest subband (tiles 0, 1, 2 and 3 in the figure), a tile group consists of 8 tiles (4 in 2D) instead of 7 tiles (3 in 2D). If K x 7 is more than N, the total number of tiles in the coarsest subband, the first file will only contain N tiles. Within a tile group, chunks are interleaved so that higher-order chunks (in precision) are stored before lower-order ones (see the figure above).

Beside the chunks, each file contains a header with some metadata about the chunks. Each header corresponds to a tile group, and is 20 bytes in size. The 20 bytes consists of several parts: two 16-bit floating-point numbers, storing the minimum and maximum values of all samples in the tile groups, a 37-bit pointer, pointing to the chunk-aligned address of the first chunk in the tile group, an 11-bit integer storing the maximum exponent among all wavelet coefficients in the tile group, and eight 10-bit values, each specifies the number of chunks for a tile in the tile group.

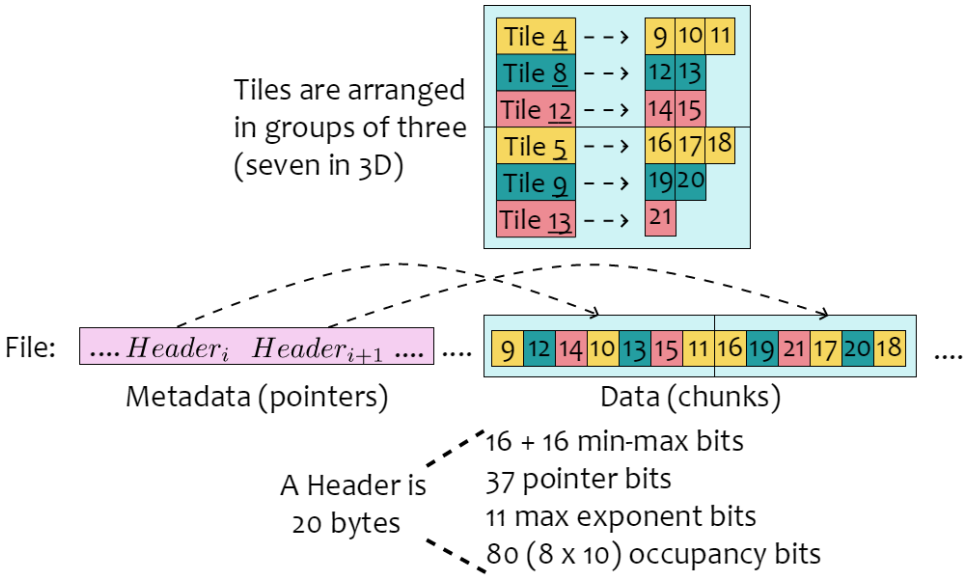


Figure 31. File headers.

We have implemented a prototype that converts raw binary volumes into our file format. Once fully functional, our proposed file format has the following capabilities:

- Lossy compression should be comparable to zfp: the overheads in the form of file headers should be minimal.
- The user can specify an error tolerance during compression and decompression. This capability is inherited from zfp.
- Reading of low-resolution data without touching the high-resolution bits is achieved by storing the coarser-subband tiles before finer-subband ones.
- Reading of low-precision data without touching the high-precision bits is achieved by reading data in chunks instead of tiles (which contain all precision bits).
- Incremental reading in any combination of precision and resolution is achieved by allowing random access to the chunks.
- Random access at block level, supporting region-of-interest reading is supported with pointers to the tile groups.
- Reading more important data blocks before less important ones is achieved through estimating the energy contribution of a tile group using the stored maximum exponent in the header.
- Skipping of irrelevant data blocks (e.g., during isocontour extraction) is achieved by using the stored minimum and maximum values in the tile group headers. Furthermore, since the tiles form a spatial hierarchy, this "empty space skipping" capability works in a hierarchial manner, allowing to scale to very large domains.

Binomial coding for compression of particle positions

We implemented for binomial coding and tested it on a few particle data sets. A kd-tree is built on the particles where the splitting plane is always the midpoint of a node’s extent. We encode the number of particles in the left children in compressed form, taking advantage of the fact that this

number follows the binomial distribution with regard to the number of particles in the parent node (if the particles are assumed to be uniformly distributed). The binomial distribution is approximated with a Gaussian, and we use an arithmetic coder for compression. With this method we could compress 14% better than the state-of-the-art methods which do not make use of the binomial distribution.

Lastly, we rendered particles at reduced levels of details (reduced precision in positions), and tried to match the rendering results with ones where the particles are rendered at full precision in their positions. We have found that in many cases, a saving of approximately 7x in data size is possible without compromising the rendering quality.

Table 1. Our implementation most of the time can achieve 12% to 14% saving compared to the state of the art. One exception is the cosmology data set, where the distribution of n is far from Binomial.

data sets	state-of-the-art (bytes)	ours (bytes)
alfredo	5,579	4,834
nanosphere	244,311	211,461
priya	1,730,378	1,538,745
vis contest	576,138	492,427
cosmo	32,277,741	37,784,143

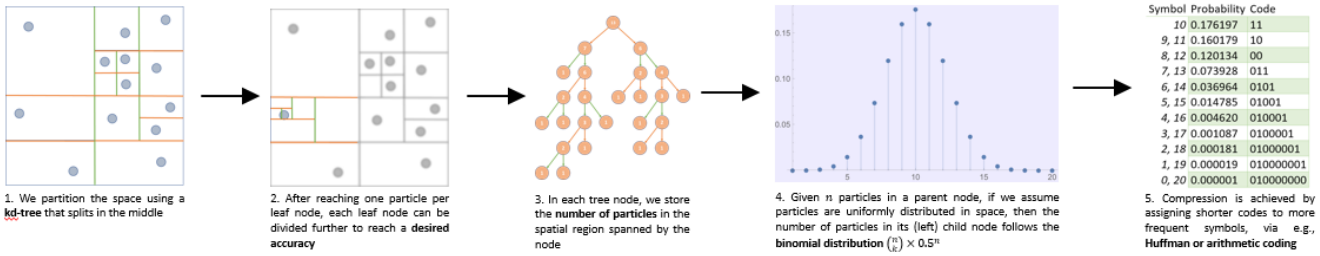


Figure 32. Particle compression pipeline.

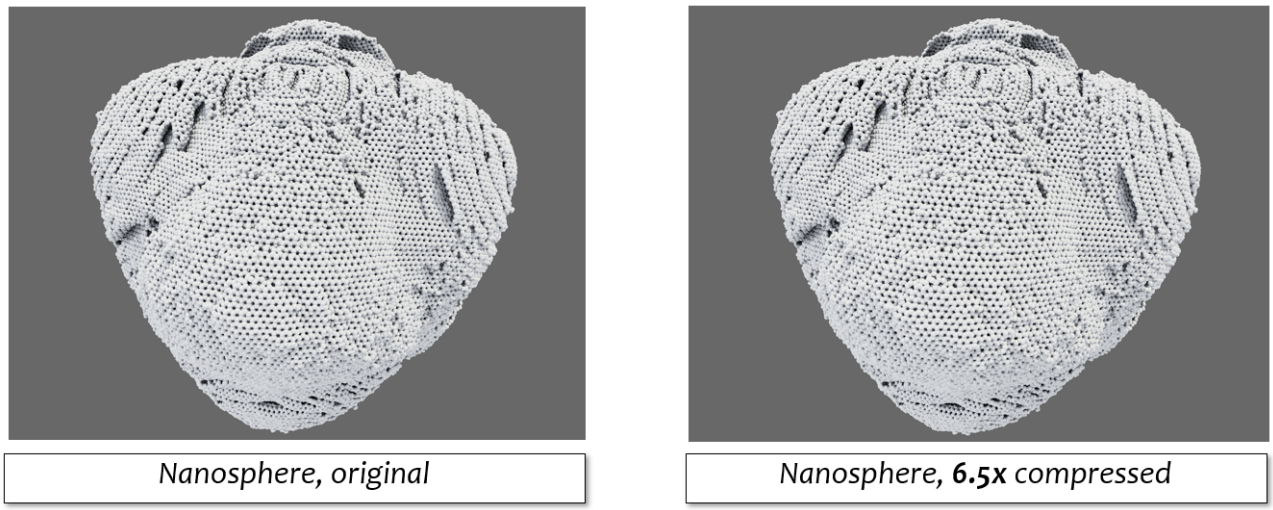


Figure 33. Rendering of compressed particles.

Binomial coding for Lorenzo predicted residuals

We have been investigating the use of binomial coding to compress residuals of the Lorenzo

predictor which is used by the FPZIP compression library. The idea was that if a binary tree based on the sum operator (instead of max) is build on the residuals, according to the Central Limit Theorem, the values on upper levels will tend to a Gaussian, which can be approximated well with a binomial distribution. We learned soon after that this idea overlooked the fact that the distribution of a child needs to be conditioned on the value of its parent. In the case that the child follows the Exponential distribution, this conditional distribution turns out to be uniform, which cannot be easily exploited in coding. We then shifted to investigating using the max operator and encoding the (left-right) differences, conditioned on the max.

Assuming the unsigned residuals follow the exponential distribution with parameter λ (which can be approximated using the Maximum Likelihood estimator), and X and Y are i.i.d random variables ~Exponential(λ), then the pdf of (n=X-Y|□m=max□□(X,Y)) is (λ exp□(λ|n|))/(2(exp□(λm)-1)). More general results have also been derived, where X and Y are the max and sum of more than one random variables.

However, the distribution of Lorenzo residuals can be far from an exponential distribution, leading to the analytically-computed distribution being very far from the actual distribution for the (left-right) differences, conditioned on the max. We are experimenting with other, more general distributions such as Gamma and Lomax. Another issue with this approach is that in order for the statistics to work, residuals must be independent, which is not the case in practice. We have observed that transforming the residuals using linear wavelets help reduce the spatial correlation significantly.

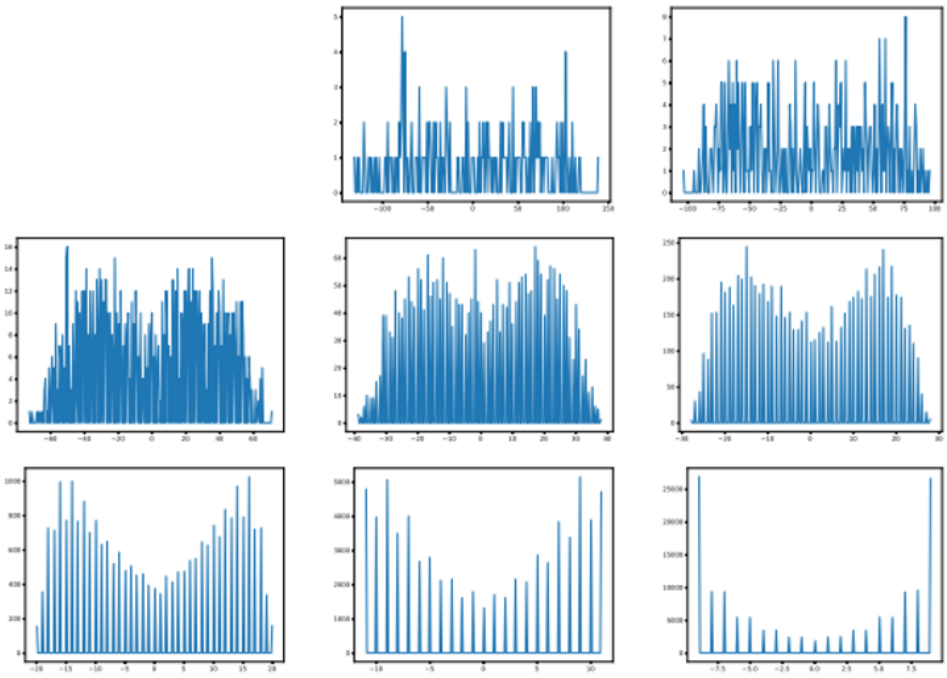


Figure 34. Distribution of (left child - right child), conditioned on the max, on each level (with shuffling), using 10th most frequent parent (Diffusivity).

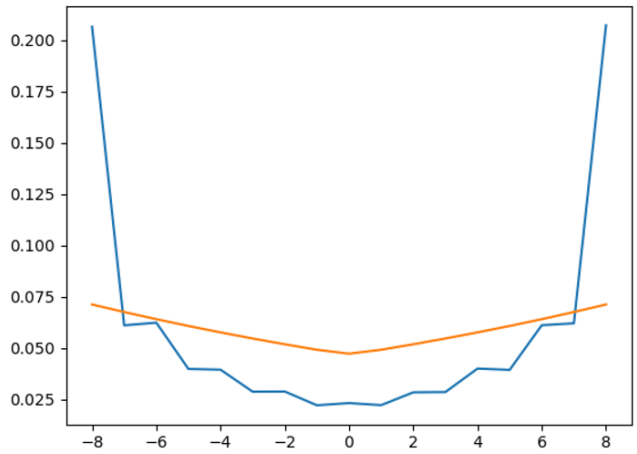


Figure 35. Distributions of (left-right), conditioned on the max (=8), on the leaf level. Orange: analytical distribution, Blue: actual distribution.

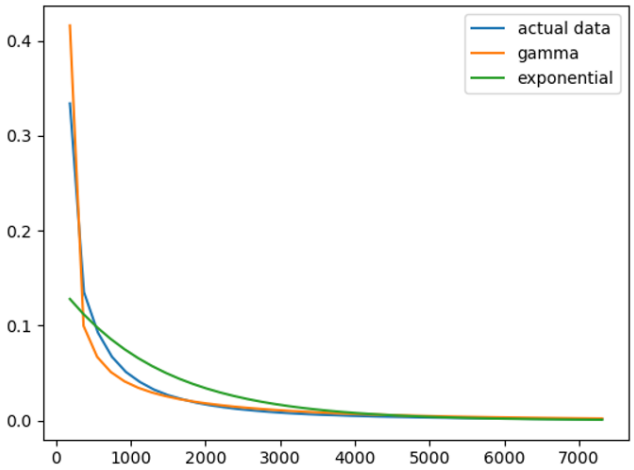


Figure 36. Approximating distribution of Lorenzo residuals using Gamma and Exponential distrbutions.

References

[1] S. Kumar, D. Hoang, S. Petruzza, V. Pascucci, J. Edwards. Reducing network congestion and synchronization overhead during aggregation of hierarchical data. 24th IEEE International Conference on High Performance Computing, Data, and Analytics (HiPC 2018).

[2] (Poster) S. Kumar, S. Petruzza, D. Hoang, V. Pascucci. Accelerating In-situ Feature Extraction of Large-Scale Combustion simulation with Subsampling. The 26th International Symposium on High Performance Parallel and Distributed Computing (HPDC 2017).

[3] D. Hoang, P. Klacansky, H. Bhatia, P.T. Bremer, P. Lindstrom, V. Pascucci. A Study of the Trade-off Between Reducing Precision and Reducing Resolution for Data Analysis and Visualization. IEEE Transaction on Visualization and Computer Graphics, January 2019.