Compressed Floating-Point Arrays for High-Performance Computing

SIAM Conference on Imaging Science

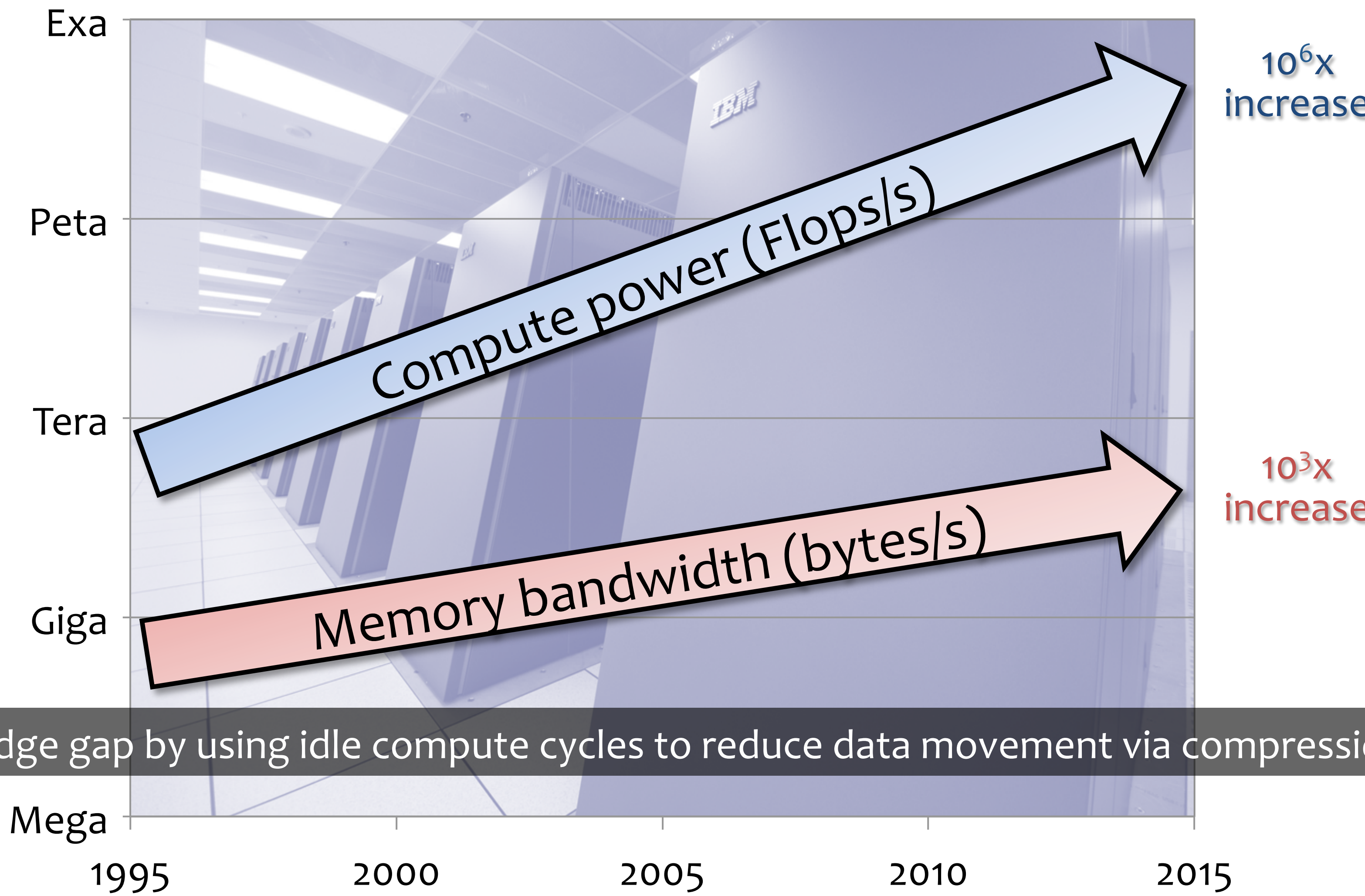
Peter Lindstrom

May 24, 2016

LLNL-PRES-692806
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

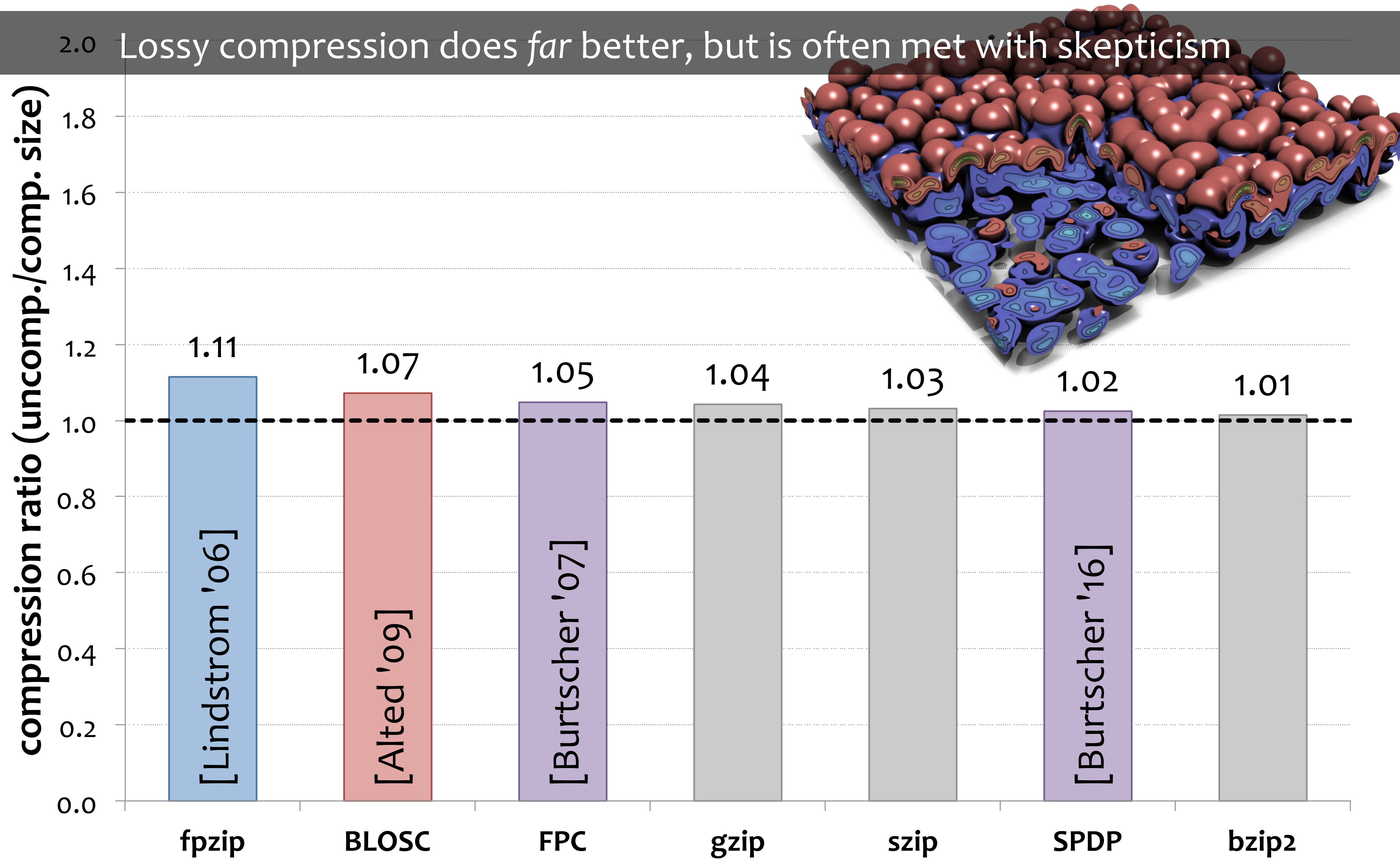
Trend points to widening gap between compute power and memory bandwidth



Idea: Bridge gap by using idle compute cycles to reduce data movement via compression

Floating-point data is difficult to compress—lossless compression is often not sufficient

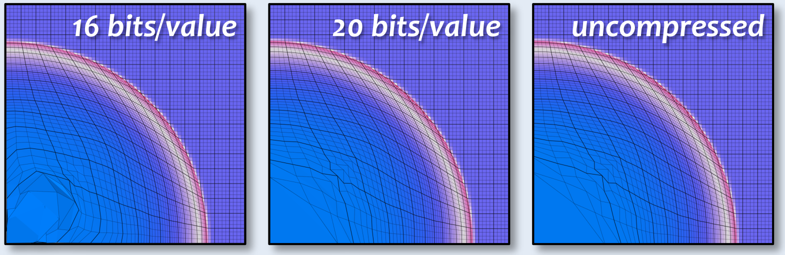
Lossy compression does *far* better, but is often met with skepticism



We have shown that 4x lossy compression of simulation state variables can be tolerated

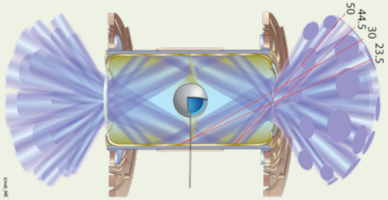
LULESH: Lagrangian shock hydrodynamics

- QoI: *radial shock position*
- 25 state variables compressed over 2,100 time steps
- At **4x compression**, relative **error < 0.06%**



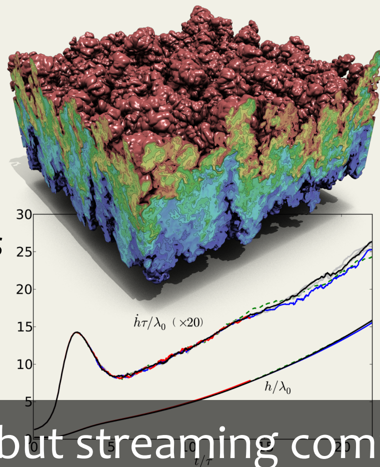
pf3D: Laser-plasma multi-physics

- QoI: *backscattered laser energy*
- At **4x compression**, relative **error < 0.1%**



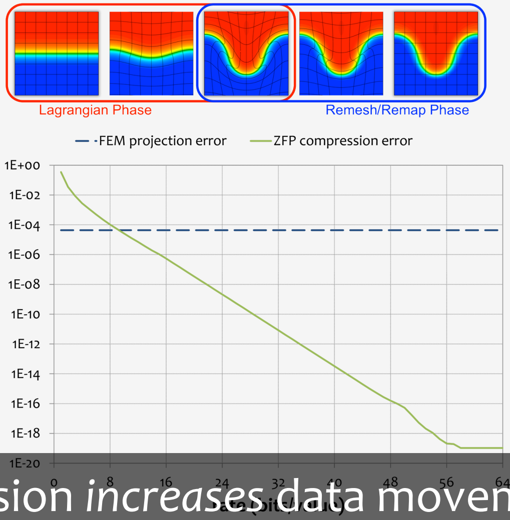
Miranda: High-order Eulerian hydrodynamics

- QoI: *Rayleigh-Taylor mixing layer thickness*
- 10,000 time steps
- At **4x compression**, relative **error < 0.2%**



MFEM: Cubic finite elements

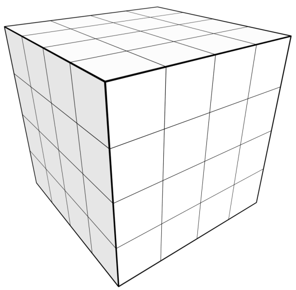
- QoI: *function approximation*
- **6x compression** with ZFP **error < 0.7%** relative to FEM error



Lossy compression of state is viable, but streaming compression *increases* data movement

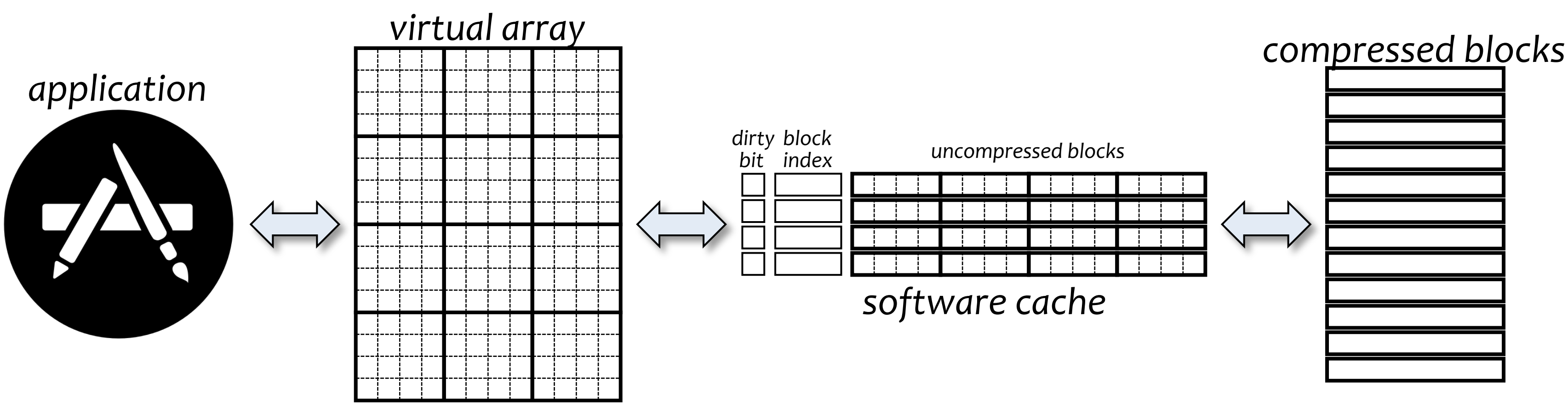
We have developed ZFP: the first inline compressor for floating-point arrays

- Inspired by ideas from h/w texture compression
  - 1D, 2D, or 3D array divided into fixed-size 4×4×4 blocks
  - Each block is independently (de)compressed
    - e.g. to a user-specified number of bits or quality
  - Fixed-size blocks ⇒ **random read/write access**
  - (De)compression is done inline, on demand
  - Write-back cache of uncompressed blocks limits data loss



- Compressed arrays via C++ operator overloading
  - Can be dropped into existing code by changing type declarations
  - double a[n] ⇔ std::vector<double> a(n) ⇔ zfp::array<double> a(n, **precision**)

Our compressed array limits data loss via a small write-back cache



Compress only "dirty" blocks when evicted from the cache

Our ZFP compressor is comprised of three distinct components

Raw floating-point array → Block floating-point transform → Orthogonal block transform → Embedded coding → Compressed bit stream

Block floating-point transform

- *Align values in a 4^d block to a common largest exponent*
- *Transmit exponent verbatim*

Orthogonal block transform

- *Lifted, separable transform using integer adds and shifts*
- *Similar to but faster and more effective than JPEG DCT*

Embedded coding

- *Encode one bit plane at a time from MSB using group testing*
- *Each bit increases quality—can truncate stream anywhere*

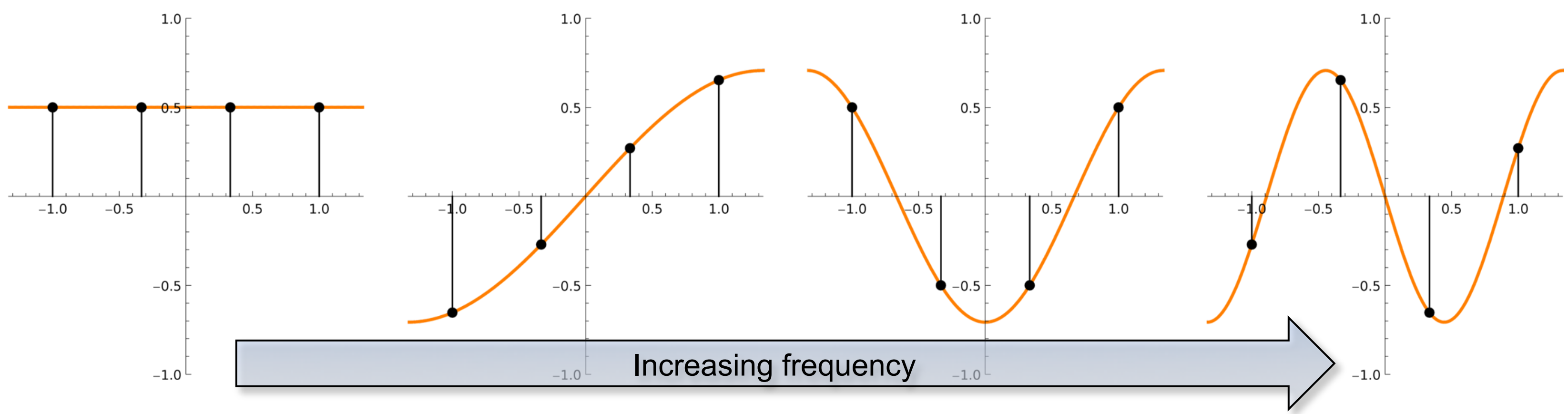
We decorrelate values in a block using a separable orthogonal transform

$$\underbrace{\begin{pmatrix} \hat{f}_1 \\ \hat{f}_2 \\ \hat{f}_3 \\ \hat{f}_4 \end{pmatrix}}_{\textit{coefficients}} = \underbrace{\frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \\ c & s & -s & -c \\ 1 & -1 & -1 & 1 \\ s & -c & c & -s \end{pmatrix}}_{\textit{orthogonal transform}} \underbrace{\begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{pmatrix}}_{\textit{block}}$$

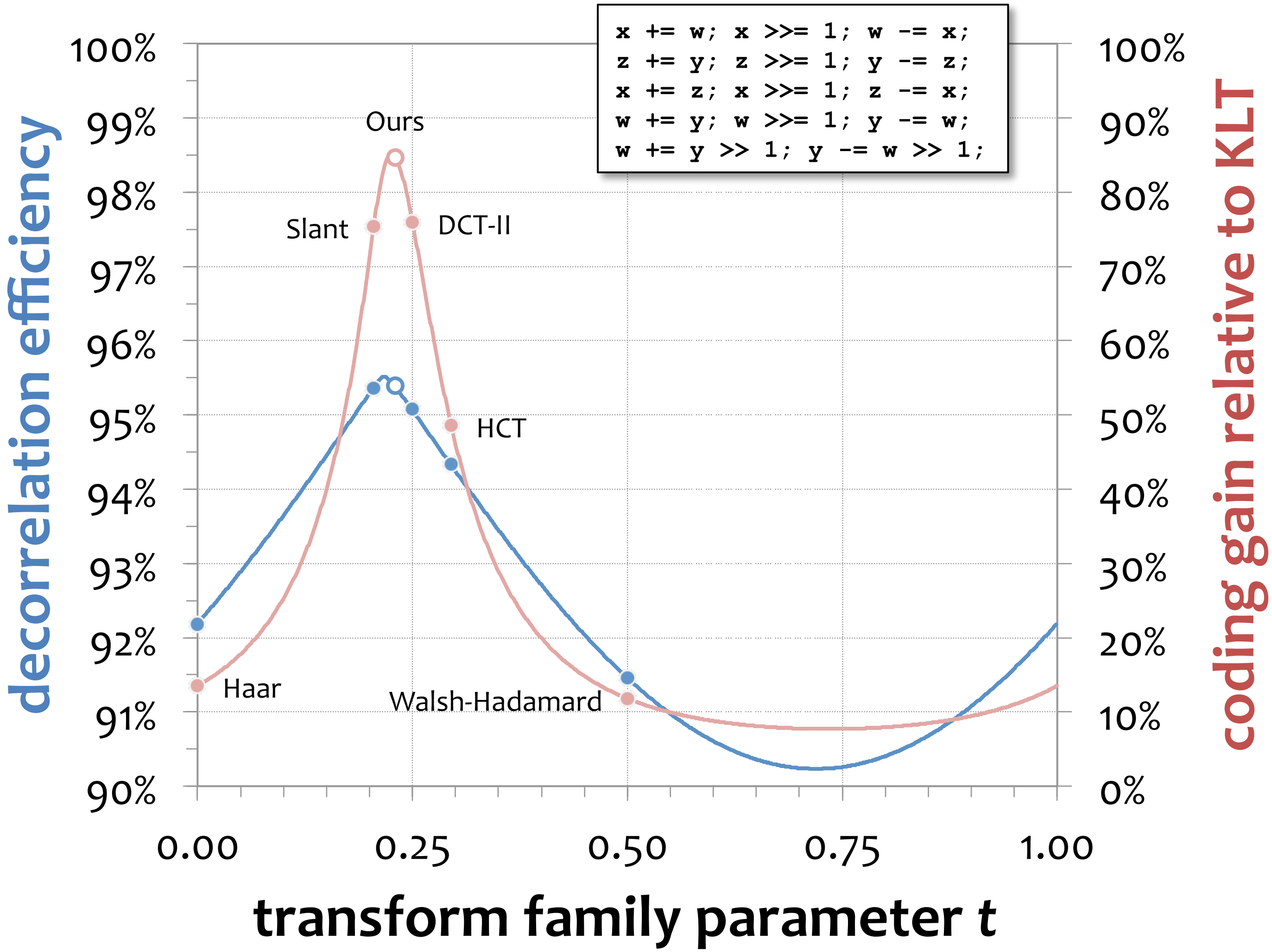
Free parameter t

$$s = \sqrt{2}\sin\tfrac{\pi}{2}t \qquad c = \sqrt{2}\cos\tfrac{\pi}{2}t$$

Basis functions for $t = 1/4$

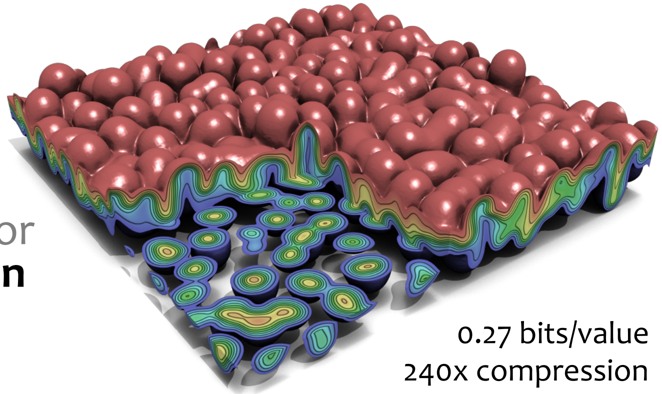


Our integer transform is efficient, effective, and well-suited for h/w implementation



ZFP is flexible and highly performant

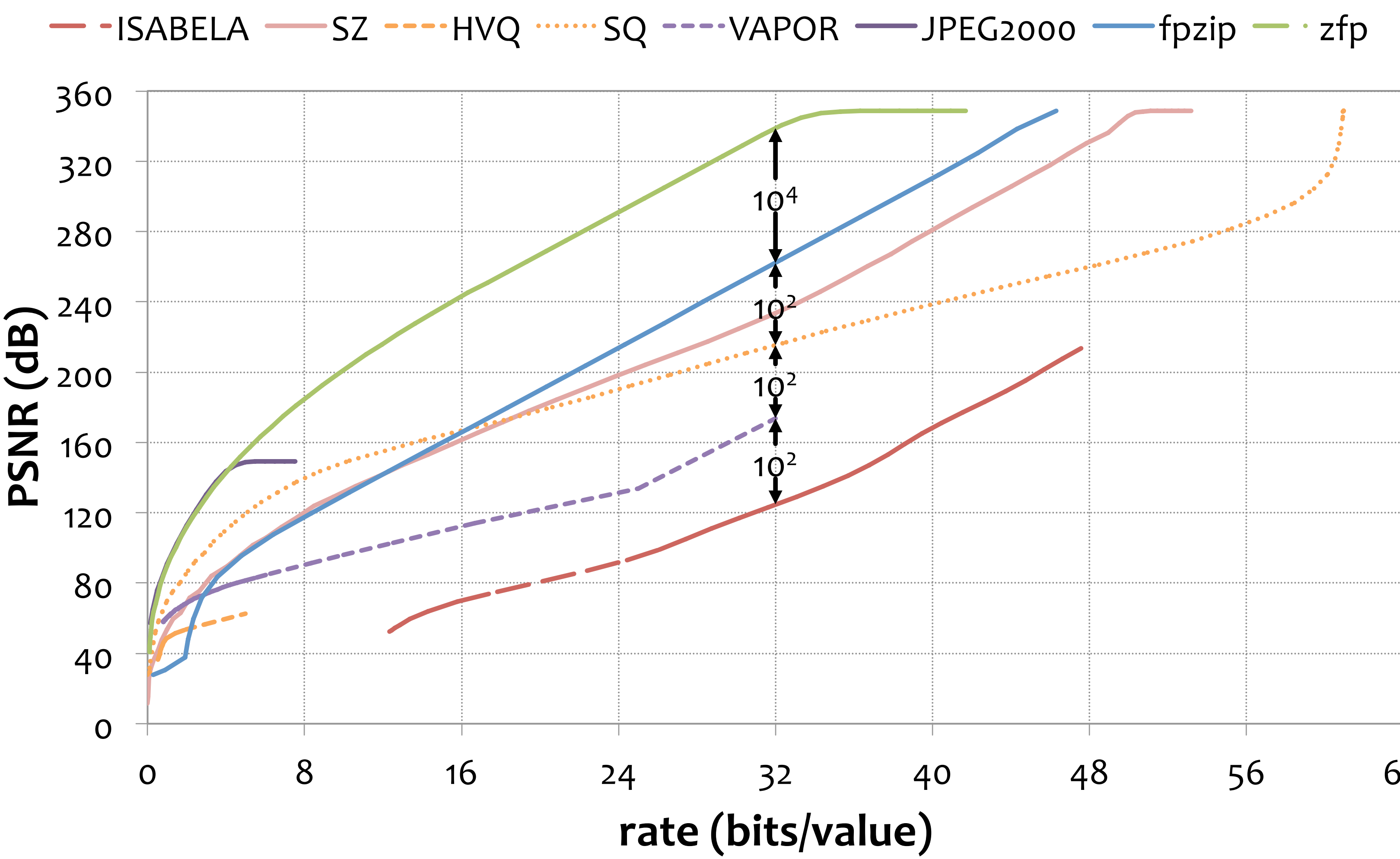
- Three compression modes in single CODEC
  - **Fixed rate:** Only inline (de)compressor with **read and write random access**
  - **Fixed precision:** Fixed no. mantissa bits ensures **relative error bound**
  - **Fixed accuracy:** User tolerance ensures **absolute error bound**
  - Also supports **progressive decompression**
- Very high quality across many science domains
  - **>100x more accurate** than closest competitor
- High, symmetric encoding & decoding speed
  - Up to **2 GB/s/core: 2-6x faster** than closest competitor
  - Simple algorithm is amenable to **h/w implementation**
    - Integer bit shifts, additions, bit-wise logical operations



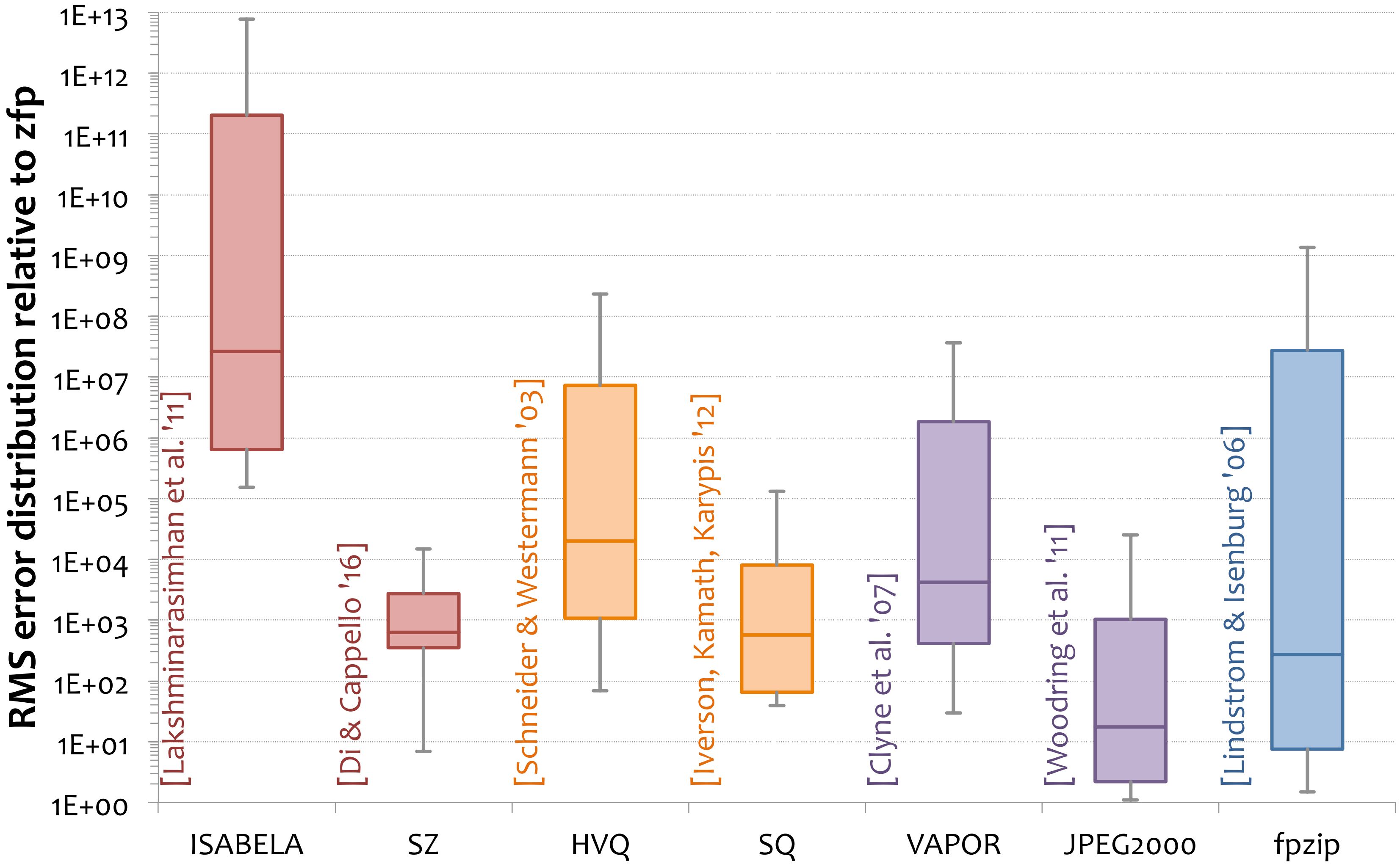
0.27 bits/value
240x compression

- Small, independent blocks of compressed data
  - Fine granularity provides **adaptive quality**, **culling**, **queries**, **domain decomp**, …
  - Supports streaming with a **tiny memory footprint** as small as a single block
  - **OpenMP**, **GPU parallelization** over (and within) blocks
  - **Resilience** to data corruption—a flipped bit affects only a single block
- C++ compressed arrays replace STL vectors with minimal code changes
  - LULESH miniapp integration took less than **20 minutes of coding effort**

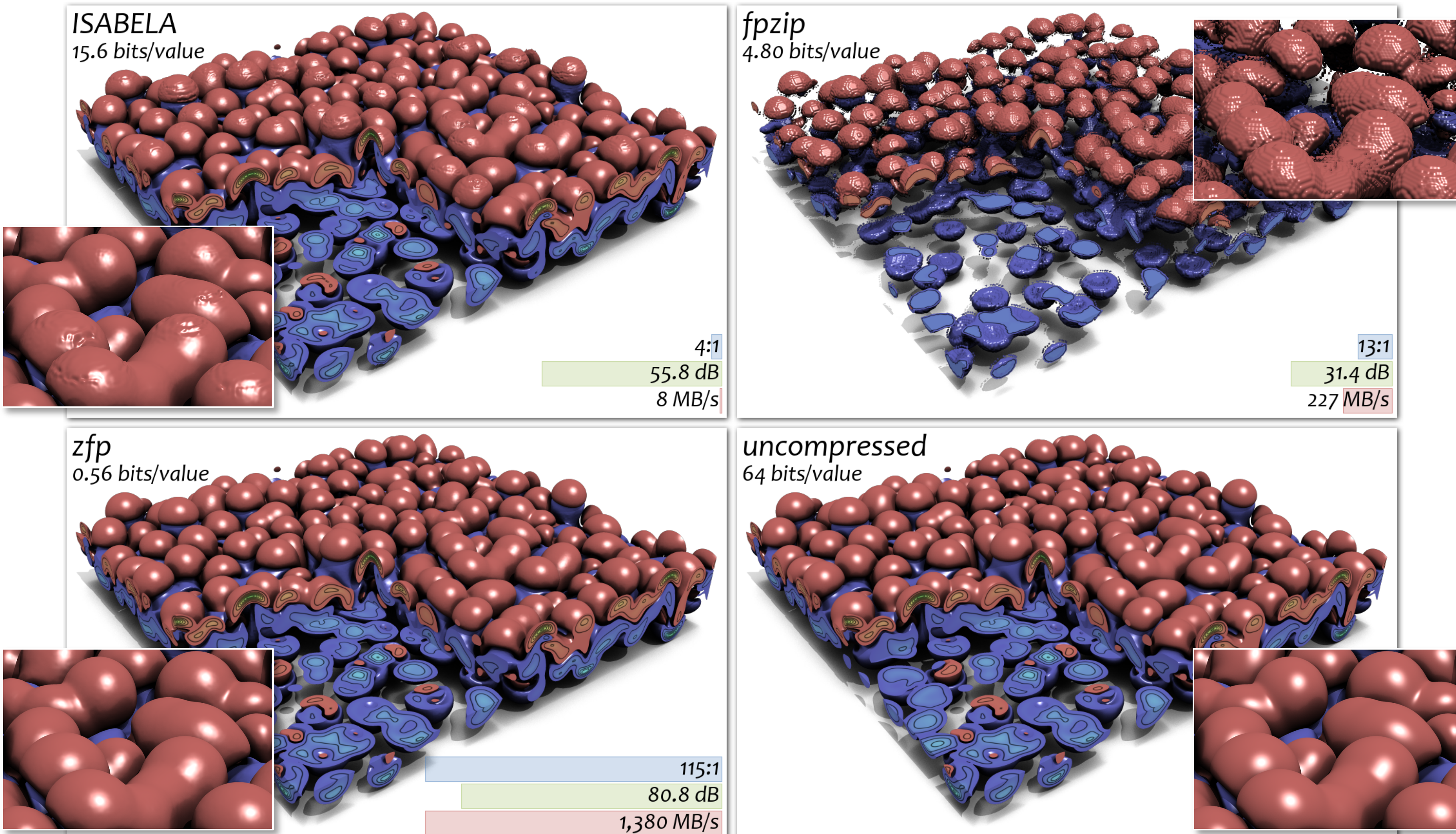
ZFP fixed-accuracy mode dramatically improves quality over previous methods



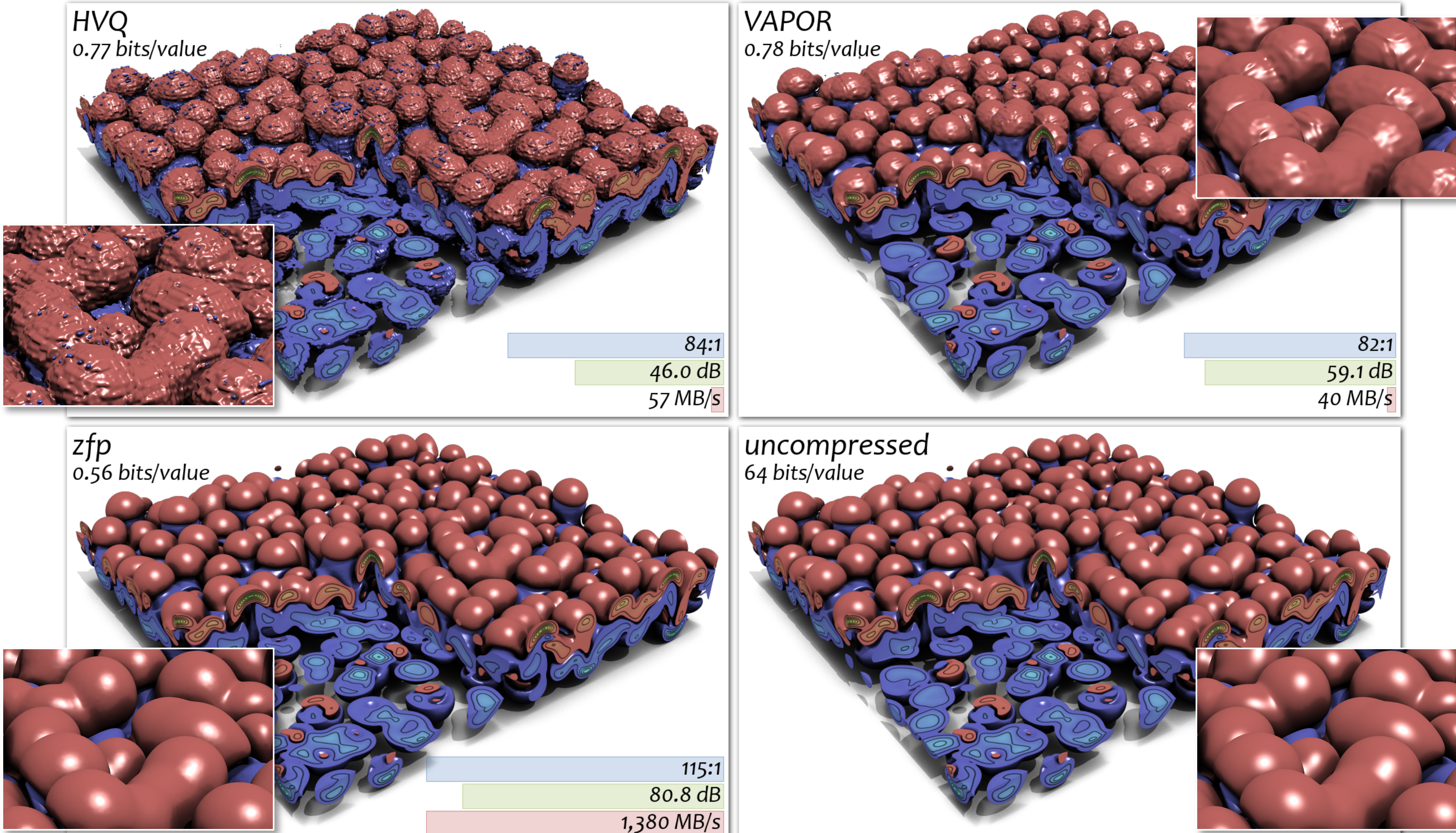
ZFP consistently achieves the highest accuracy among compressors and data sets



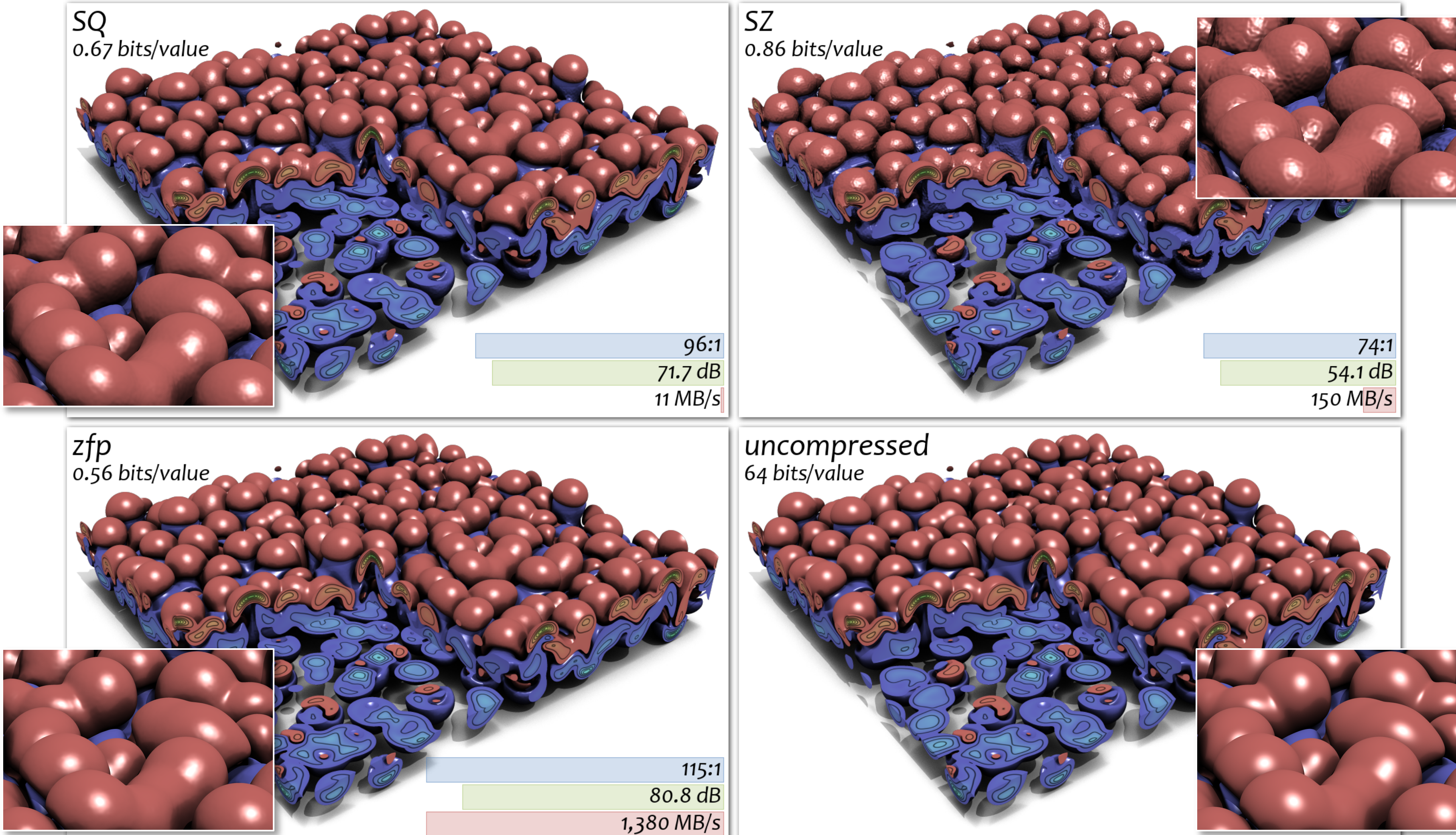
ZFP provides >100x compression with imperceptible loss in quality



ZFP provides >100x compression with imperceptible loss in quality

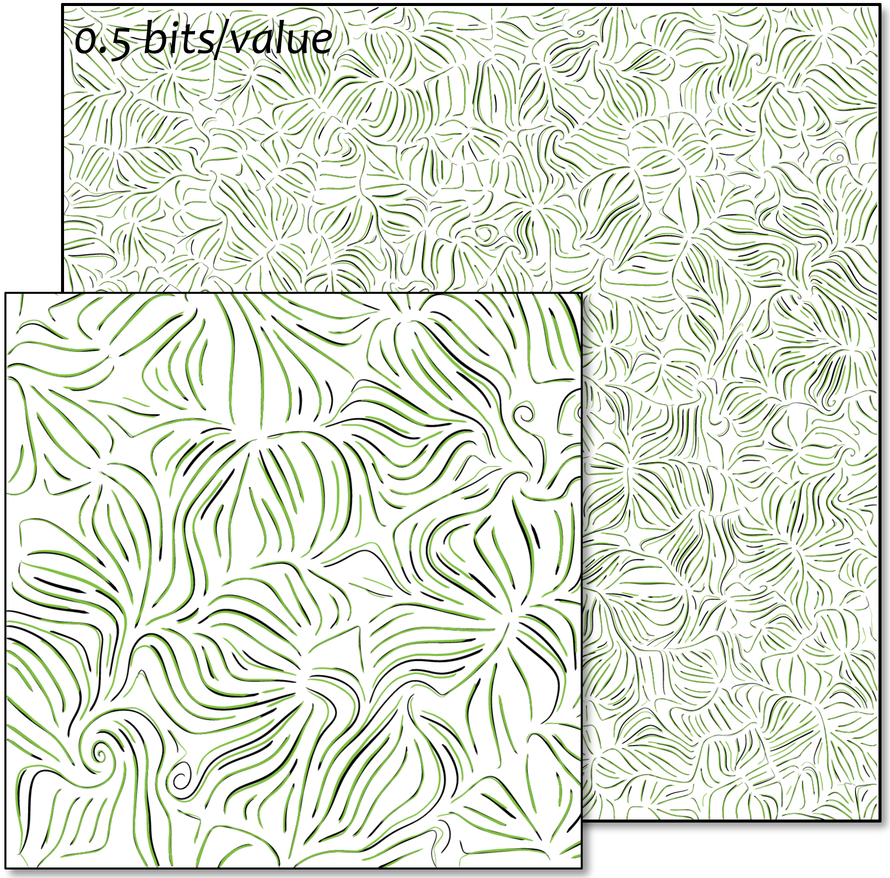


zfp provides >100x compression with imperceptible loss in quality

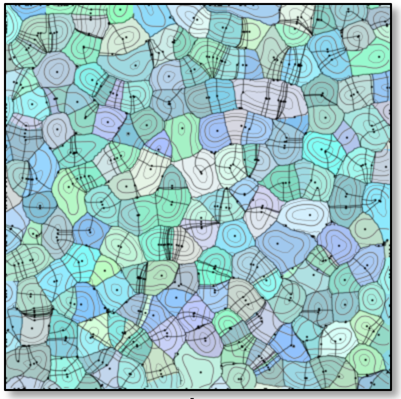


Got artifacts?

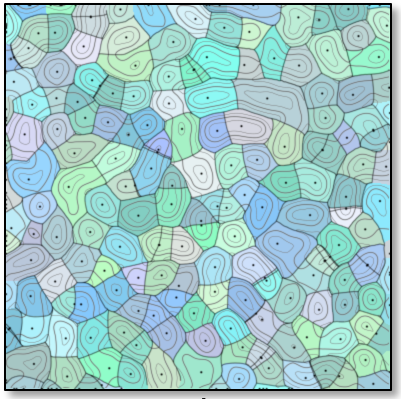
Velocity field Runge-Kutta integration shows good agreement with uncompressed field



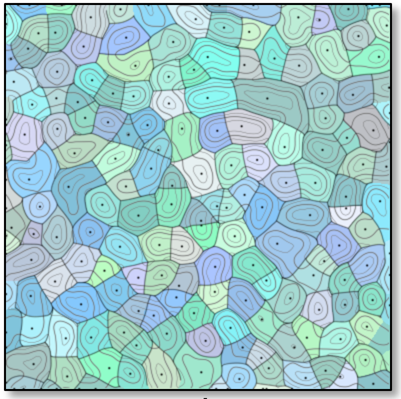
Morse segmentation at 16x compression shows lack of blocking artifacts



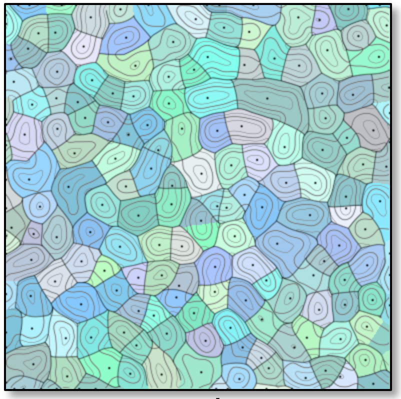
1 bit/value



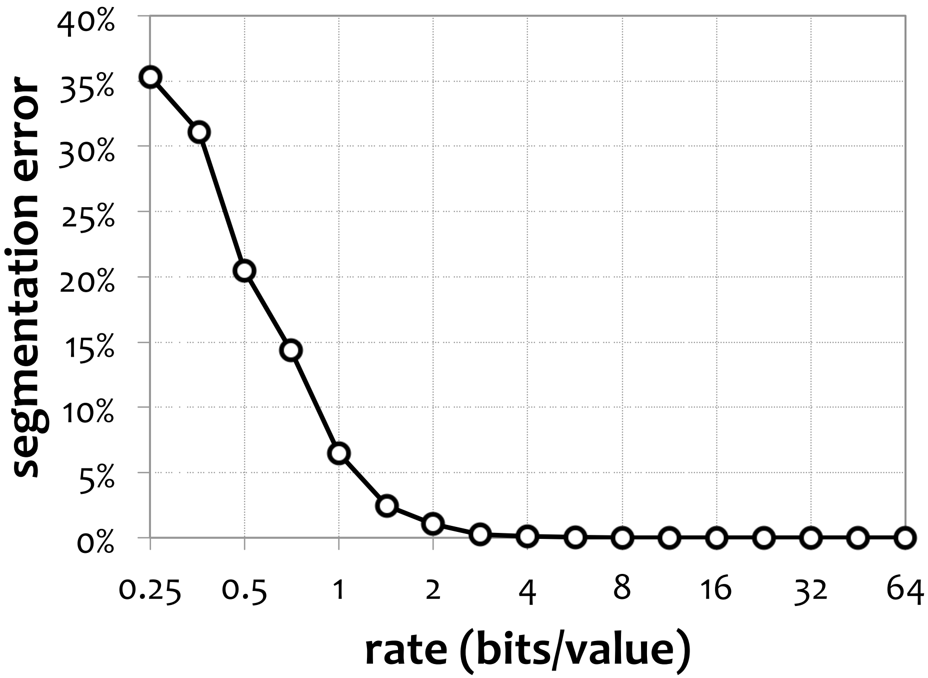
2 bits/value



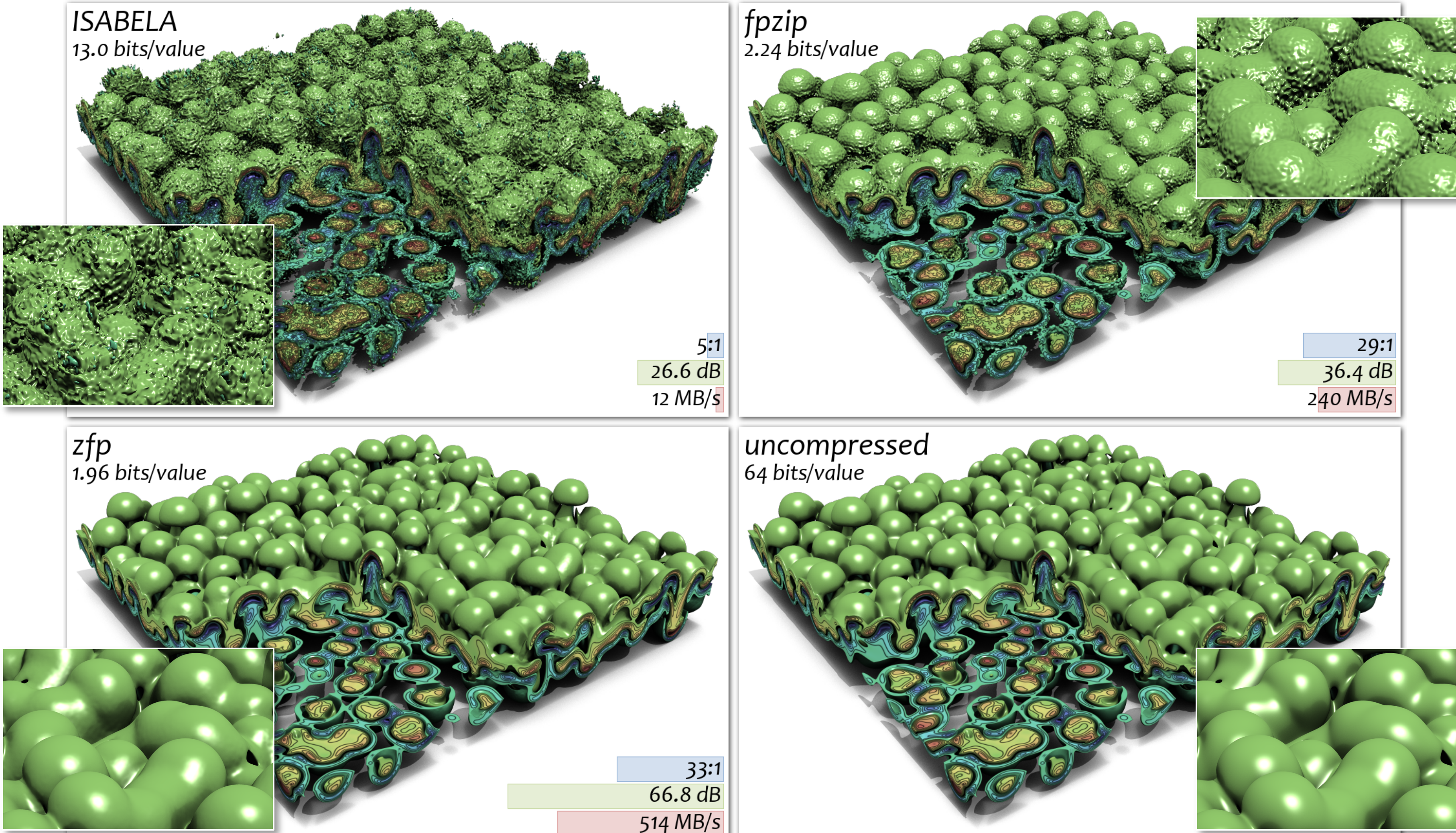
4 bits/value



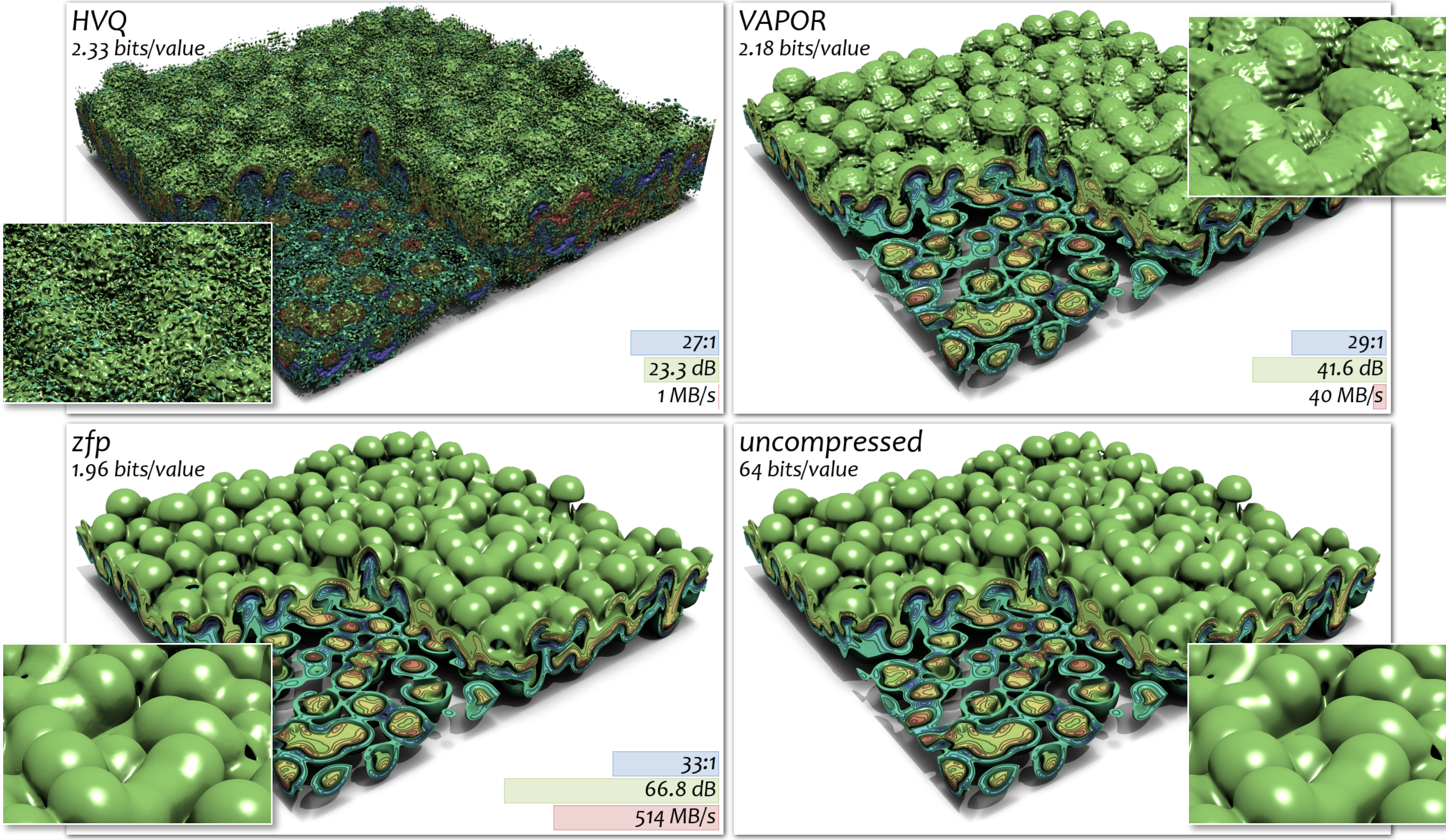
64 bits/value



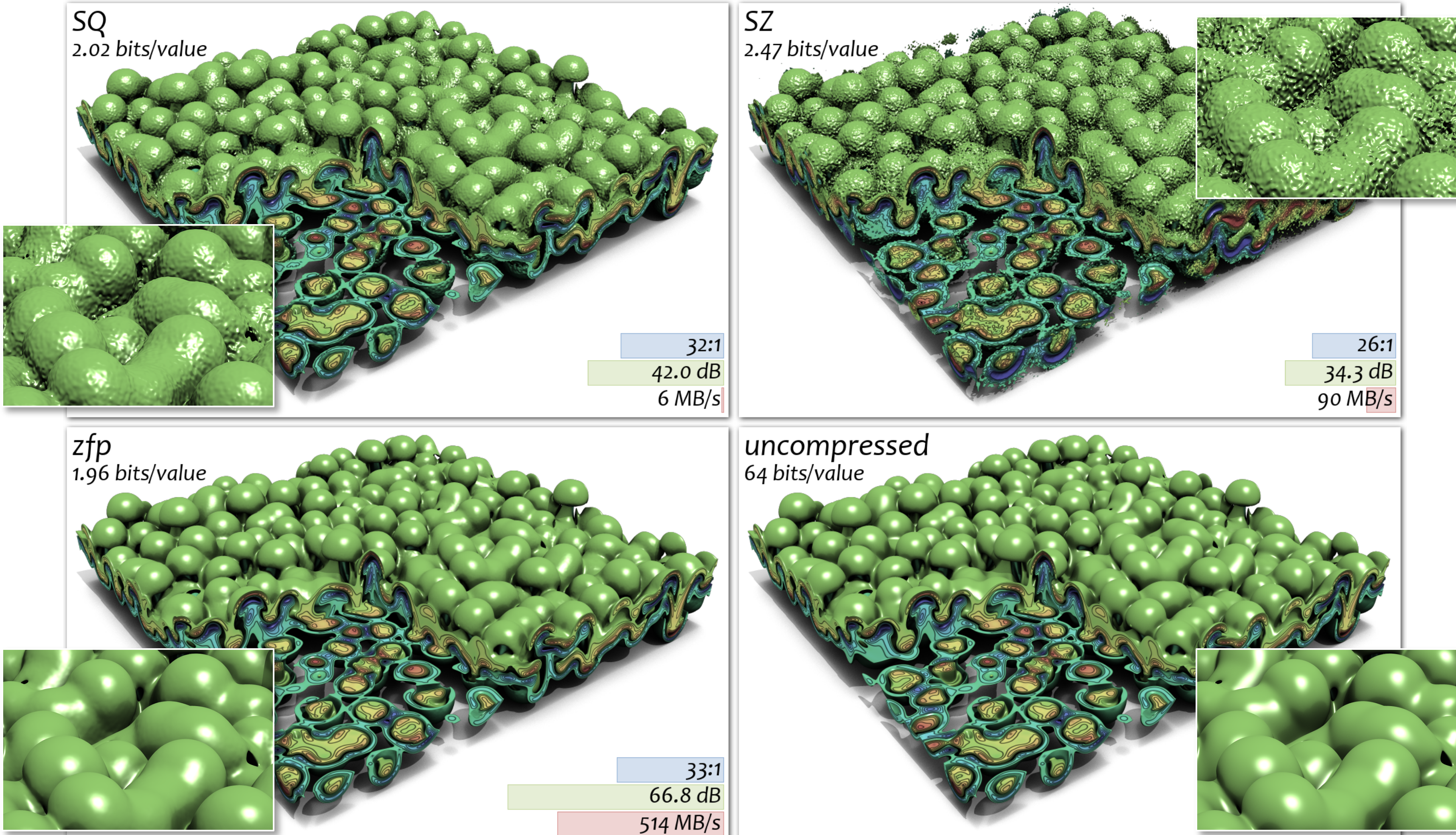
ZFP shows no artifacts in derivative computations (velocity divergence)



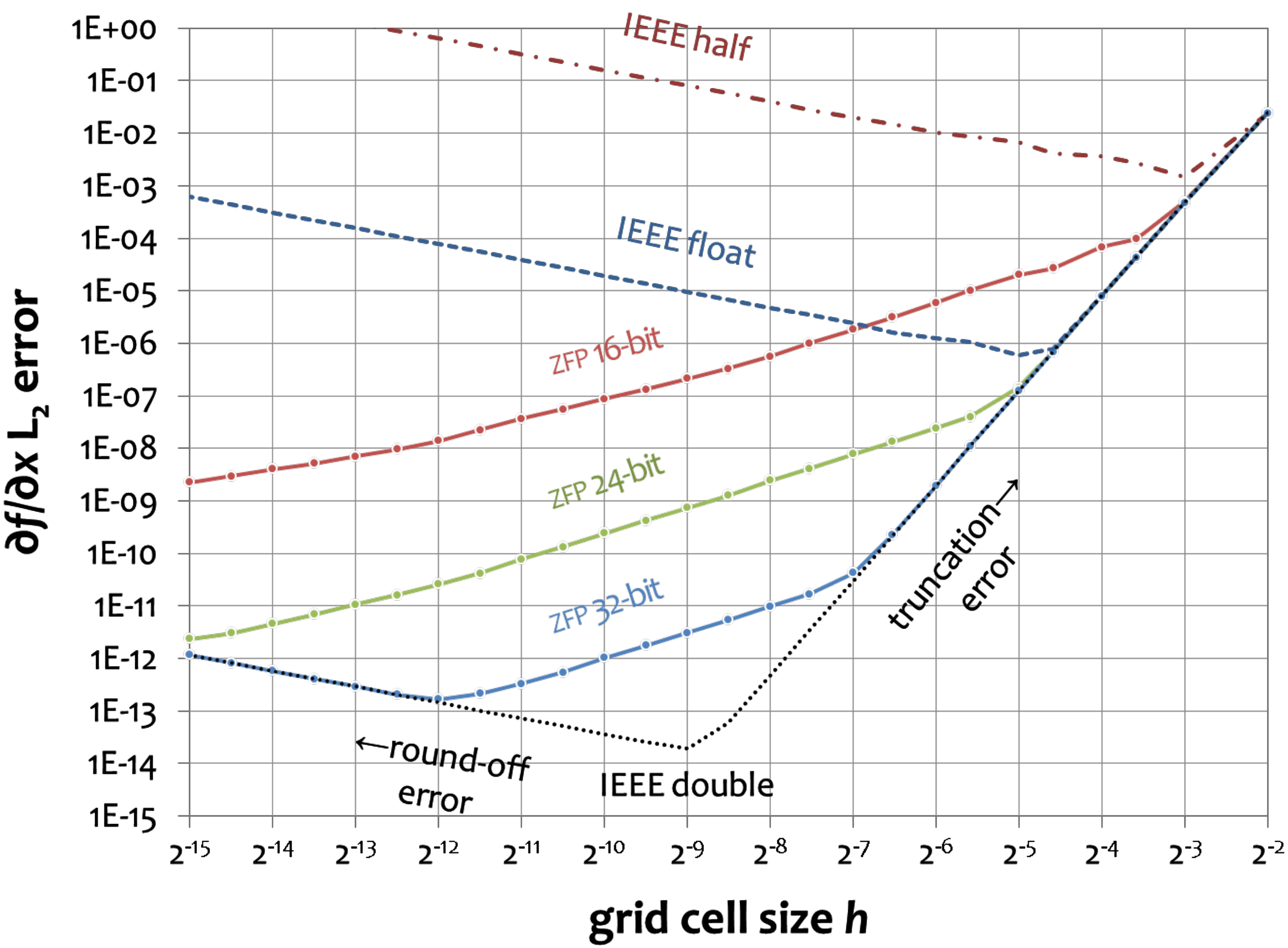
ZFP shows no artifacts in derivative computations (velocity divergence)



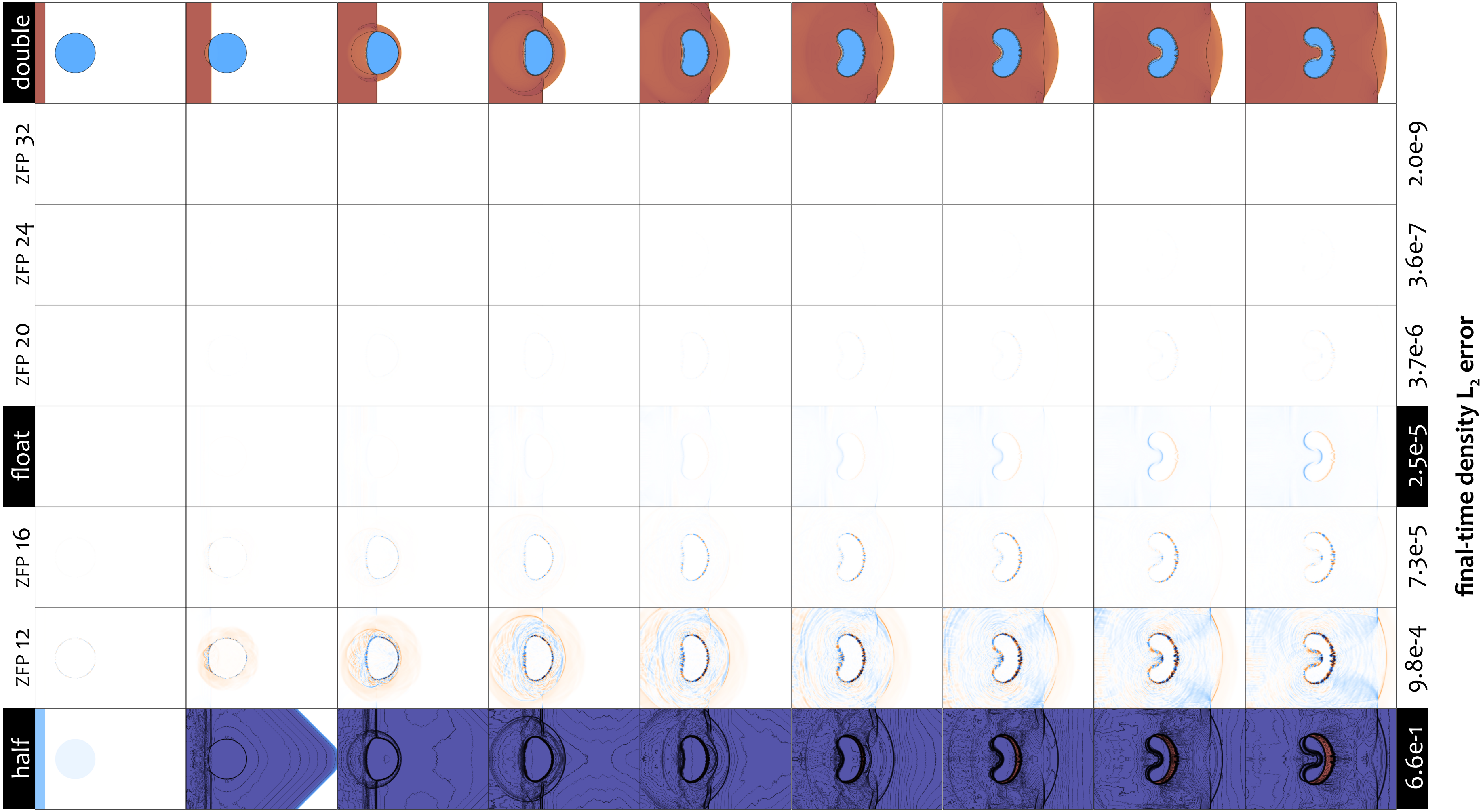
ZFP shows no artifacts in derivative computations (velocity divergence)



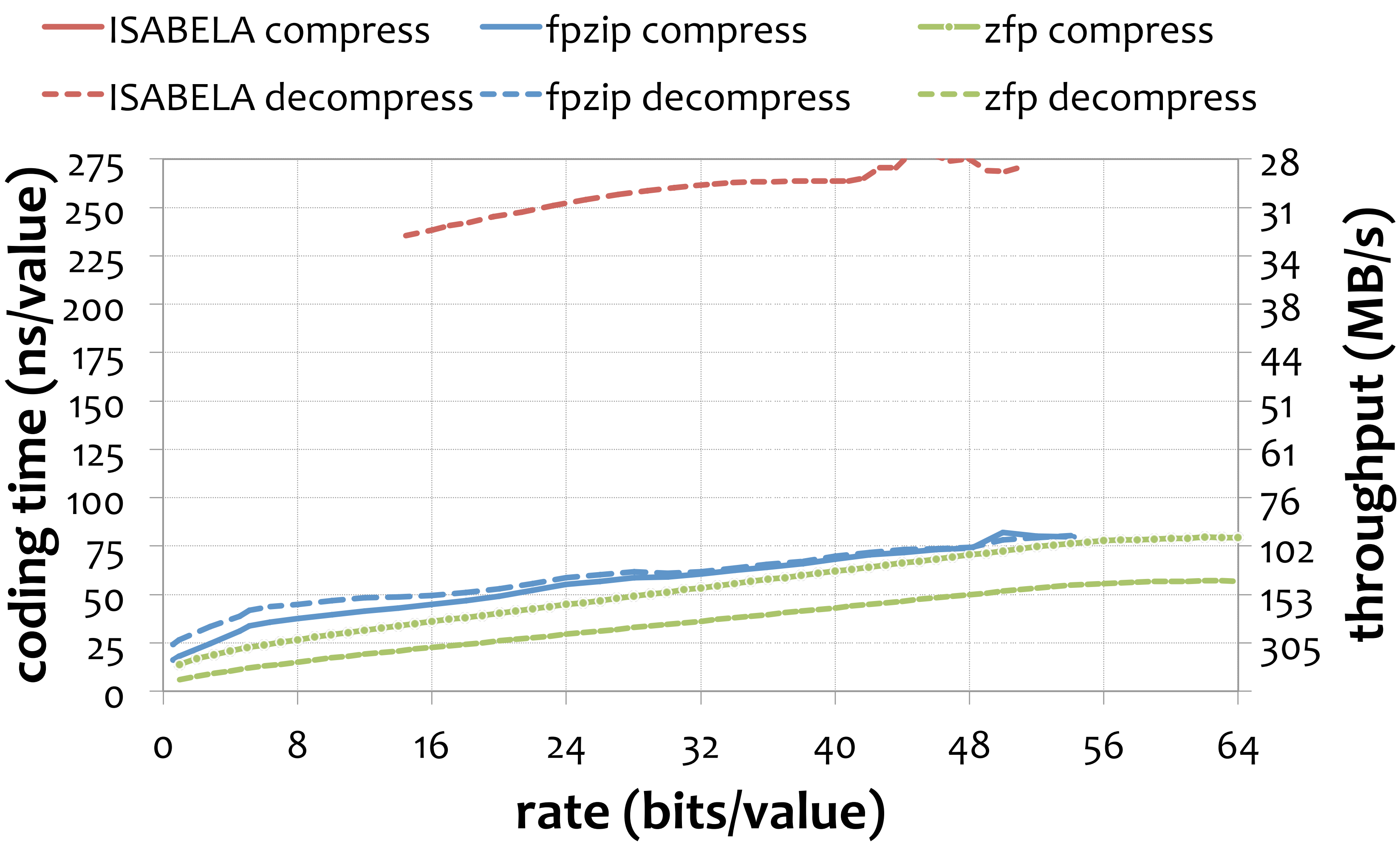
ZFP *improves* accuracy in finite difference computations using less precision than IEEE



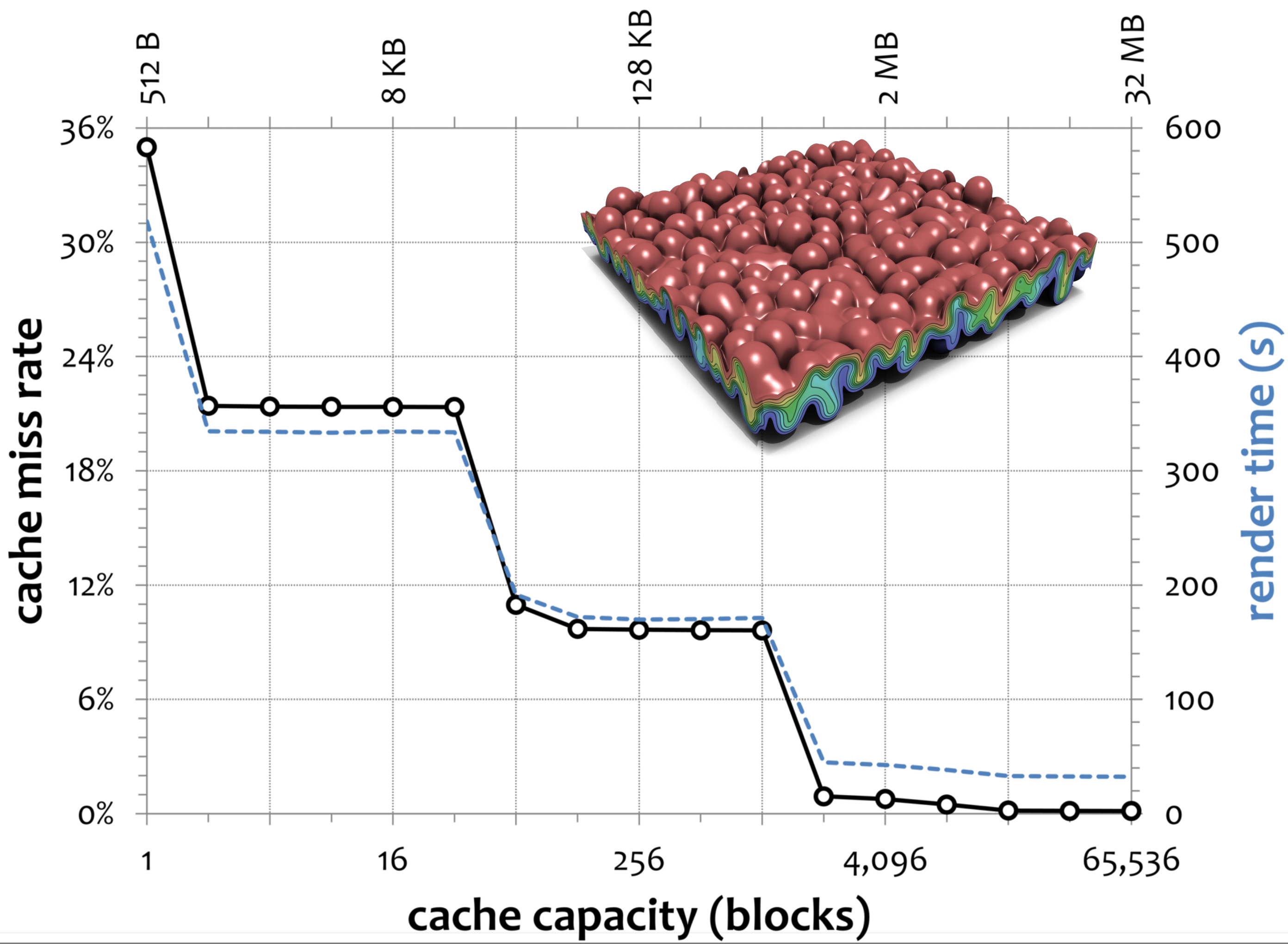
ZFP achieves 5.3x inline compression with <0.1% error in shock-bubble interaction



ZFP achieves up to 2 GB/s/core throughput and delivers predictable performance

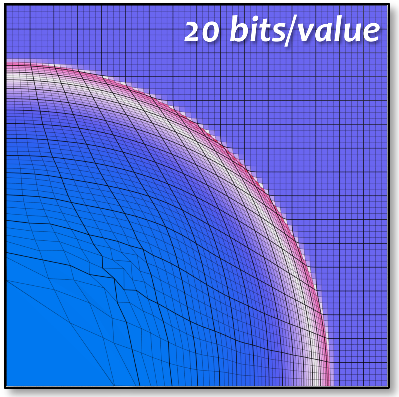


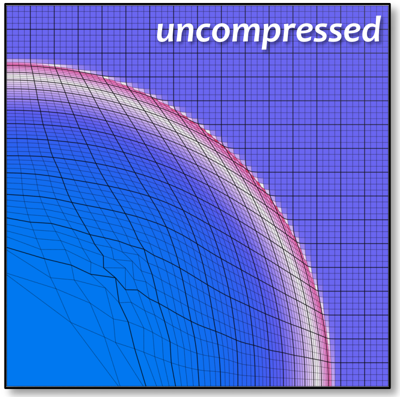
Our s/w implementation is 17% away from performance break-even point on single core

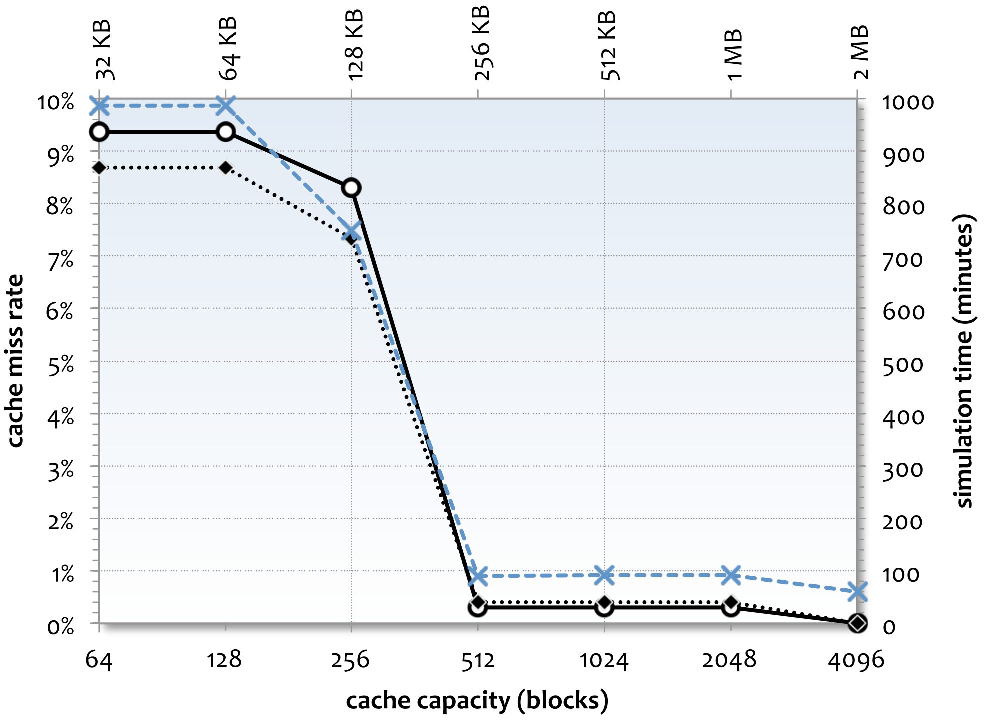


Substantial data movement reduction was achieved in LULESH mini app

quantity	*with compression*	*without compression*	*ratio*
cache size; total array size	256 KB	2 MB	8x
decompressed; read accessed	51.4 GB	733 GB	14x
compressed; write accessed	15.9 GB	242 GB	15x
running time	90 min	15 min	6x
compression calls/ block/time step	1.17	0	
read miss rate	0.3%	N/A	



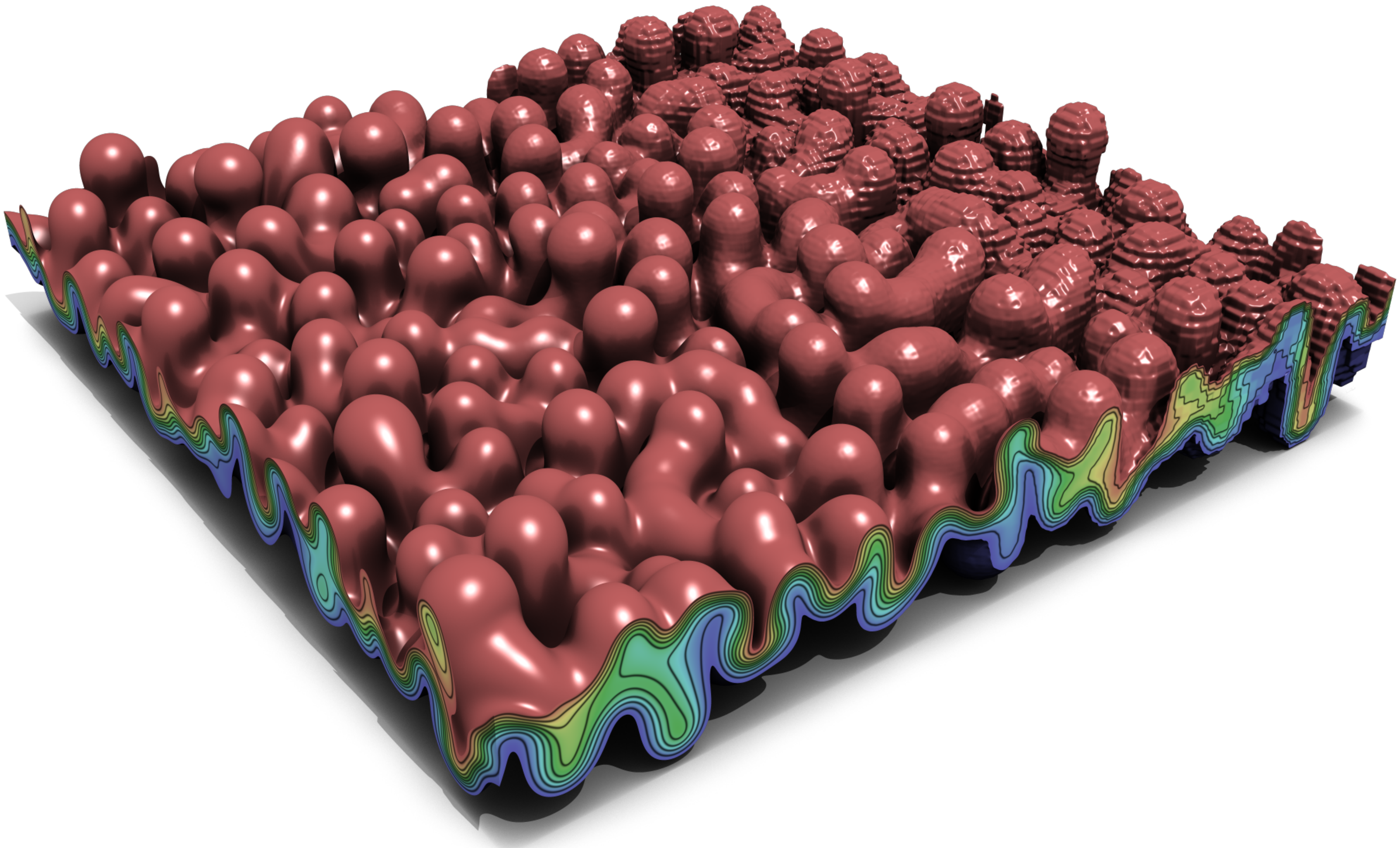




Summary: ZFP has the potential to dramatically reduce data movement in HPC applications

- ZFP is a better floating-point representation than IEEE
  - Half the storage for the same accuracy
  - Four orders of magnitude higher accuracy for the same storage
- ZFP is so far primarily used for read-only access
  - I/O compression: Turbulence, combustion, fusion, radiation transport, seismology, climate, weather, VFX, high-dynamic-range images, …
  - In-memory table compression: Equation of state, opacity tables
- Inline compression reduces memory footprint, bandwidth
  - Alleviates many-core cache and memory bandwidth contention
  - ZFP's symmetric performance ensures fast compress and decompress
- Hardware compression is ubiquitous on GPUs, mobile devices
  - How come HPC community has not caught on?

ZFP is released as open source



http://computation.llnl.gov/projects/floating-point-compression

Lawrence Livermore National Laboratory