

Compressed Number Representations for High-Performance Computing

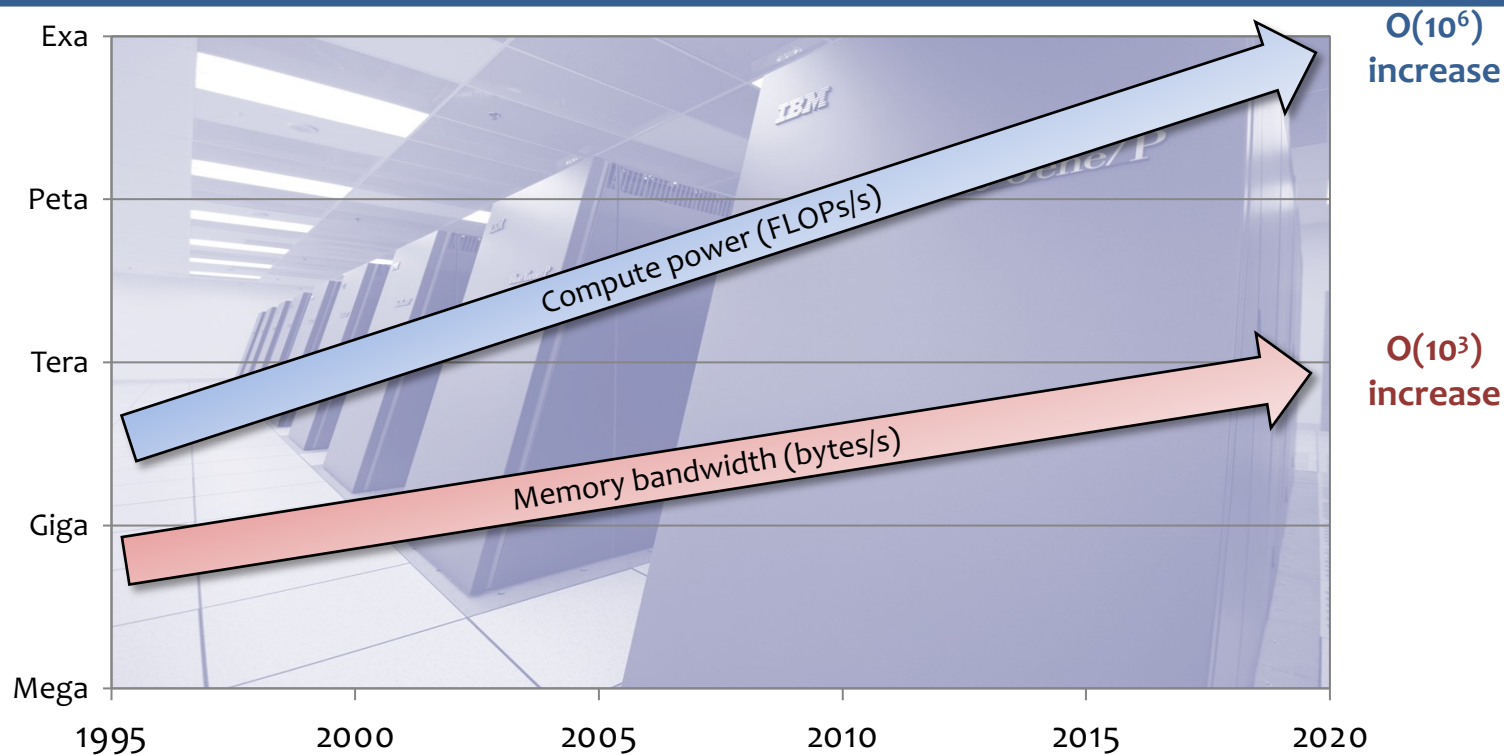
Data Analysis and Reduction for Big Scientific Data 2022

Peter Lindstrom

November 13, 2022



Data movement will dictate performance and power usage at exascale



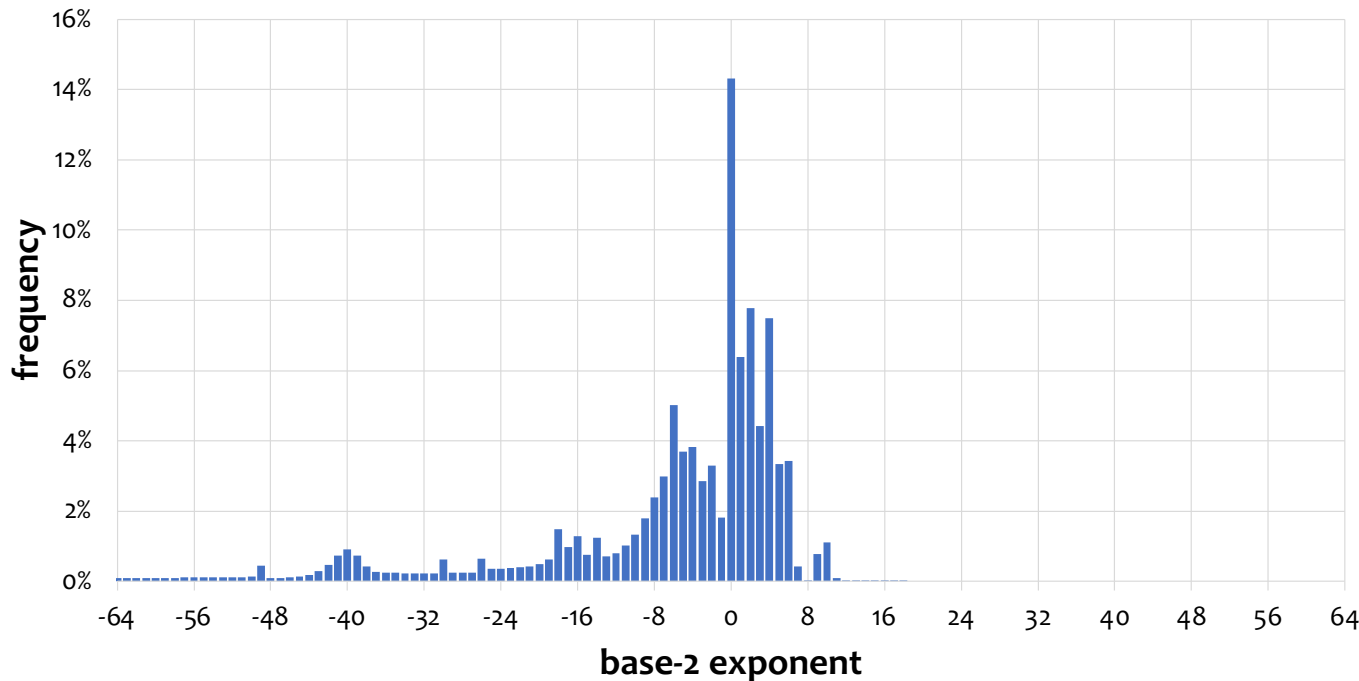
Uneconomical floating types and excessive precision are contributing to data movement

- IEEE 754 takes “one size fits all” approach when balancing range and precision
 - **Insufficient precision** for some computations, e.g., catastrophic cancellation
 - **Insufficient dynamic range** for others, e.g., under- or overflow in exponentials
- Single vs. double precision dichotomy $\Rightarrow$ super sizing floats
 - Do we really need 11 exponent bits?
 - Allows representing 10 quattuordecillion googol googol universes measured in Planck lengths!
 - Many **fraction bits are contaminated by error** (round-off, truncation, iteration, ...)
 - Pushing error around and computing on it is wasteful!
- Spatial correlation leads to redundancy in representation
 - Physics fields are usually continuous/smooth

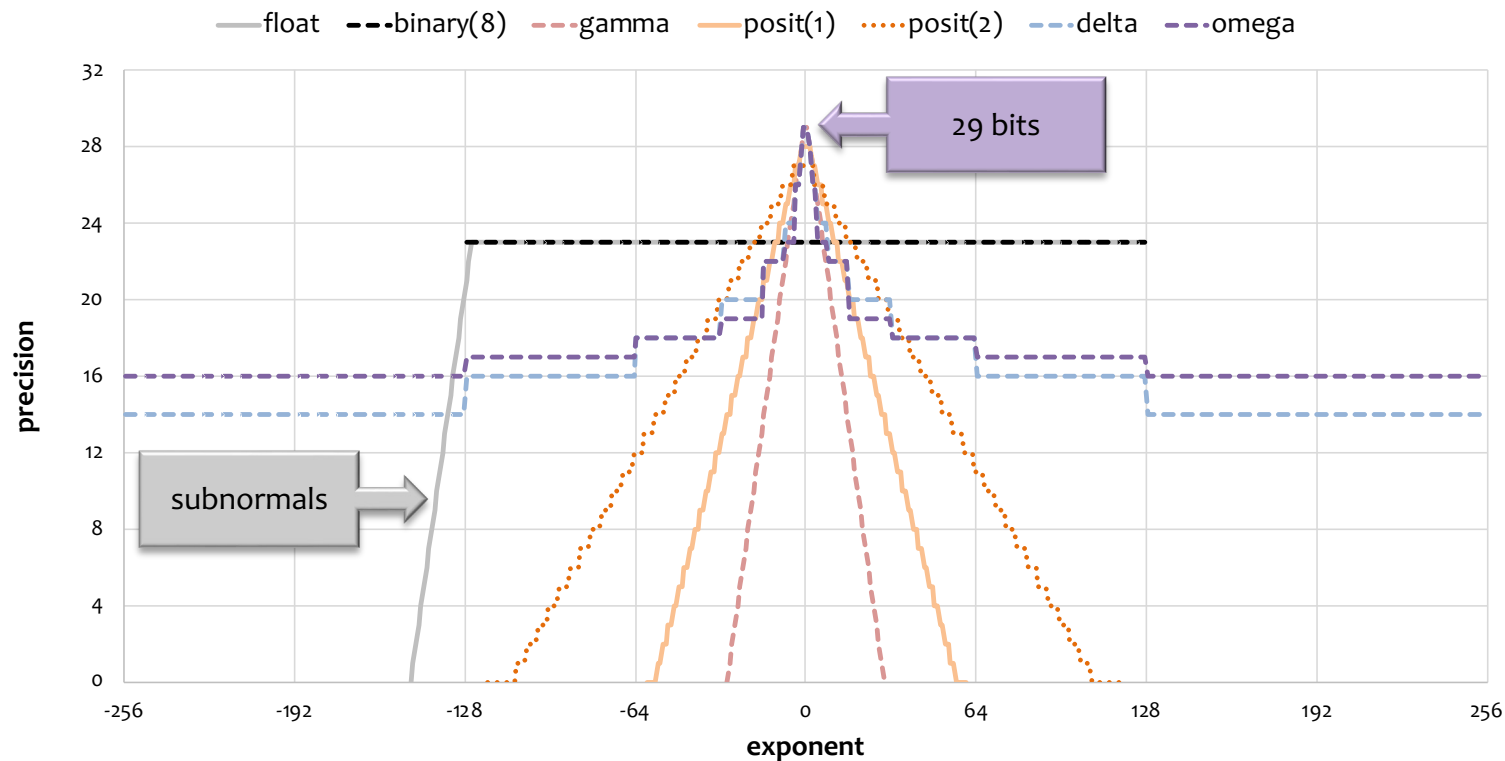
To reduce data movement cost, we must make every bit count

Most exponents are close to zero—Posits use variable-length coding of exponent [Gustafson & Yonemoto '17]

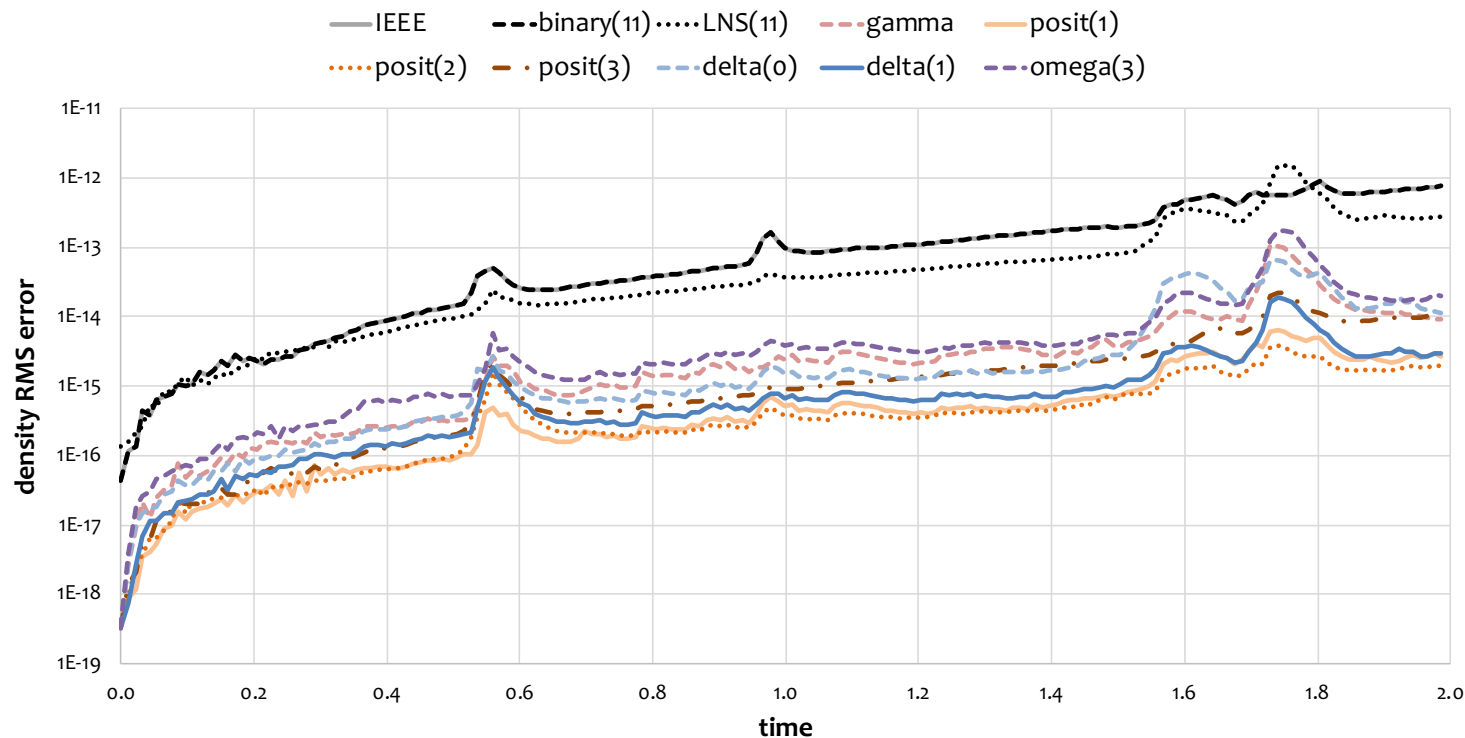
Combined exponent distribution from 32 data sources



Variable-length exponents lead to tapered precision: 6-9 more bits of precision than IEEE 754 for numbers near one

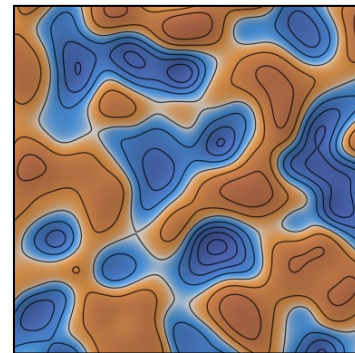
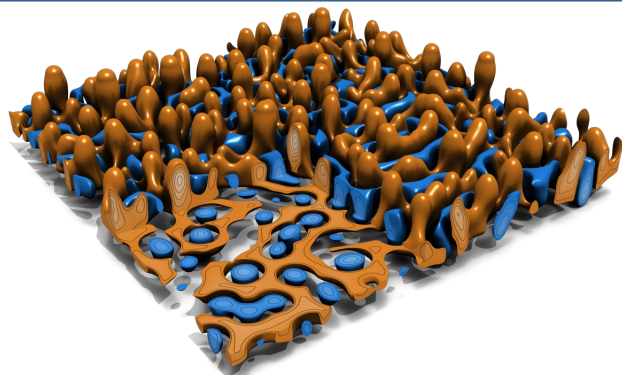


IEEE 754 consistently is the least accurate floating-point representation in numerical calculations (64-bit types)

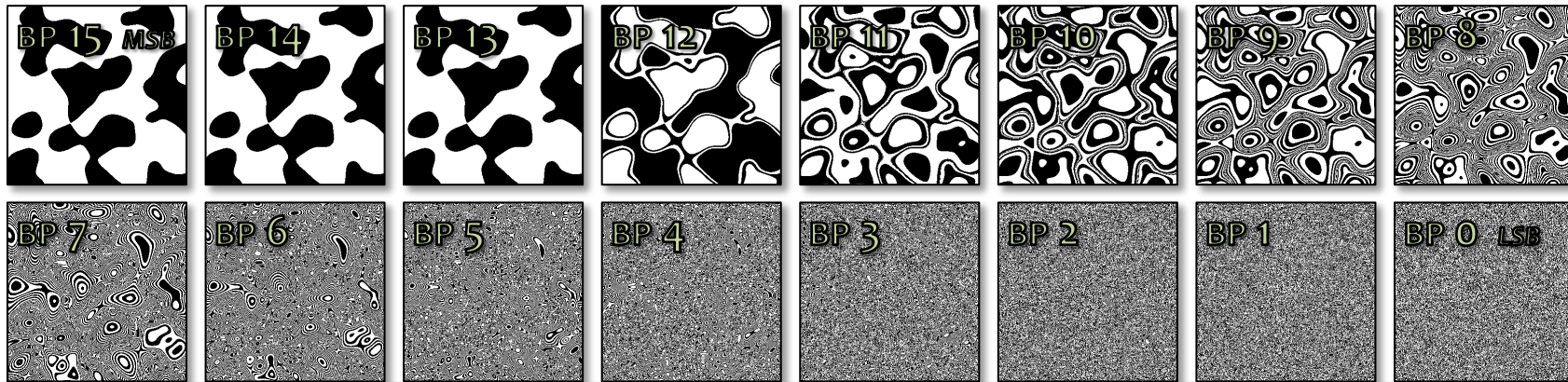


[Lindstrom et al., "Universal Coding of the Reals: Alternatives to IEEE Floating Point," CoNGA 2018]

Posits and other scalar representations do not account for spatial correlations in continuous fields



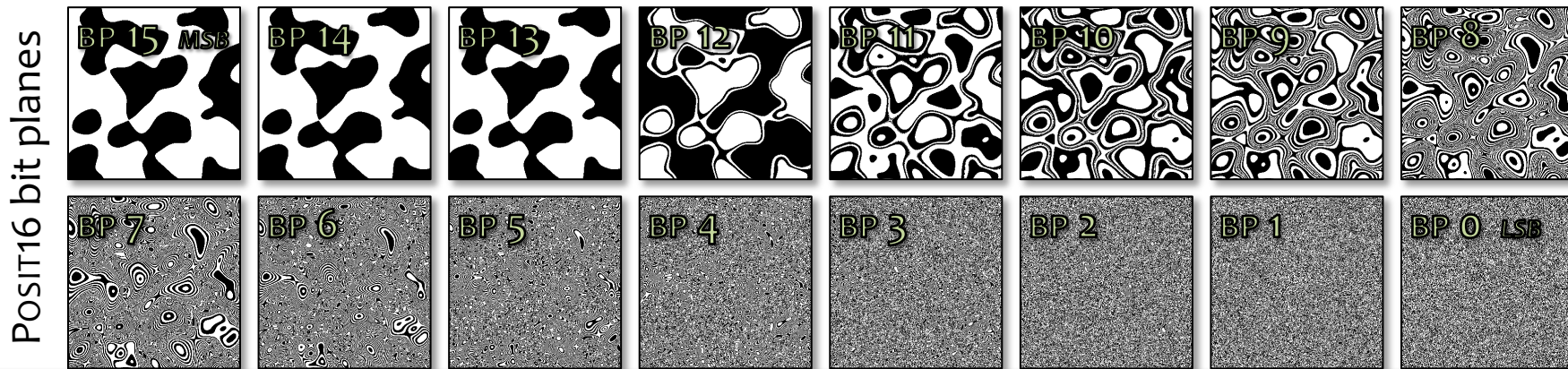
Posit16 bit planes



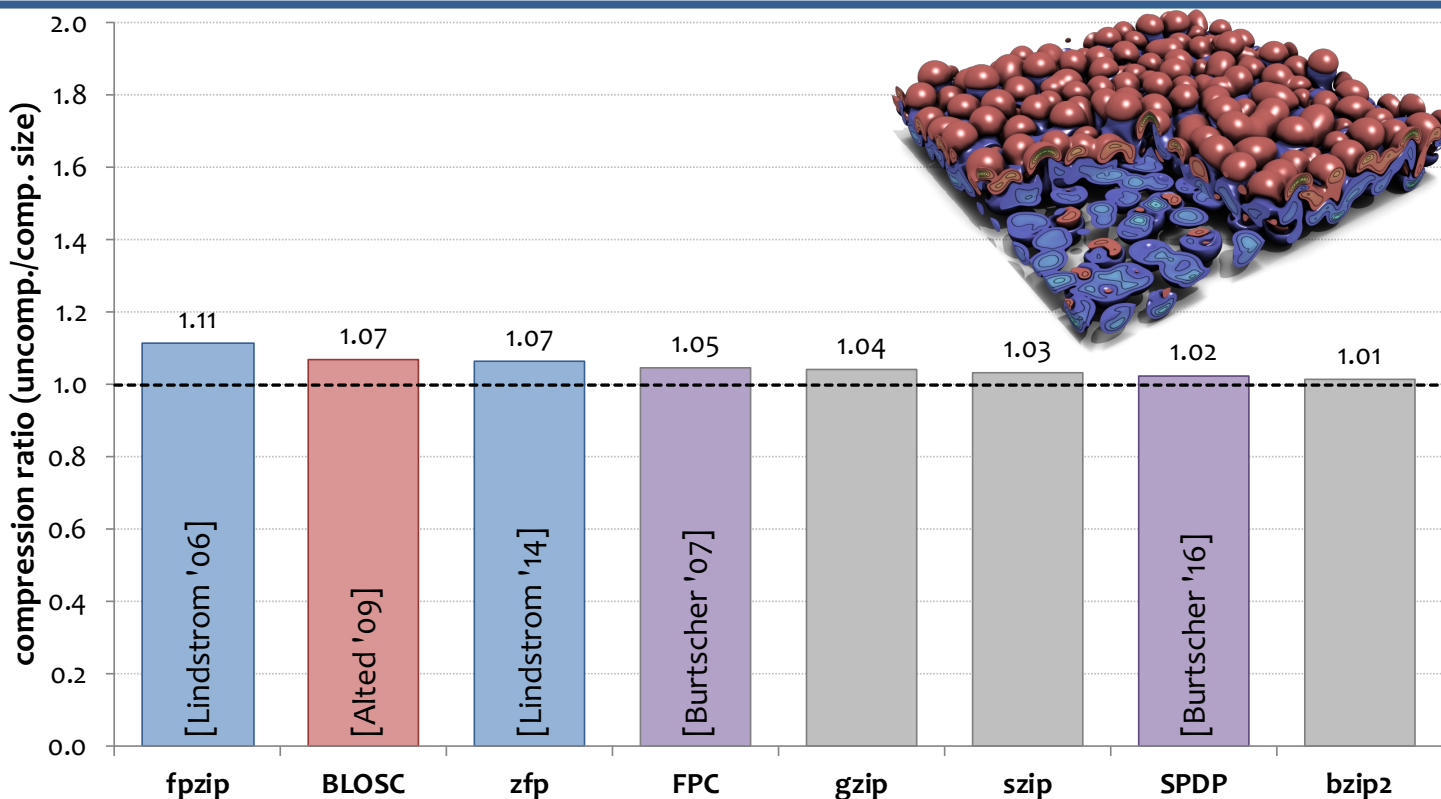
Posits and other scalar representations do not account for spatial correlations in continuous fields

- **Observation #1:** Consecutive bit planes are often correlated (low dynamic range)
- **Observation #2:** Adjacent grid points are often correlated (spatial smoothness)

Redundancy in representation wastes storage, sacrifices accuracy

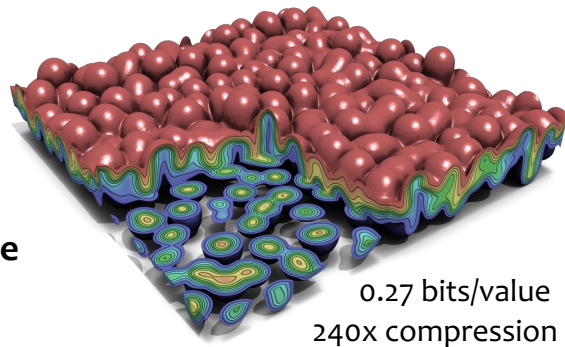


(64-bit) floating-point data does not compress well losslessly

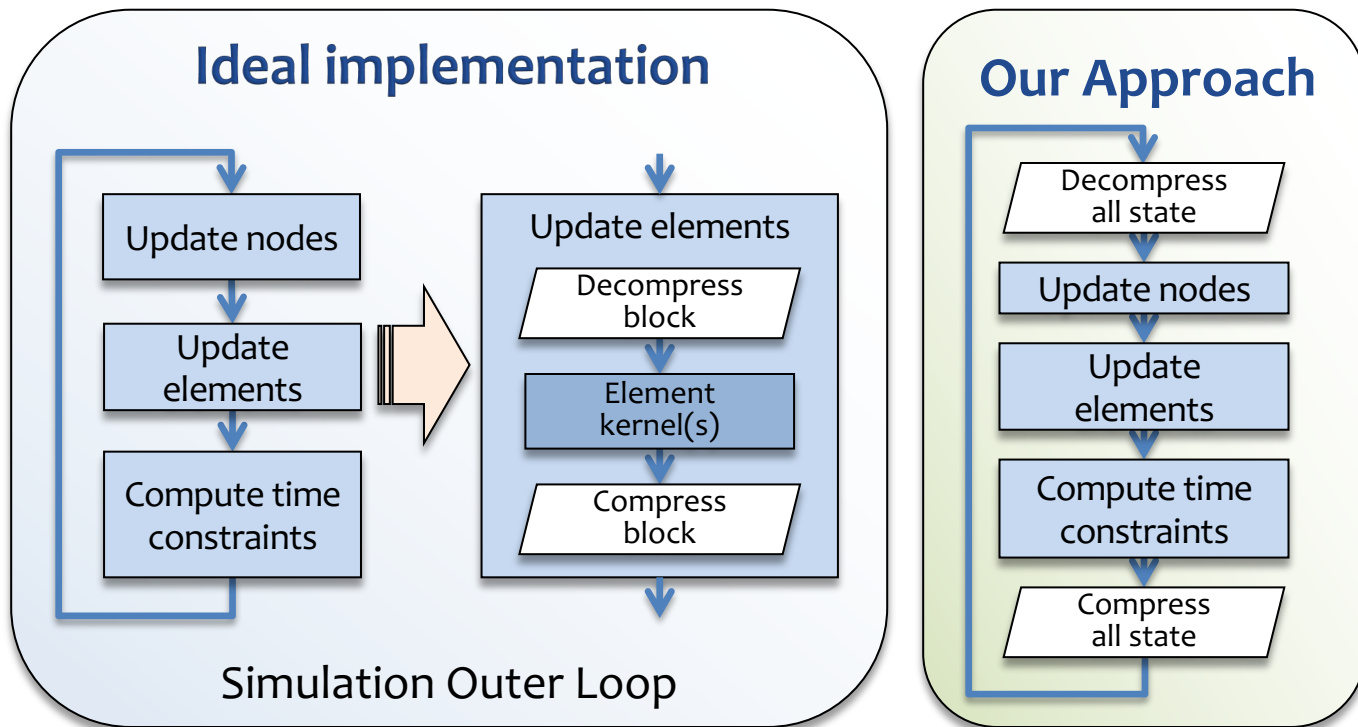


Lossy compression enables greater reduction, but is often met with skepticism by scientists

- Large improvements in compression are possible by allowing even small errors
 - Simulation often computes on meaningless bits
 - Round-off, truncation, iteration, model **errors abound**
 - Many trailing floating-point bits are effectively **random noise**
- Still, lossy compression often makes scientists nervous
 - Even though other forms of lossy **data reduction** are ubiquitous
 - **Decimation** in space and/or time (e.g., store every 100 time steps)
 - **Averaging** (hourly vs. daily vs. monthly averages)
 - **Truncation** to single precision (e.g., for analysis files)
 - State-of-the-art compressors support **error tolerances**



Can lossy-compressed in-memory storage of numerical simulation state be tolerated?

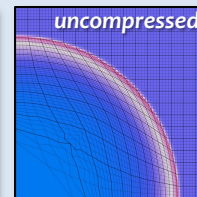
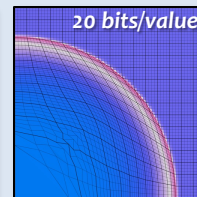
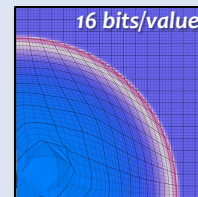


[Laney et al., "Assessing the effects of data compression in simulations using physically motivated metrics," SC 2013]

Using lossy FPZIP to store simulation state compressed, we have shown that 4x lossy compression can be tolerated

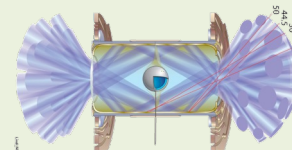
Lagrangian shock hydrodynamics

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



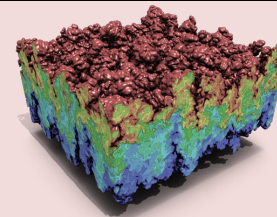
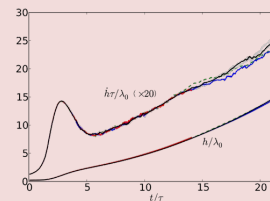
Laser-plasma multi-physics

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



High-order Eulerian hydrodynamics

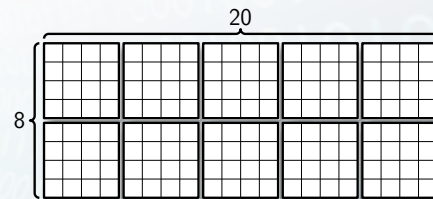
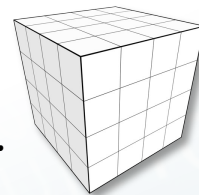
- QoI: *Rayleigh-Taylor mixing layer thickness*
- 10,000 time steps
- At **4x compression**, relative **error < 0.2%**



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

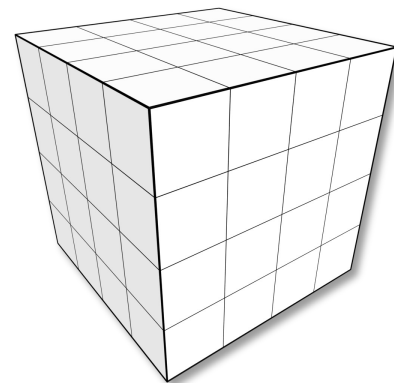
What is ZFP?

- **Answer #1:** An efficient floating **number format** for tiny tensors
 - Alternative to IEEE/SSE/AVX/tensor core registers, bfloats, posits, flexpoint, ...
- **Answer #2:** An implementation of **multidimensional arrays**
 - Alternative to std::vector, Eigen/GSL/Kokkos/NumPy arrays, ...
- **Answer #3:** A fast, **streaming compressor** for numerical data
 - Alternative to gzip, bzip2, blosc, fpzip, JPEG, ...



ZFP is a compressed number format for multi-dimensional floating-point arrays

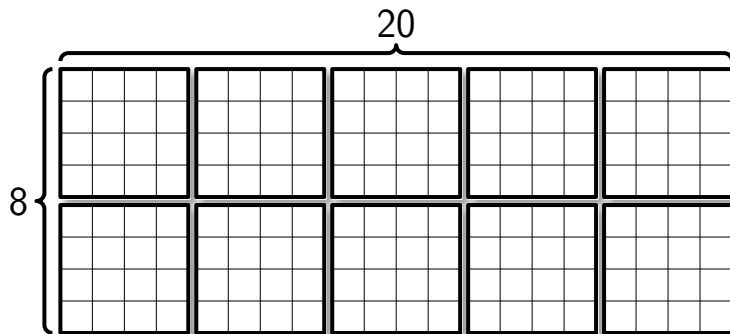
- ZFP compactly represents small blocks of real values
 - Encodes d -dimensional **block of 4^d values** as variable-length bit string
 - **Fixed-length code** obtained via bit string truncation
 - Analogous to FP rounding: $1/5 = 0.001100110011... \approx 0.0011$
 - “Common” blocks have shorter codes $\Rightarrow$ less or no round-off error
 - **H/W friendly**: integer additions and bitwise operations
- ZFP **replaces IEEE 754** as format for numerical computations
 - **Less storage** for same accuracy
 - **Higher accuracy** for same storage
 - **Fine-grained precision** selection



[Lindstrom, “Fixed-rate compressed floating-point arrays,” IEEE TVCG 2014]

ZFP multi-dimensional arrays offer in-memory compressed storage with high-speed read and write access

- ZFP provides C++ classes for multi-dimensional arrays
 - Read & write **random access** at block granularity
 - Block decomposition is transparent to user
 - User specifies **memory footprint** or **error tolerance**
 - **Conventional API:** C++ operator overloading hides complexity of (de)compression
 - `double a[n]` $\Leftrightarrow$ `std::vector<double> a(n)` $\Leftrightarrow$ `zfp::array<double> a(n, bits_per_value)`
 - C, experimental NumPy APIs available



ZFP's C++ compressed arrays can replace STL vectors and C arrays with minimal code changes

// example using STL vectors

```
std::vector<double> u(nx * ny, 0.0);
u[x0 + nx*y0] = 1;
for (double t = 0; t < tfinal; t += dt) {
    std::vector<double> du(nx * ny, 0.0);
    for (int y = 1; y < ny - 1; y++)
        for (int x = 1; x < nx - 1; x++) {
            double uxx = (u[(x-1)+nx*y] - 2*u[x+nx*y] + u[(x+1)+nx*y]) / dxx;
            double uyy = (u[x+nx*(y-1)] - 2*u[x+nx*y] + u[x+nx*(y+1)]) / dyy;
            du[x + nx*y] = k * dt * (uxx + uyy);
        }
    for (int i = 0; i < u.size(); i++)
        u[i] += du[i];
}
```

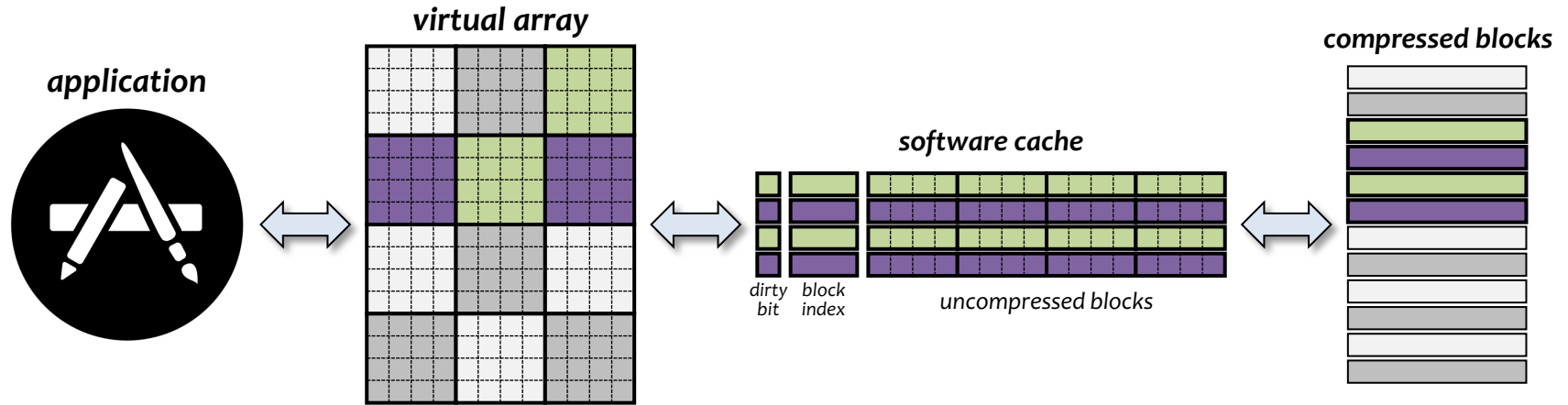
// example using ZFP arrays

```
zfp::array2<double> u(nx, ny, bits_per_value);
u(x0, y0) = 1;
for (double t = 0; t < tfinal; t += dt) {
    zfp::array2<double> du(nx, ny, bits_per_value);
    for (int y = 1; y < ny - 1; y++)
        for (int x = 1; x < nx - 1; x++) {
            double uxx = (u(x-1, y) - 2*u(x, y) + u(x+1, y)) / dxx;
            double uyy = (u(x, y-1) - 2*u(x, y) + u(x, y+1)) / dyy;
            du(x, y) = k * dt * (uxx + uyy);
        }
    for (int i = 0; i < u.size(); i++)
        u[i] += du[i];
}
```

■ required changes

■ optional changes for improved readability

ZFP arrays limit data loss via a small write-back cache

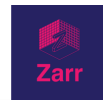


ZFP supports fast, parallel (de)compression of whole arrays

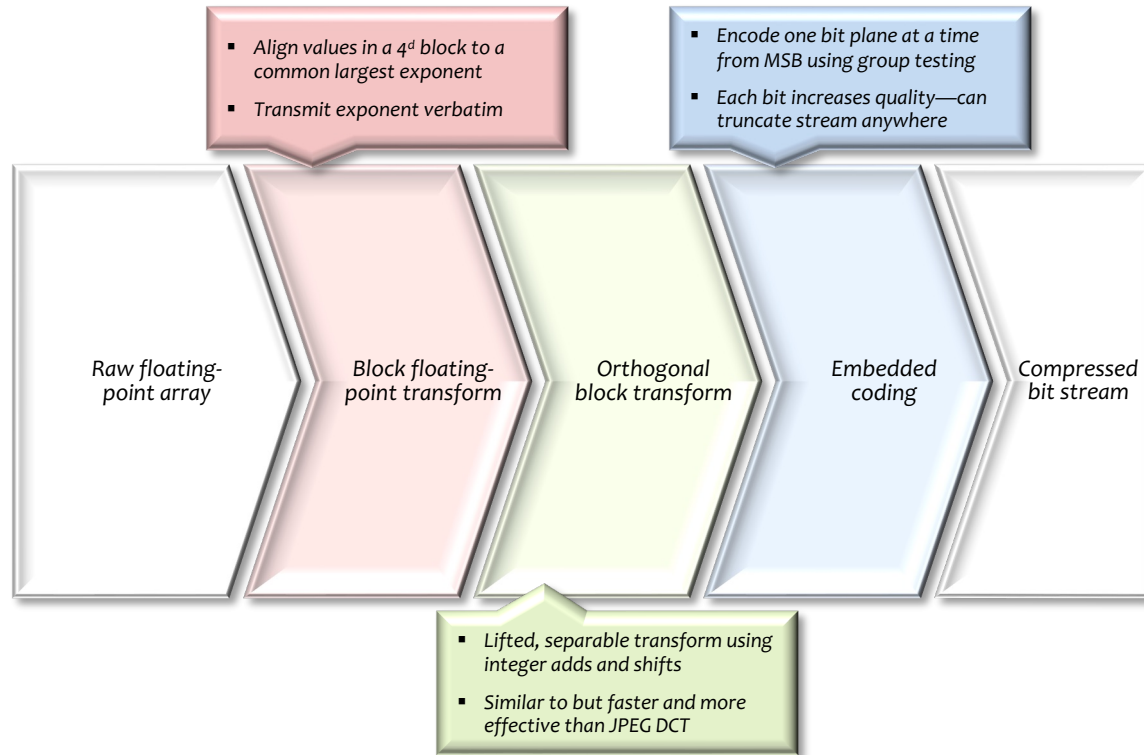
- ZFP supports streaming compression for I/O, communication, storage, ...
 - Supports absolute and relative **error tolerances** and **lossless** compression
 - Serial, **OpenMP**, **CUDA**, **HIP**, **SYCL**, **AVX**, **NEC VE**, and **FPGA** back-ends
 - Up to **700 GB/s** parallel throughput
 - **C**, **C++**, **Python**, **Fortran** bindings
 - 3rd party Julia & Rust bindings available
- ZFP is supported by numerous tools and applications
 - I/O: **ADIOS2**, **BLOSC**, **HDF5[®]**, **OpenZGY**, **Silo**, **Zarr**
 - Serialization: **Conduit** (ZFP array classes only)
 - Communication: **MVAPICH2-GDR**
 - Viz & data analysis: **ArcGIS**, **Open Inventor[™]**, **ParaView**, **TTK**, **VTK-m**
 - Other: **Intel[®] IPP**, **STRUMPACK**



MVAPICH



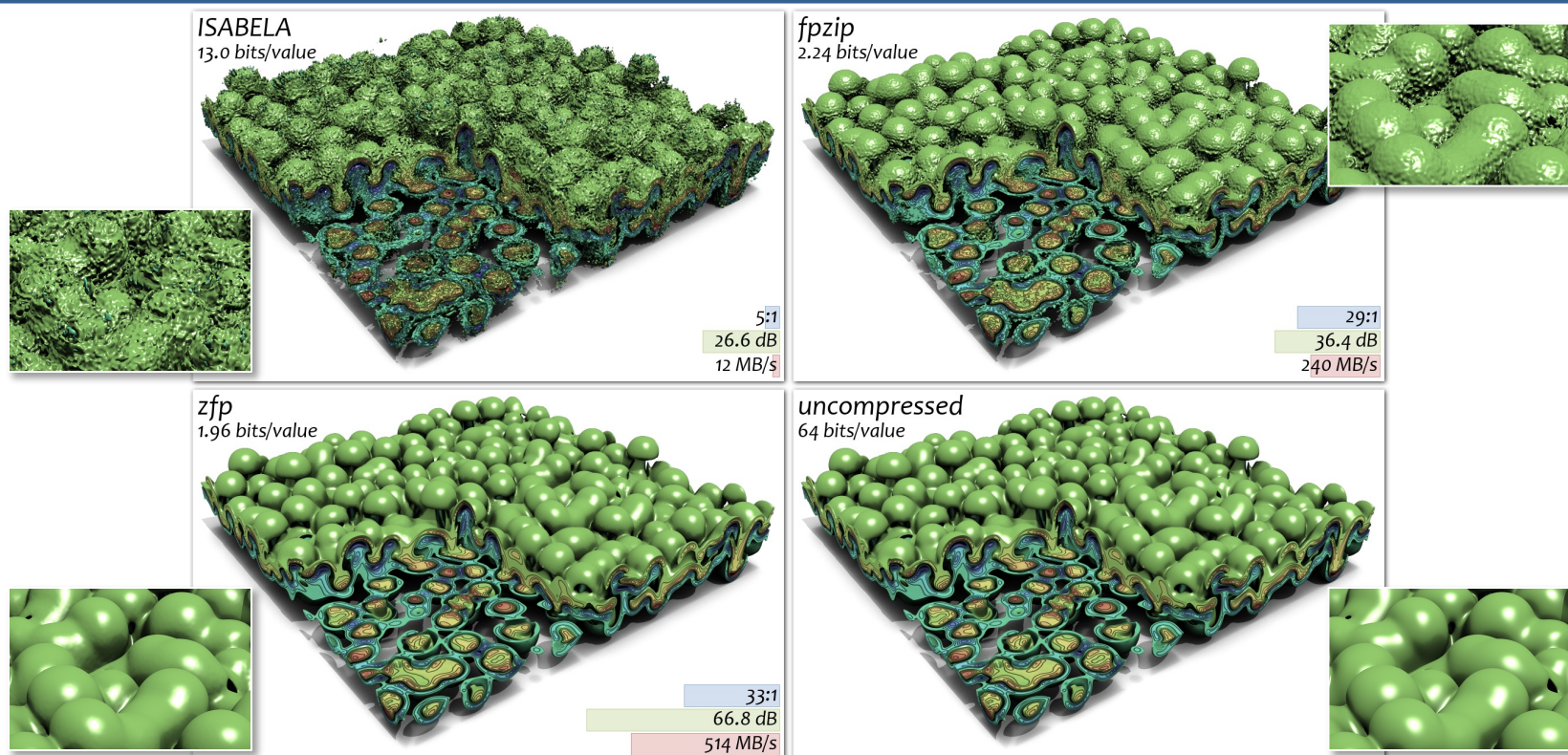
The ZFP encoder is comprised of three distinct components



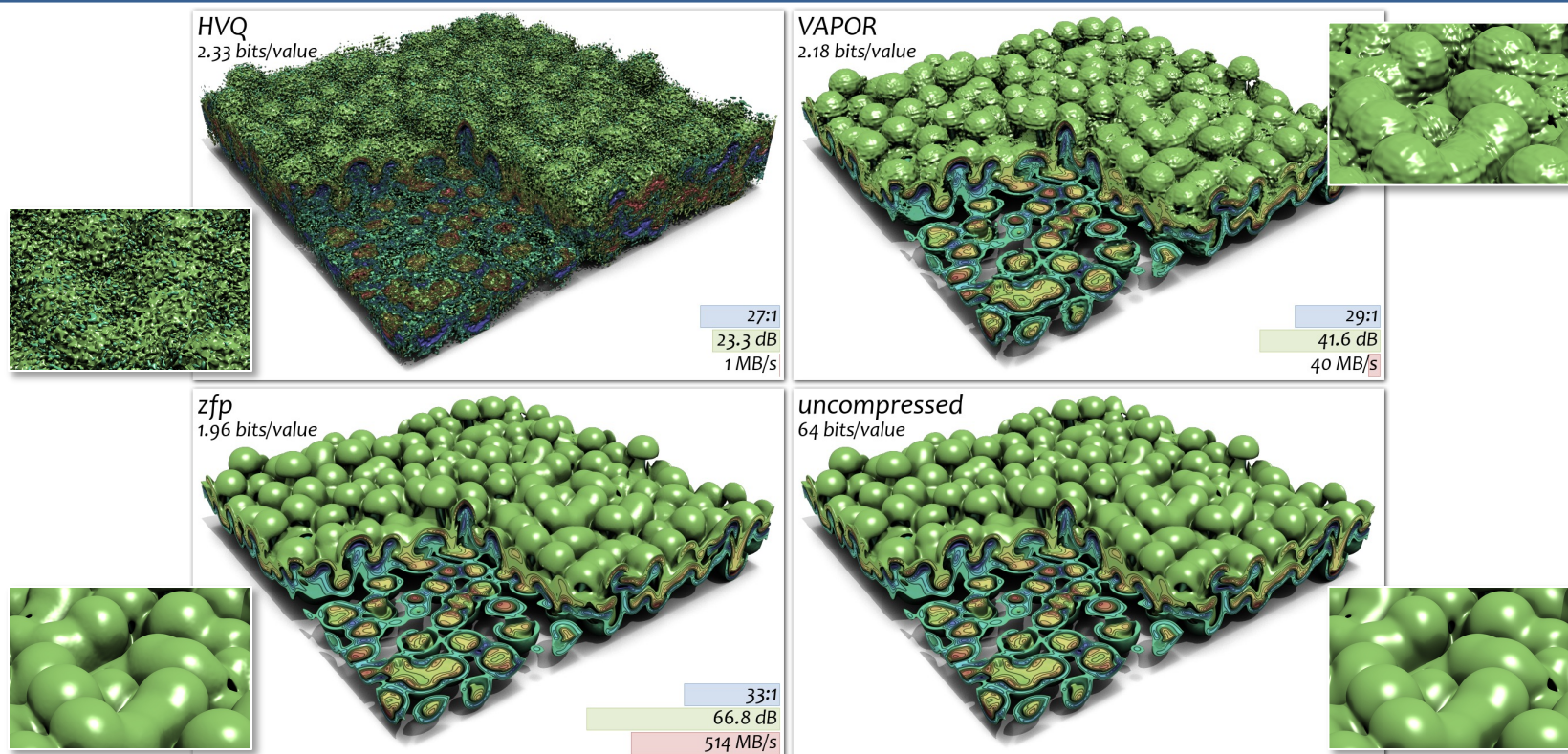
Got artifacts?



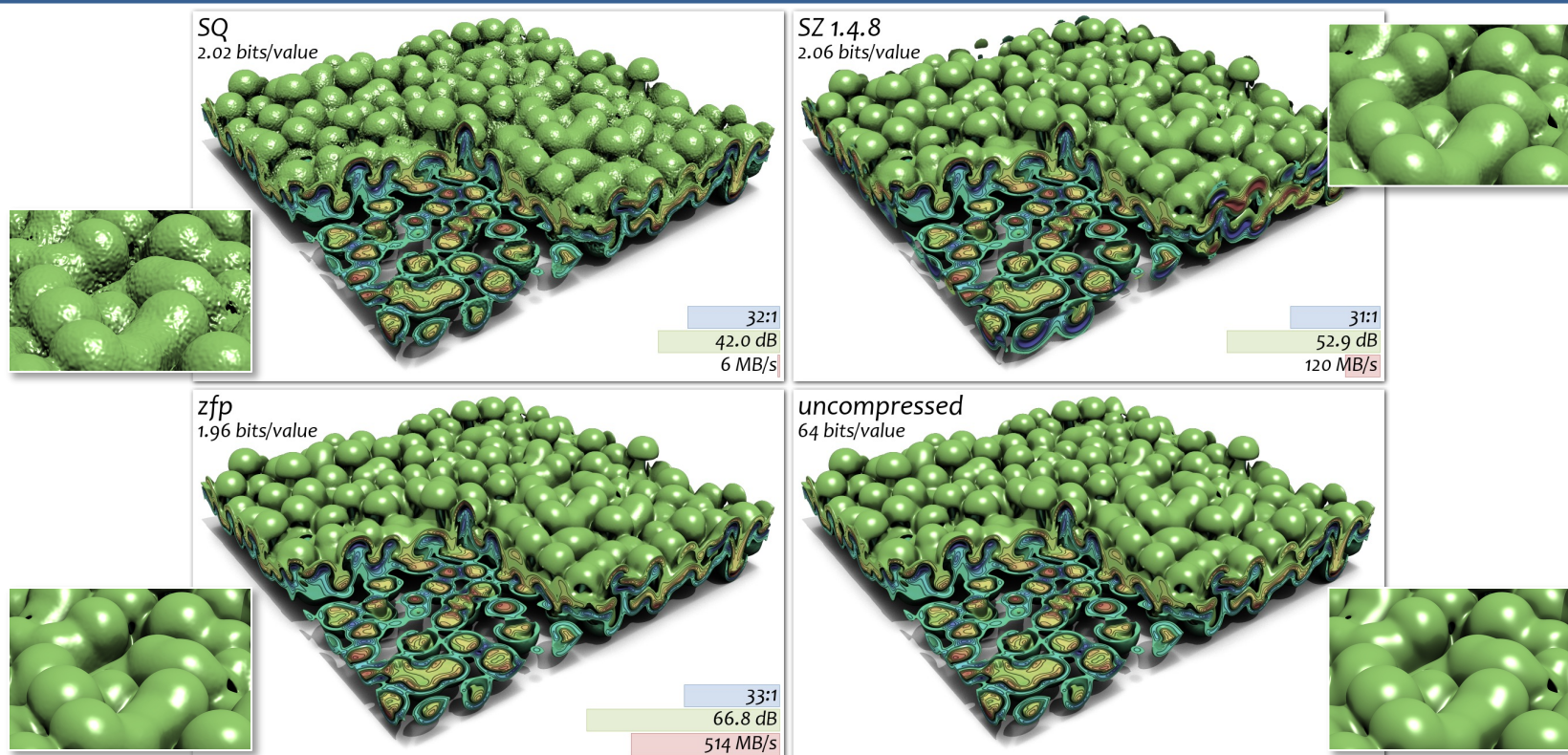
ZFP shows no artifacts in derivative computations (velocity divergence)



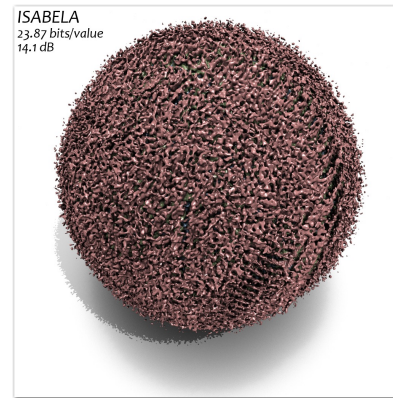
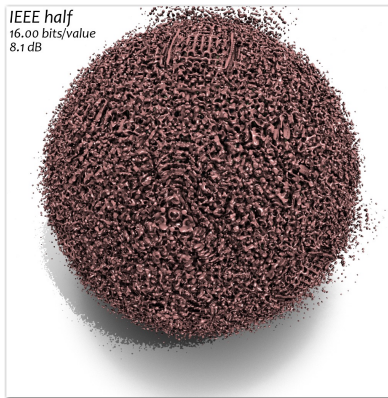
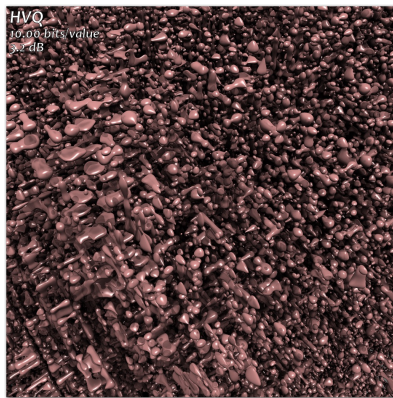
ZFP shows no artifacts in derivative computations (velocity divergence)



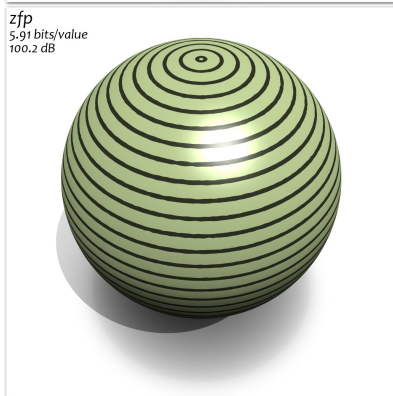
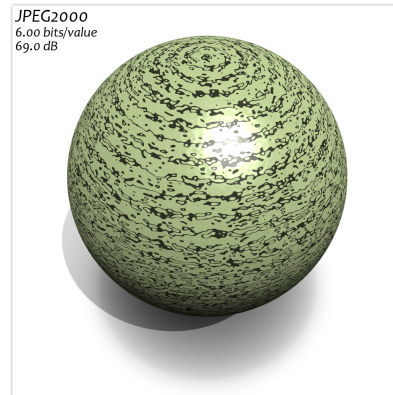
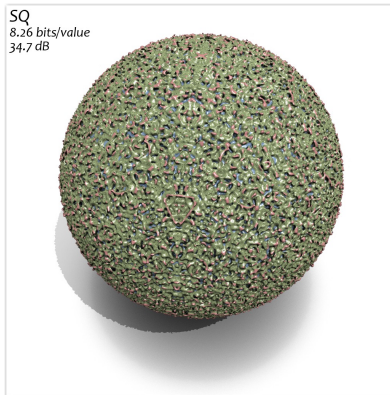
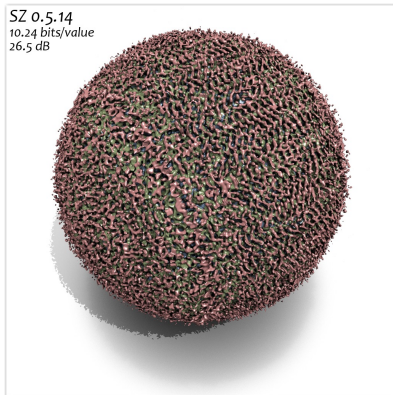
ZFP shows no artifacts in derivative computations (velocity divergence)



ZFP shows few artifacts in 2nd, 3rd derivative computations (Laplacian and highlight lines)

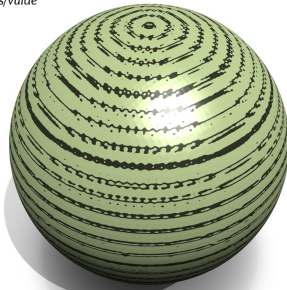


ZFP shows few artifacts in 2nd, 3rd derivative computations (Laplacian and highlight lines)

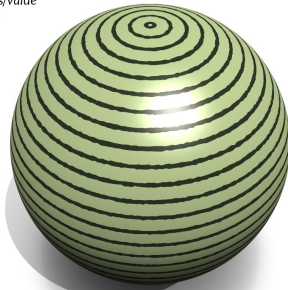


ZFP shows few artifacts in 2nd, 3rd derivative computations (Laplacian and highlight lines)

VAPOR
6.25 bits/value
77.5 dB



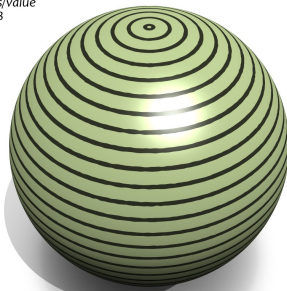
fpzip
5.91 bits/value
80.3 dB



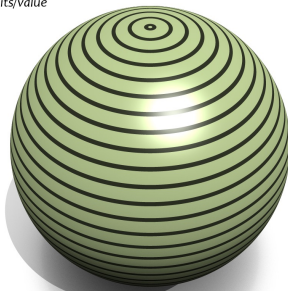
IEEE float
32.00 bits/value
86.3 dB



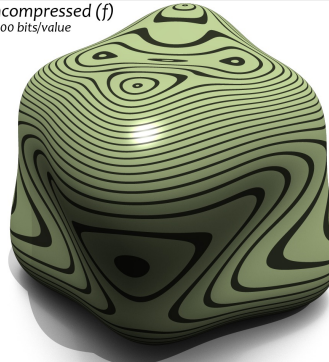
zfp
5.91 bits/value
100.2 dB



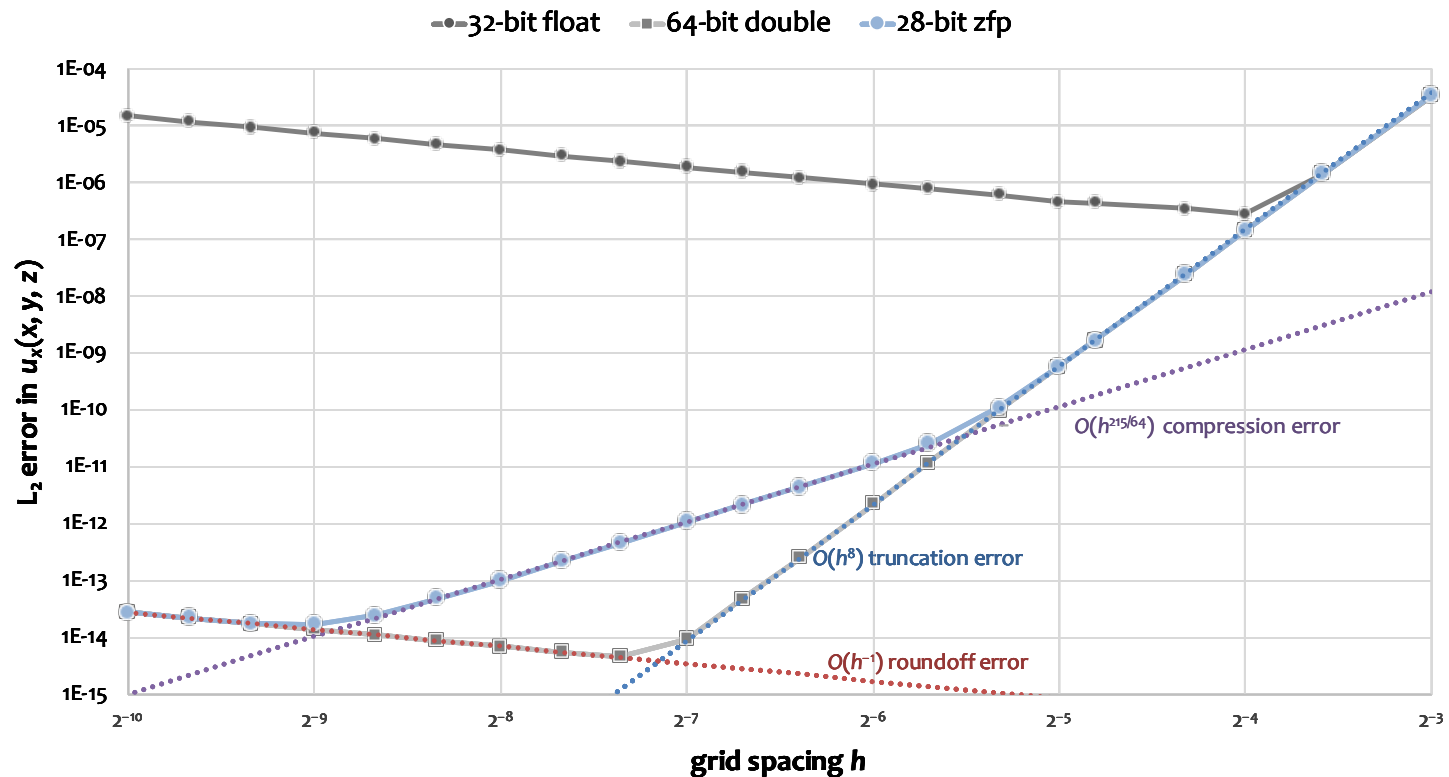
uncompressed (Δf)
64.00 bits/value



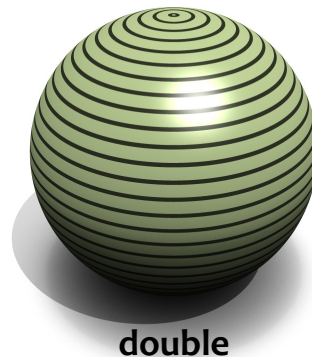
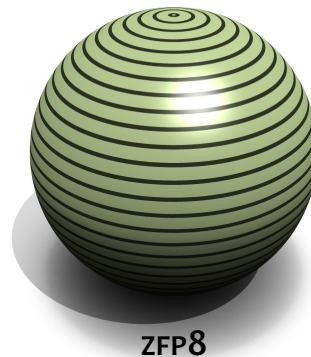
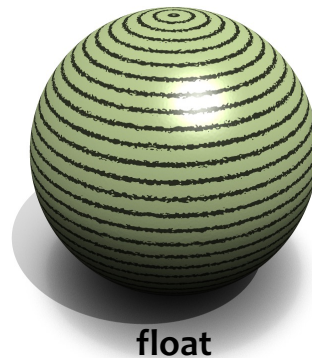
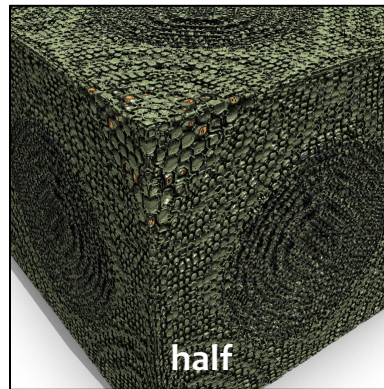
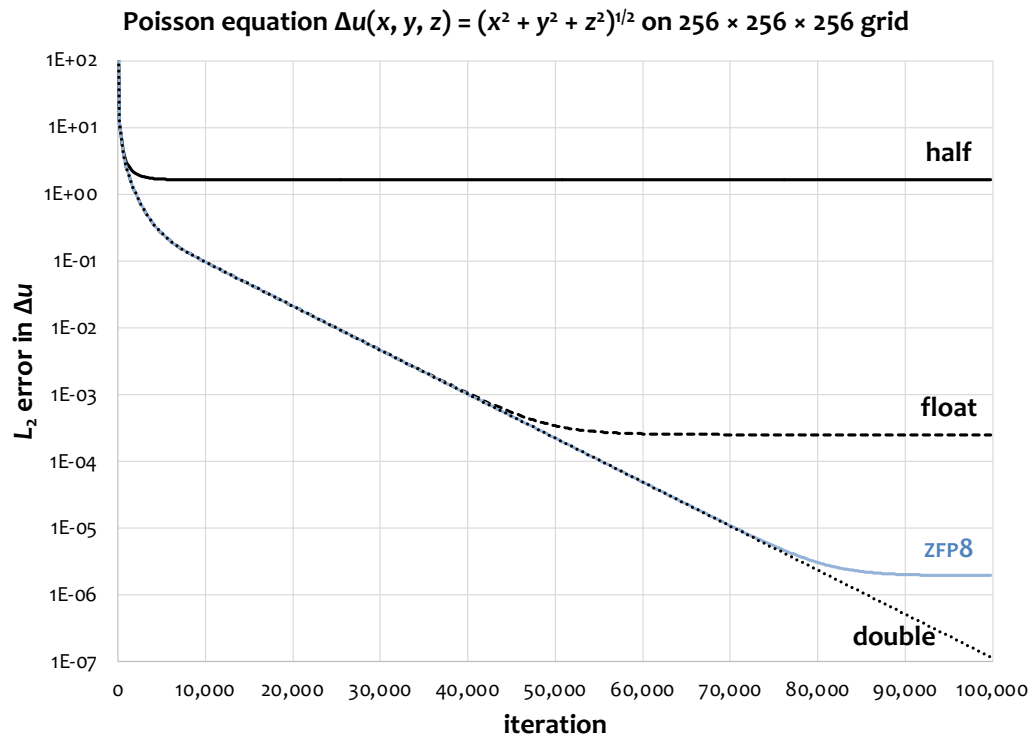
uncompressed (f)
64.00 bits/value



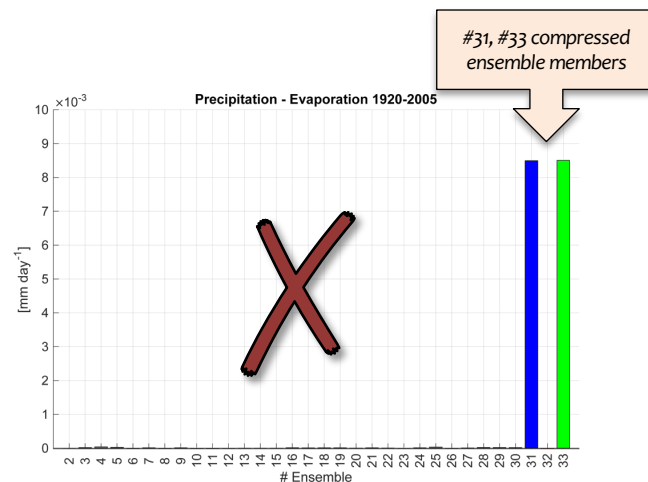
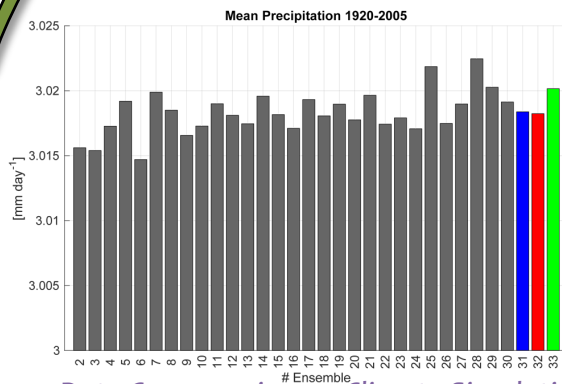
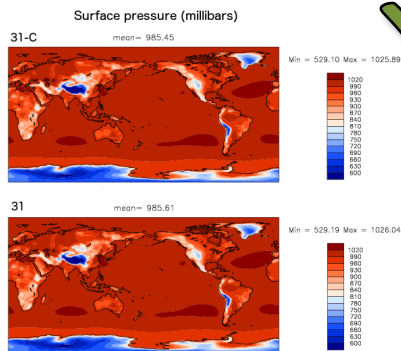
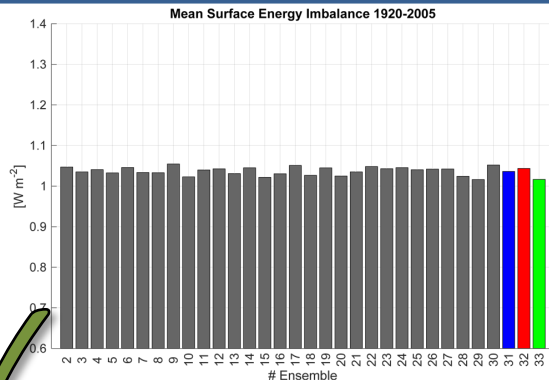
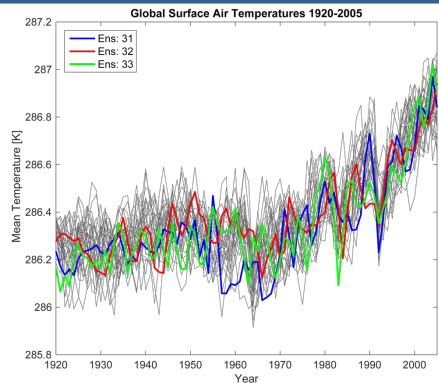
Contrary to conventional floating point, finite-difference accuracy using zfp *increases* with grid resolution



8-bit ZFP yields 2 orders of magnitude higher accuracy than 32-bit floats in Poisson equation solver



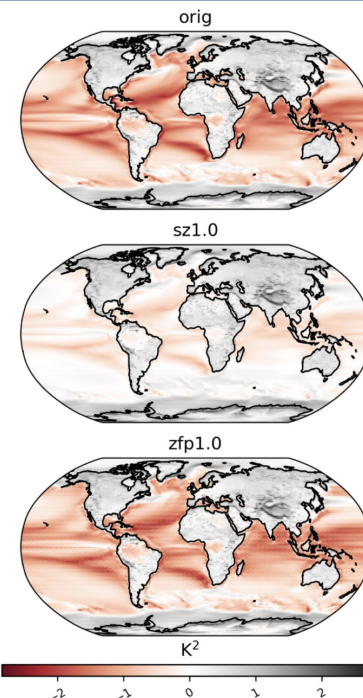
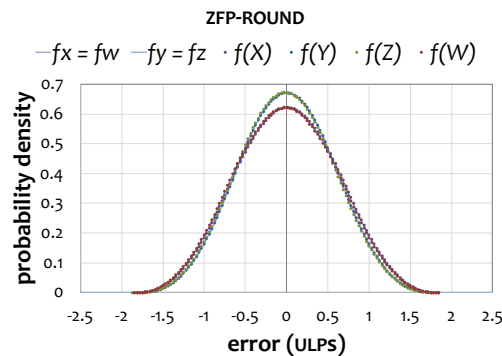
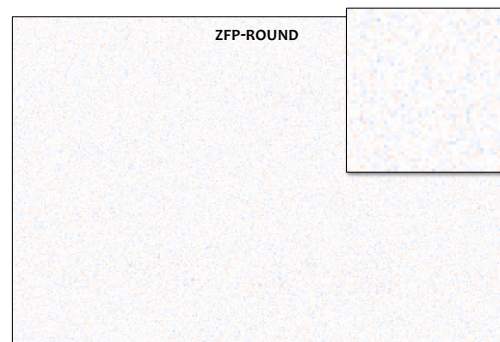
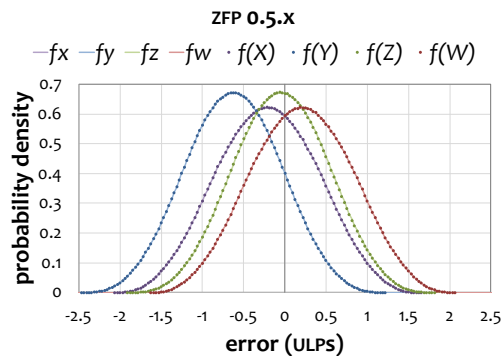
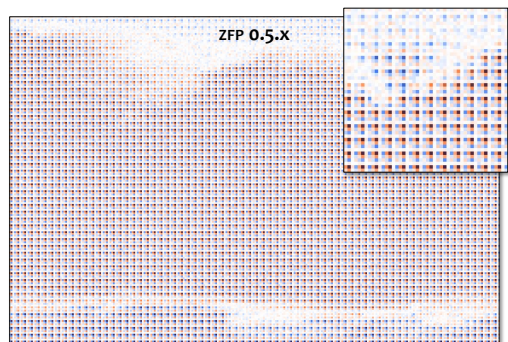
“Pepsi Challenge”: FPZIP systematic rounding toward zero leads to occasional issues in climate data analysis



[Baker et al., “Evaluating Lossy Data Compression on Climate Simulation Data within a Large Ensemble,” GMD 2016]

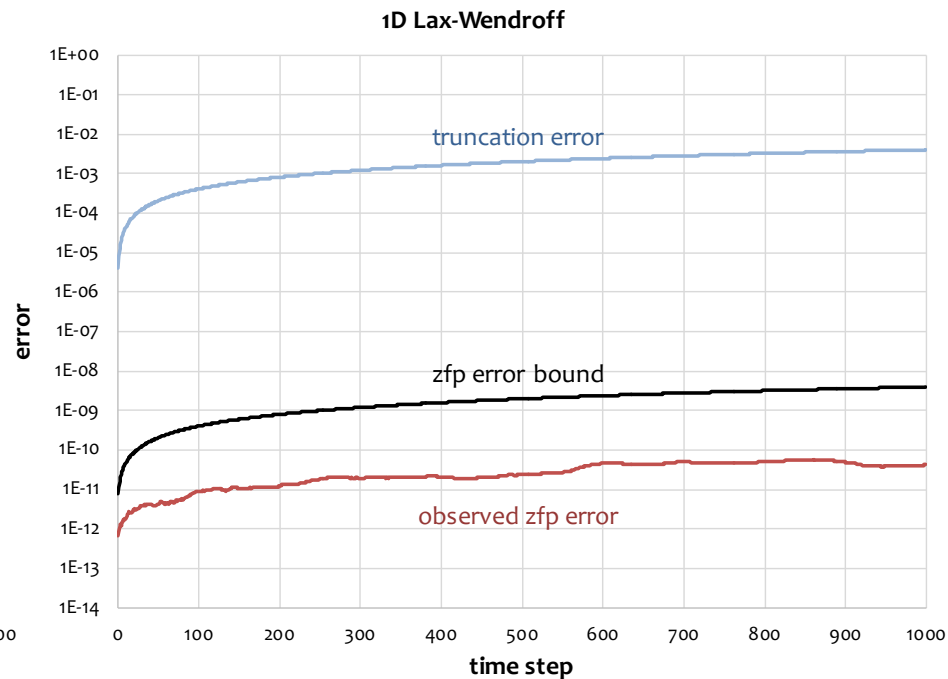
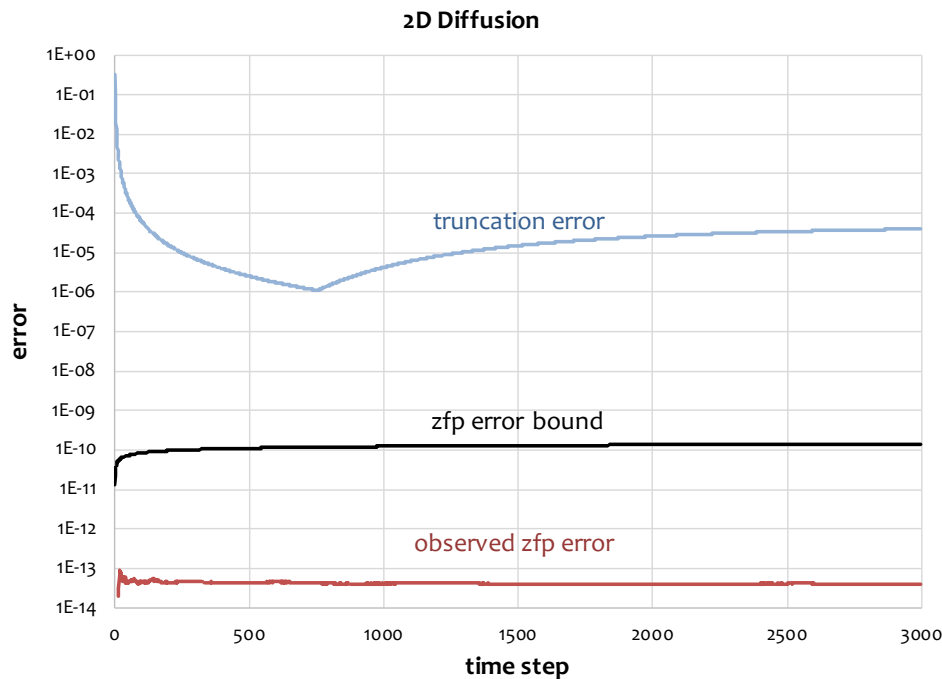
ZFP error correction ensures unbiased errors

time-averaged
compression error



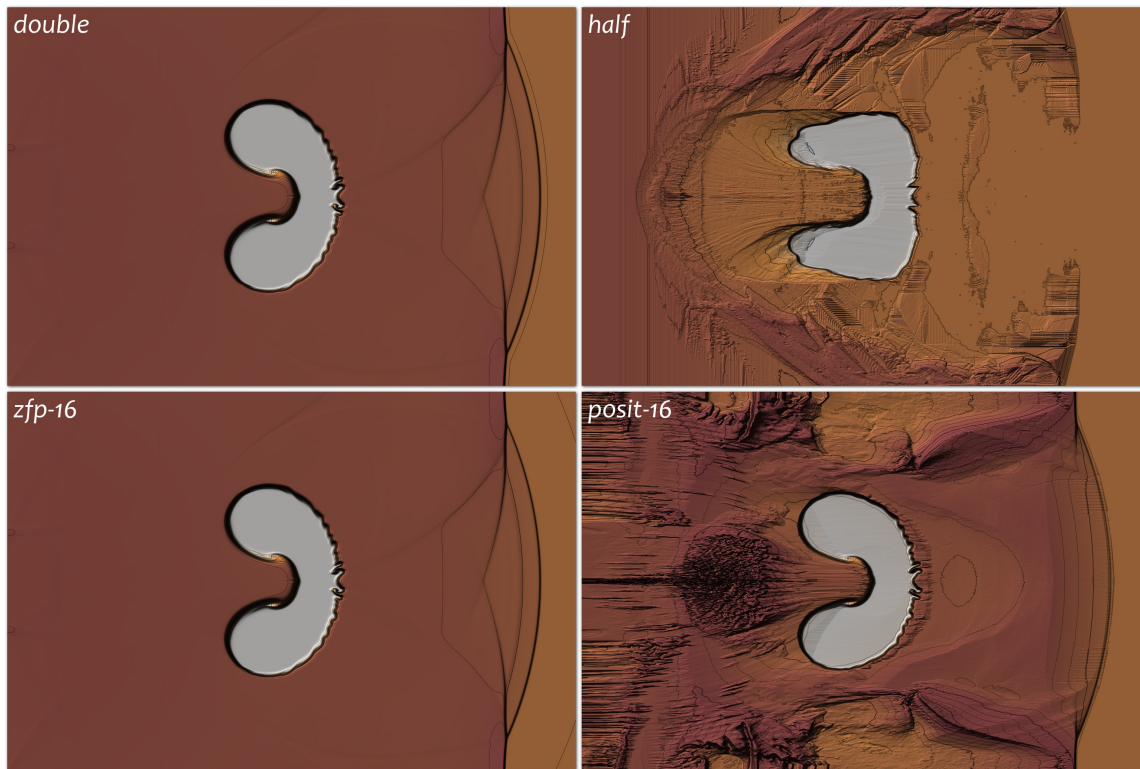
[Hammerling et al., “A Collaborative Effort to Improve Lossy Compression Methods for Climate Data,” DRBSD 2019]

We have developed rigorous error bounds for ZFP, both for static data and in iterative methods



[Fox et al., “Stability Analysis of Inline ZFP Compression for Floating-Point Data in Iterative Methods,” SIAM SISC 2020]

PDE computations using 16-bit zFP are qualitatively identical to 64-bit IEEE double precision



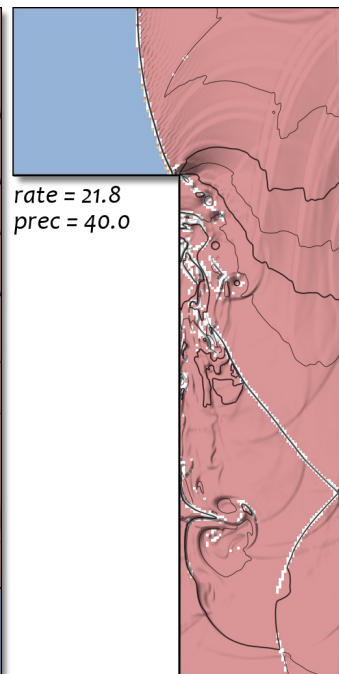
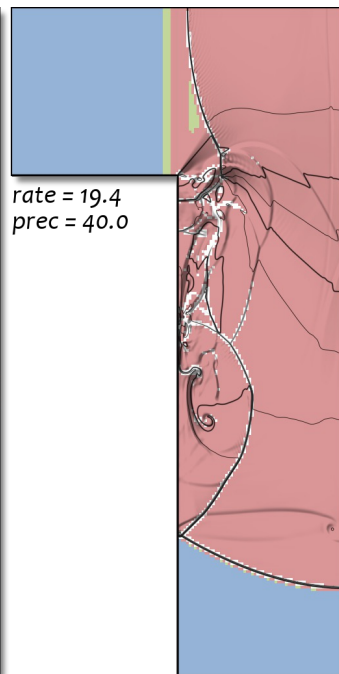
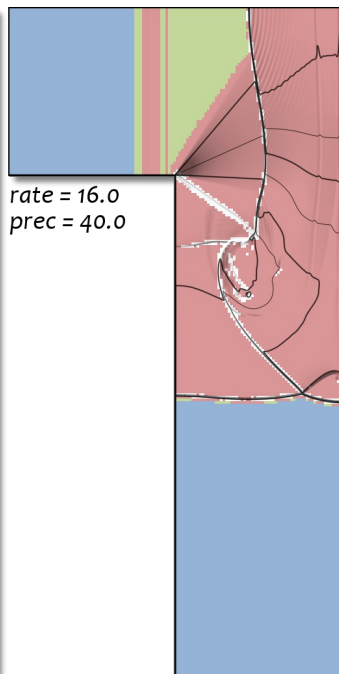
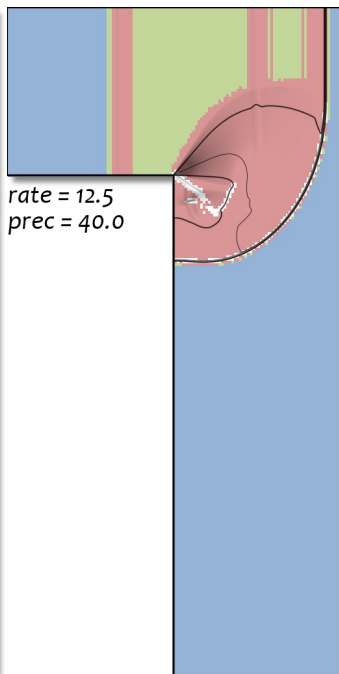
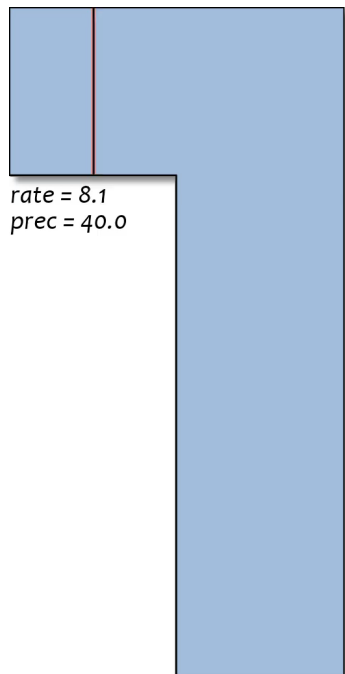
ZFP variable-rate C++ arrays allocate bits where needed

8 bits/value

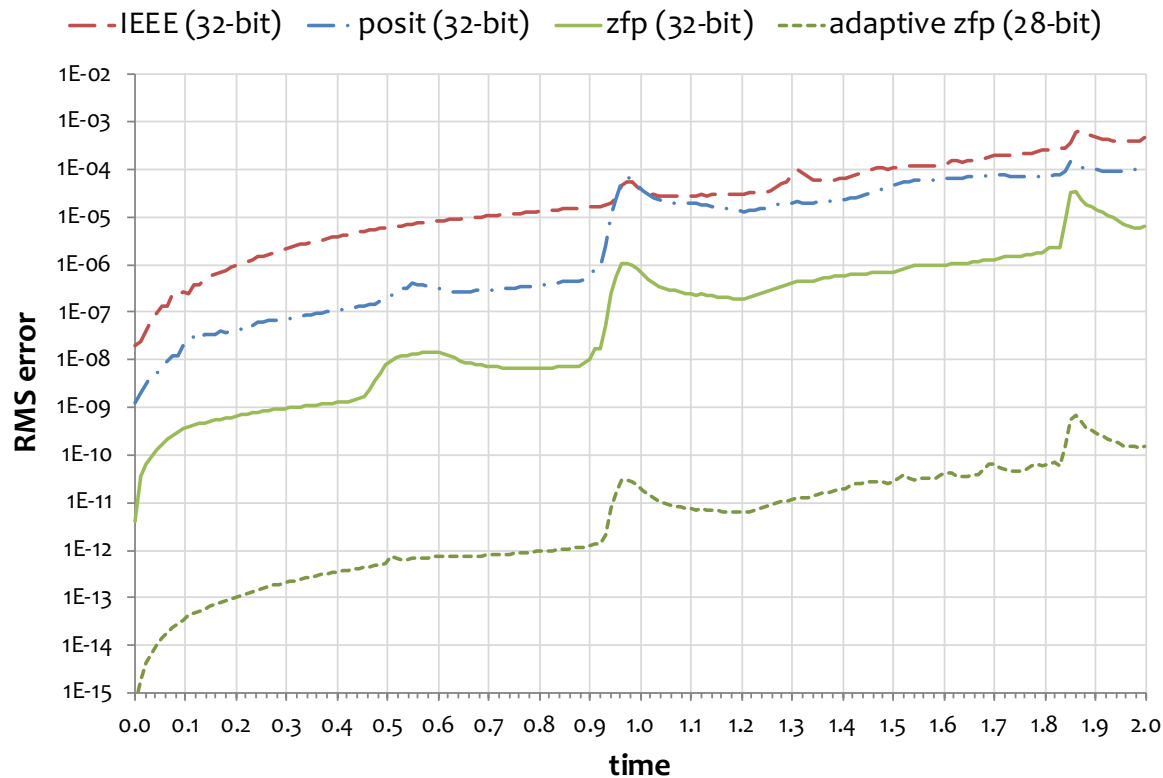
16 bits/value

32 bits/value

64 bits/value



Adaptive-rate ZFP increases accuracy over IEEE float by 6 orders of magnitude while using less storage



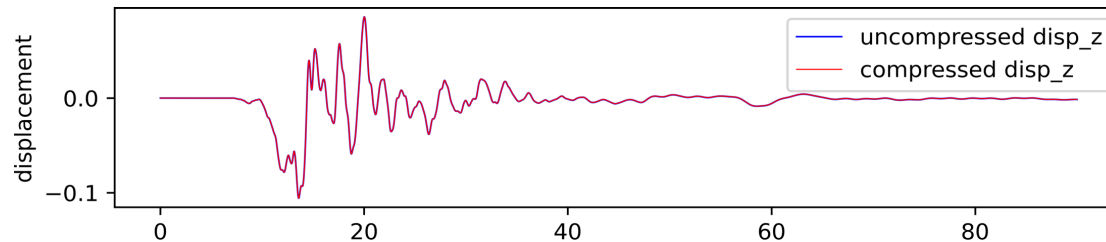
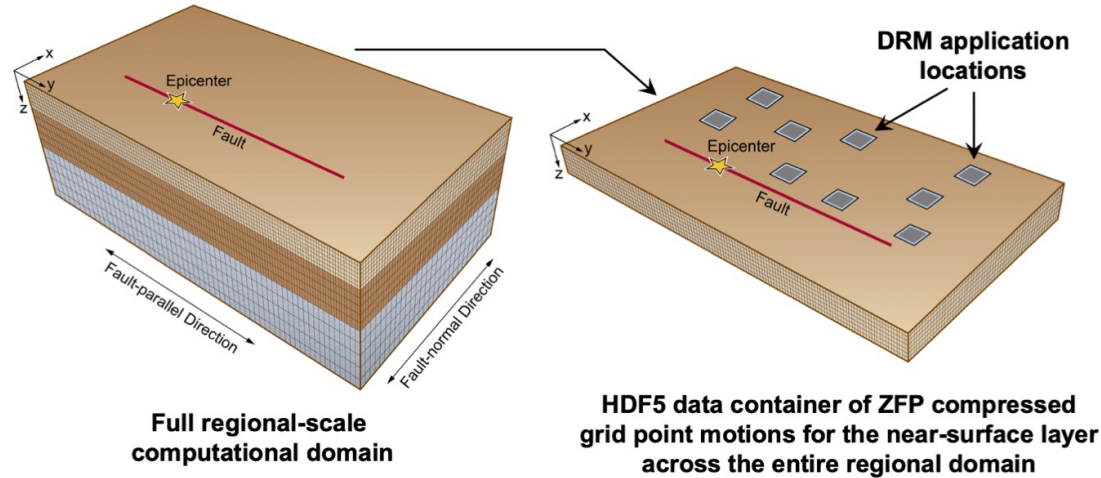
ZFP reduces I/O by 30x in CM1 tornado simulations

[Work done by Leigh Orf, UW-Madison]



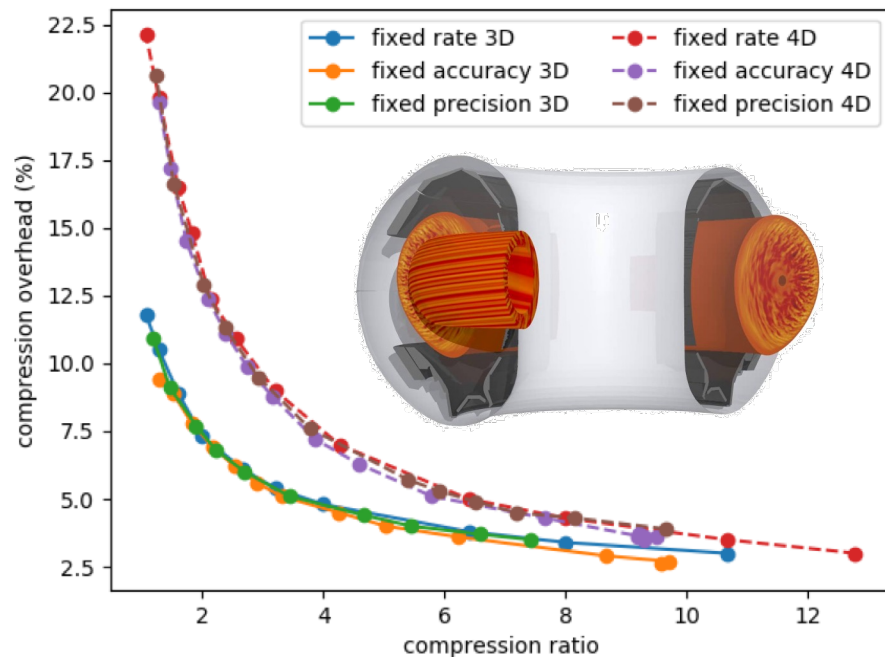
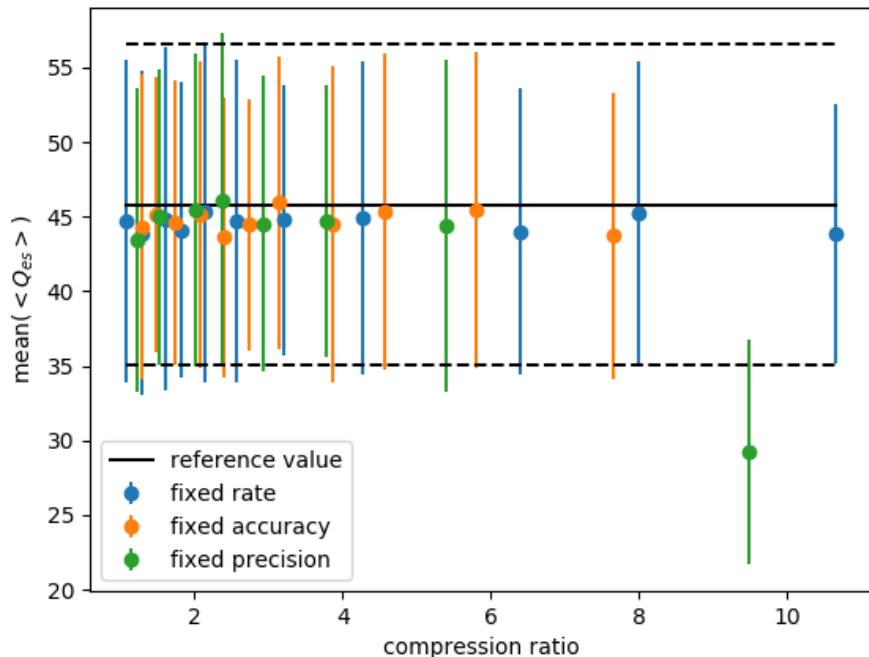
ZFP HDF5 plugin enables 250x compression in sw4 seismic code

[Work with David McCallen et al., LBL]



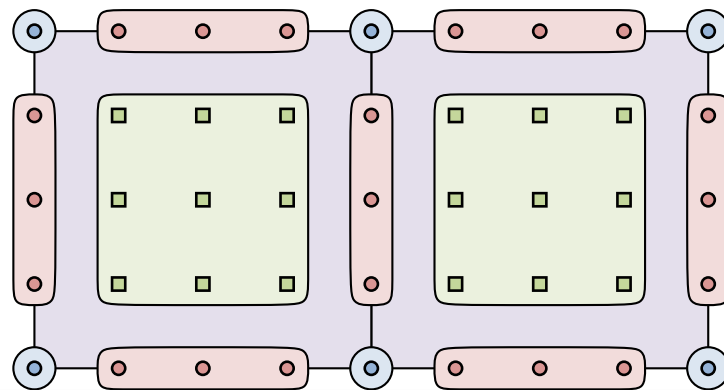
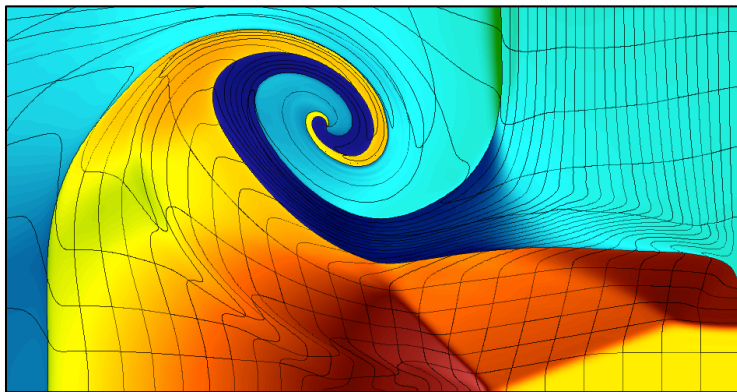
Inline compression: ZFP reduces 5D simulation state in GENE fusion code by 10x with acceptable loss in accuracy

[Work by Denis Jarema & Frank Jenko, MPI]



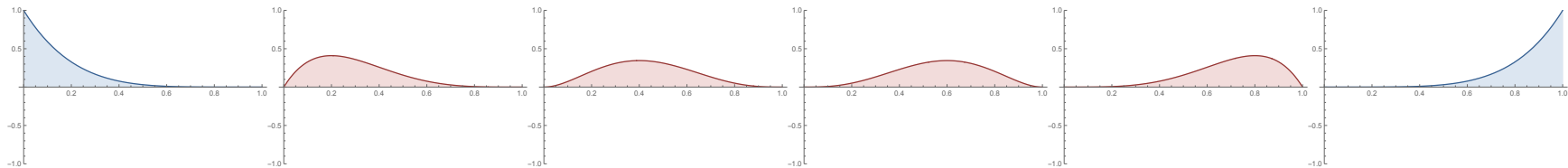
ZFPOLY generalizes ZFP to continuous polynomials bases for use in MFEM finite-element computations

- Interior DOFs of quad/hex elements can be compressed by ZFP as “tiny grids”
 - Finite-element polynomial order, p , dictates grid size—may require expensive padding
- ZFPOLY: ZFP generalization to continuous finite-element polynomials
 - L_2 space: Legendre polynomials (no continuity across elements)
 - H_1 space: integrated Legendre polynomials (C^0 continuity on element boundaries)
 - DOFs live on **vertices**, **edges**, and **elements**

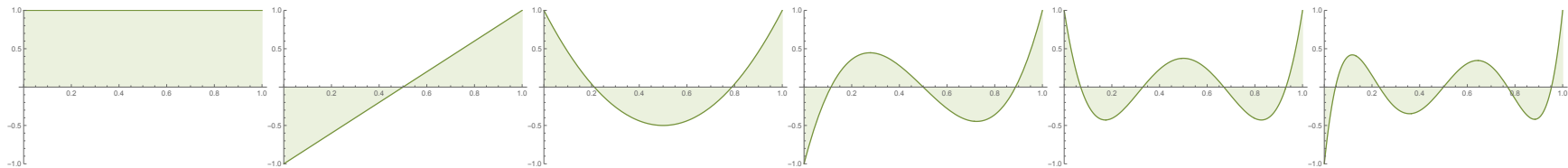


On H_1 finite element space, ZFPOLY uses integrated Legendre polynomials to avoid DOF duplication on element boundaries

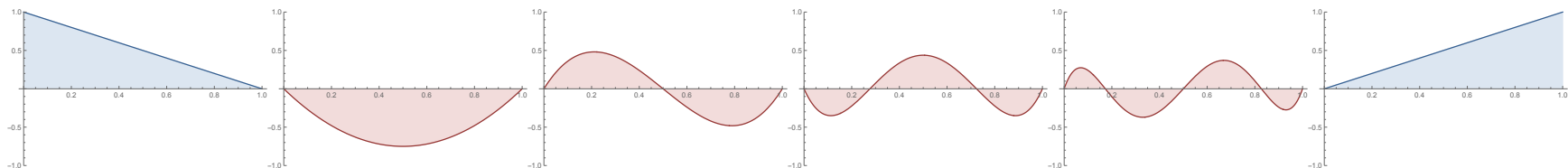
Bernstein



Legendre

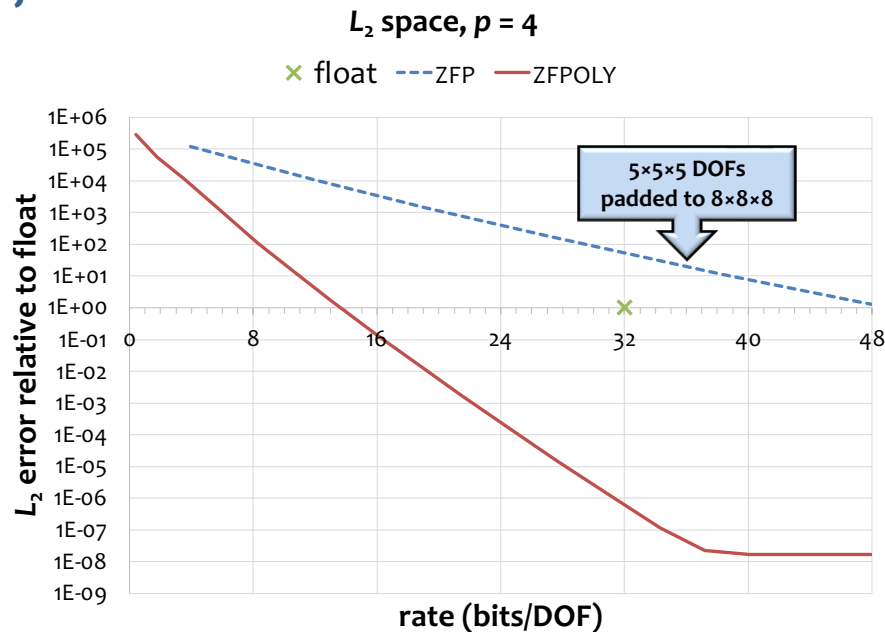
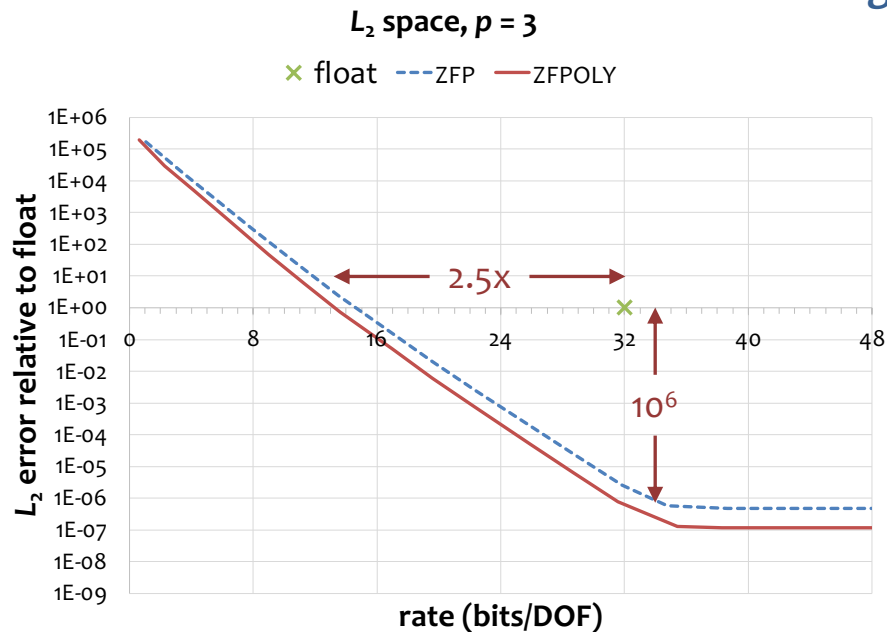


ZFPOLY



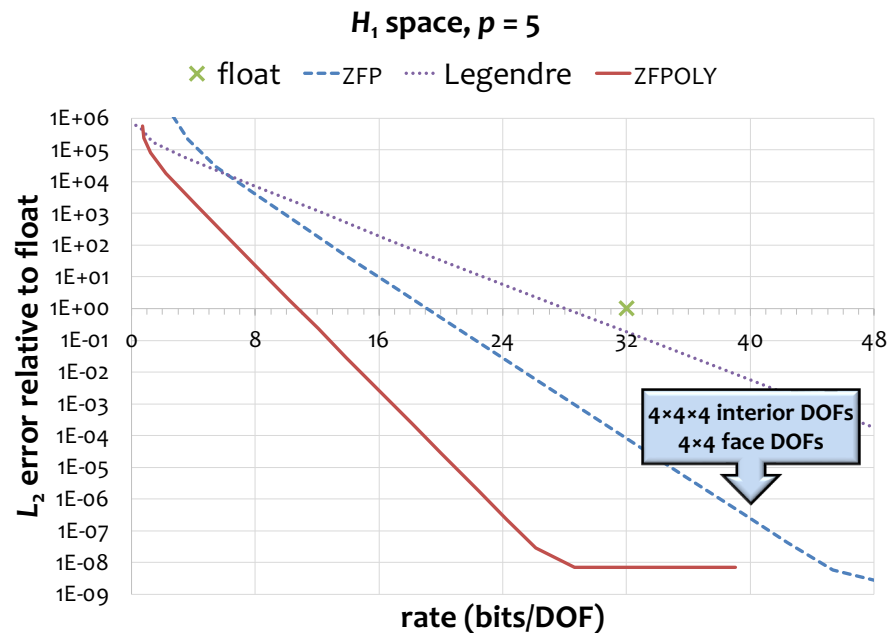
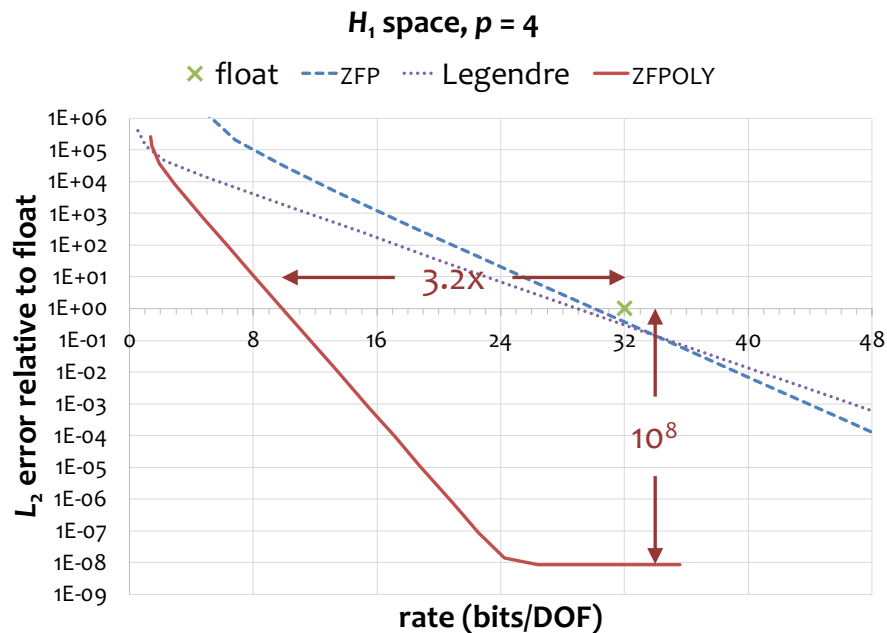
Compared to IEEE float, ZFPOLY reduces storage by 2.5x or accuracy by 6 orders of magnitude in 3D Laghos Sedov L_2 data

Energy field



Compared to IEEE float, ZFPOLY reduces storage by 3x or accuracy by 8 orders of magnitude in 3D Laghos Sedov H_1 data

Mesh $\times$ position

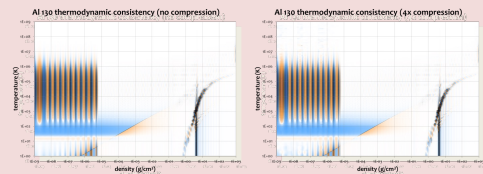


ZFP is being used in production in numerous petascale applications

With P. Sterne @ LLNL

Equation of State and Opacity Tables

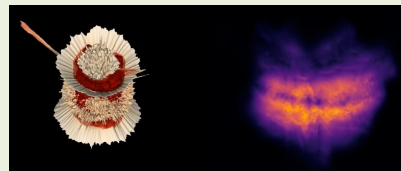
- Constant material tables consume a large fraction of memory
- ZFP arrays enable random access to compressed tables
- **4x compression** ensures thermodynamic consistency



With S. Langer @ LLNL

Simulated X-ray diagnostics

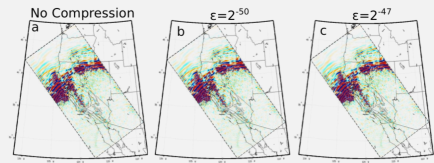
- Trinity runs generated **4 PB** of uncompressed data
- 15 months to transfer to LLNL; 2 months of disk space available
- **10x compression** allowed for successful data transfer



With P. Chen @ UW

Full-3D Seismic Tomography

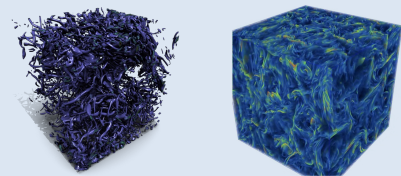
- Adjoint-wavefield competitor uses 200 GB
- Scattering-integral formulation uses **1.8 PB**
- **40x compression, 6x less I/O time**
- 10^{-6} compression error < 10^{-2} observation noise



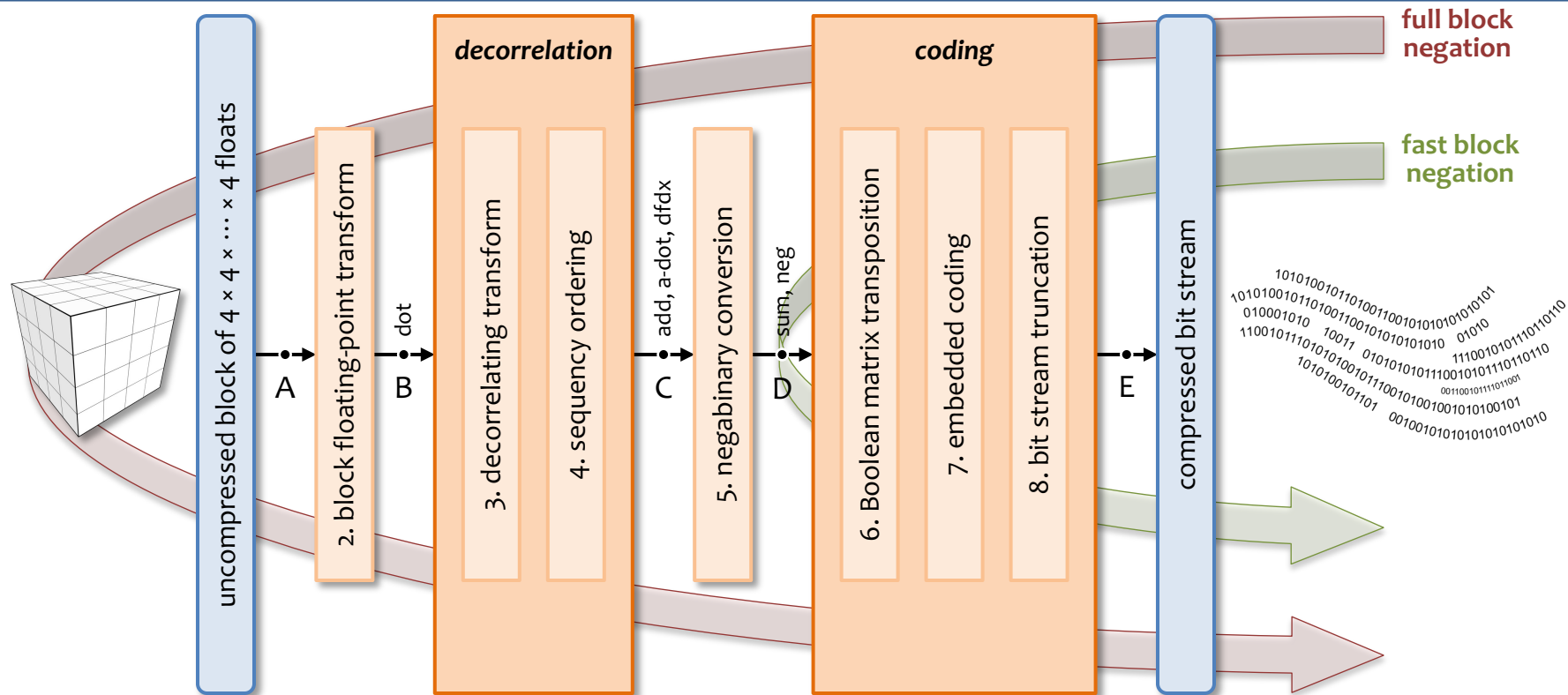
With R. Burns @ JHU

Johns Hopkins Turbulence Database

- Trinity runs use $8K^3$ grids = **2 TB/field/time step**
- Without compression, only selected features saved
- ZFP allows retaining full fields
- Compressed data server being considered

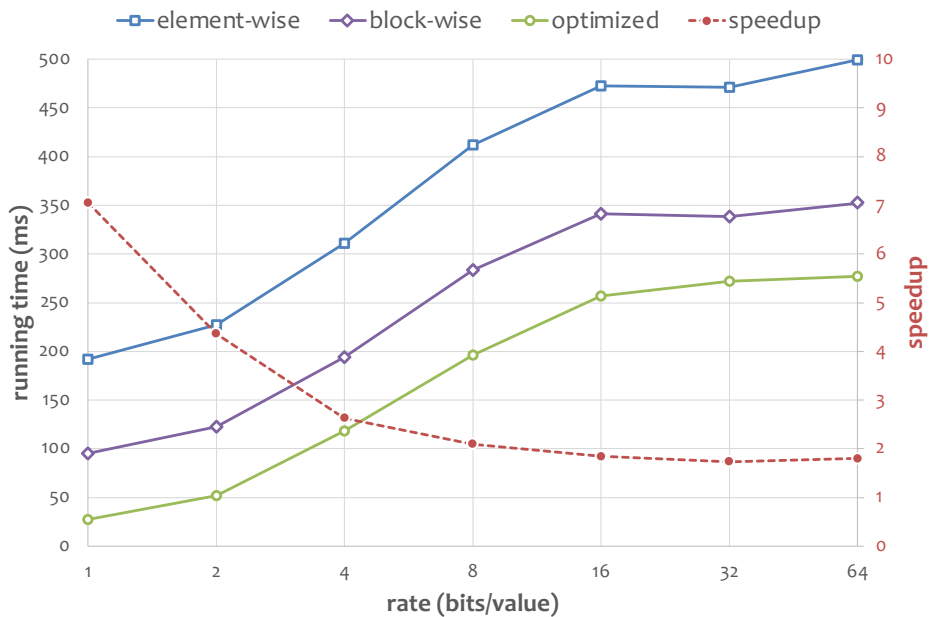


Many array operations can work on partially decompressed data without full decompress, compute, re-compress steps

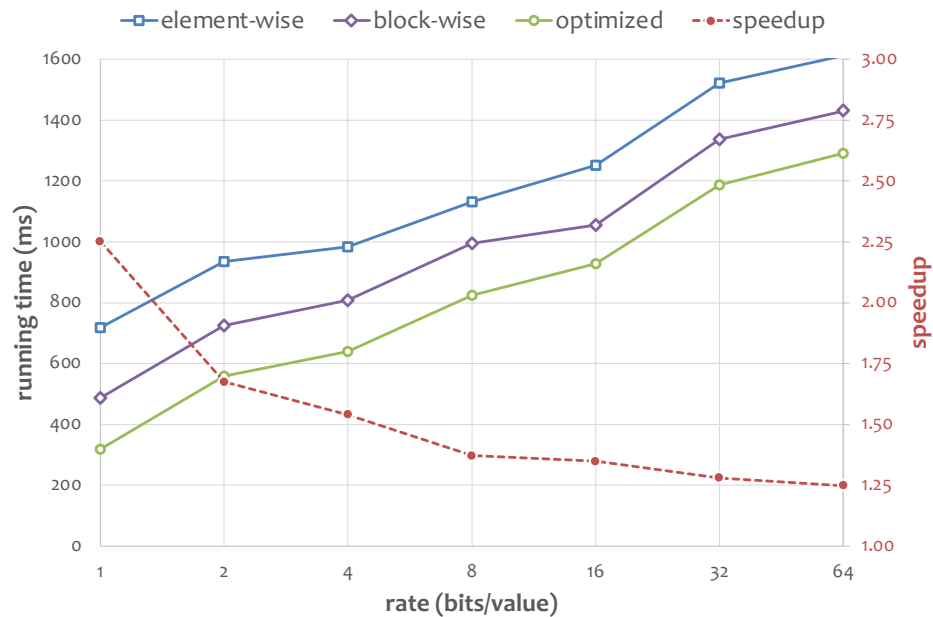


Optimized path results in speedups up to 7x over element-wise array computations with IEEE 754 arithmetic

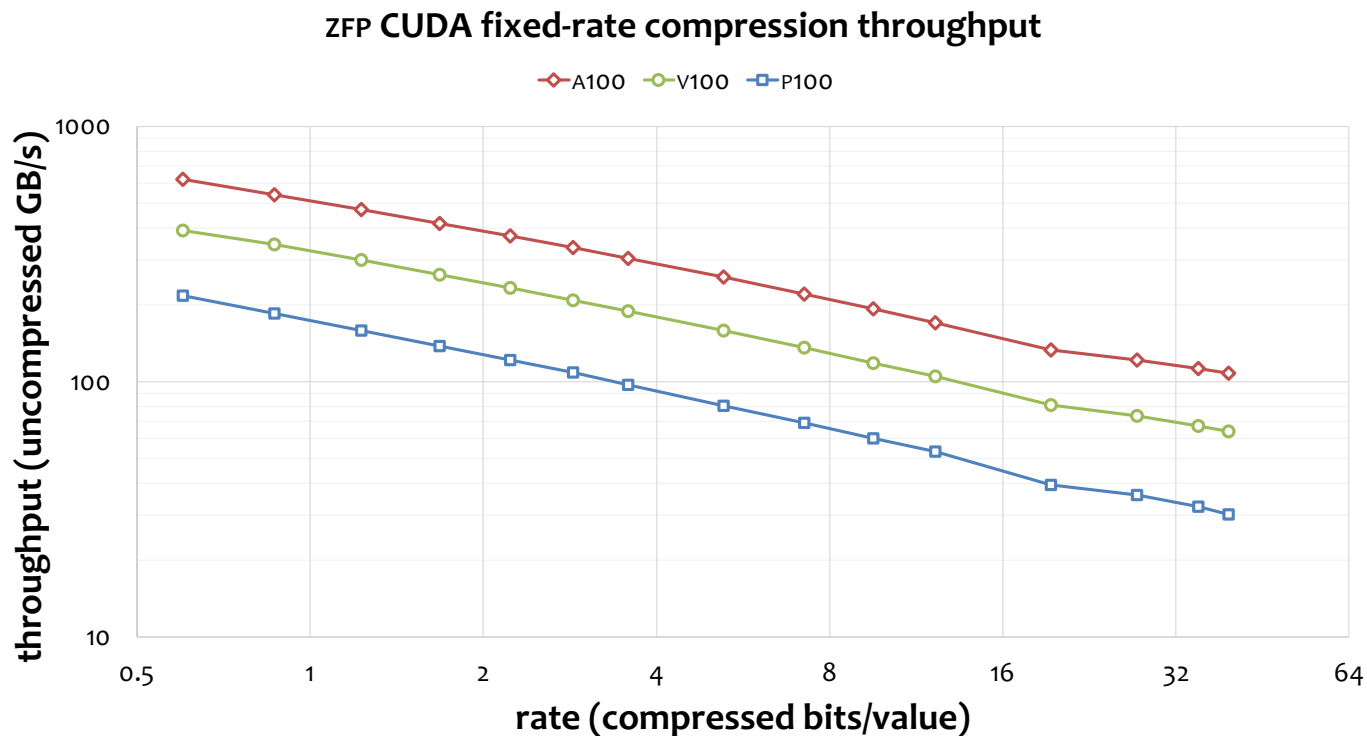
block sum



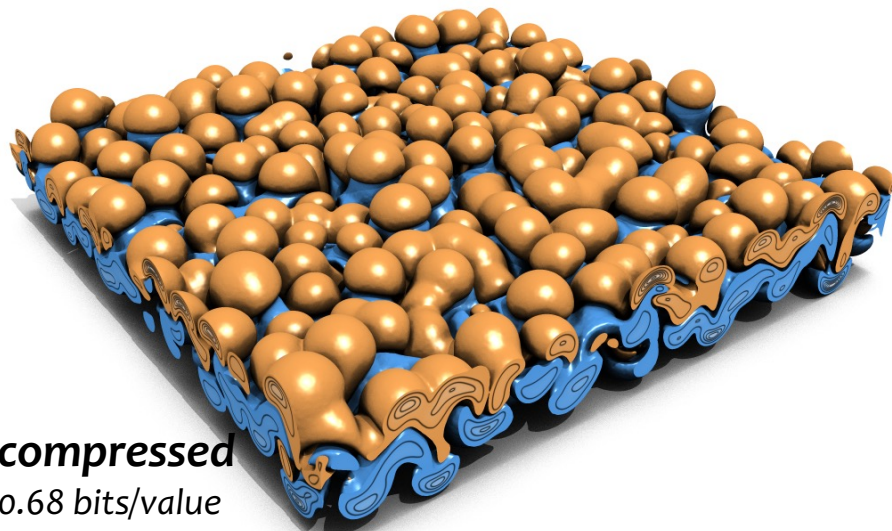
block negation



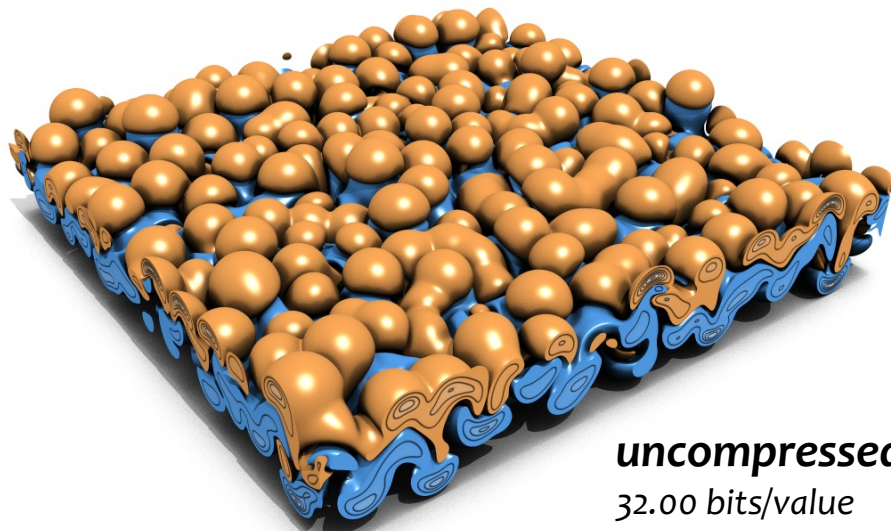
ZFP GPU compression achieves up to 700 GB/s throughput



ZFP's in-memory compressed arrays improve accuracy per bit stored and reduce time to solution

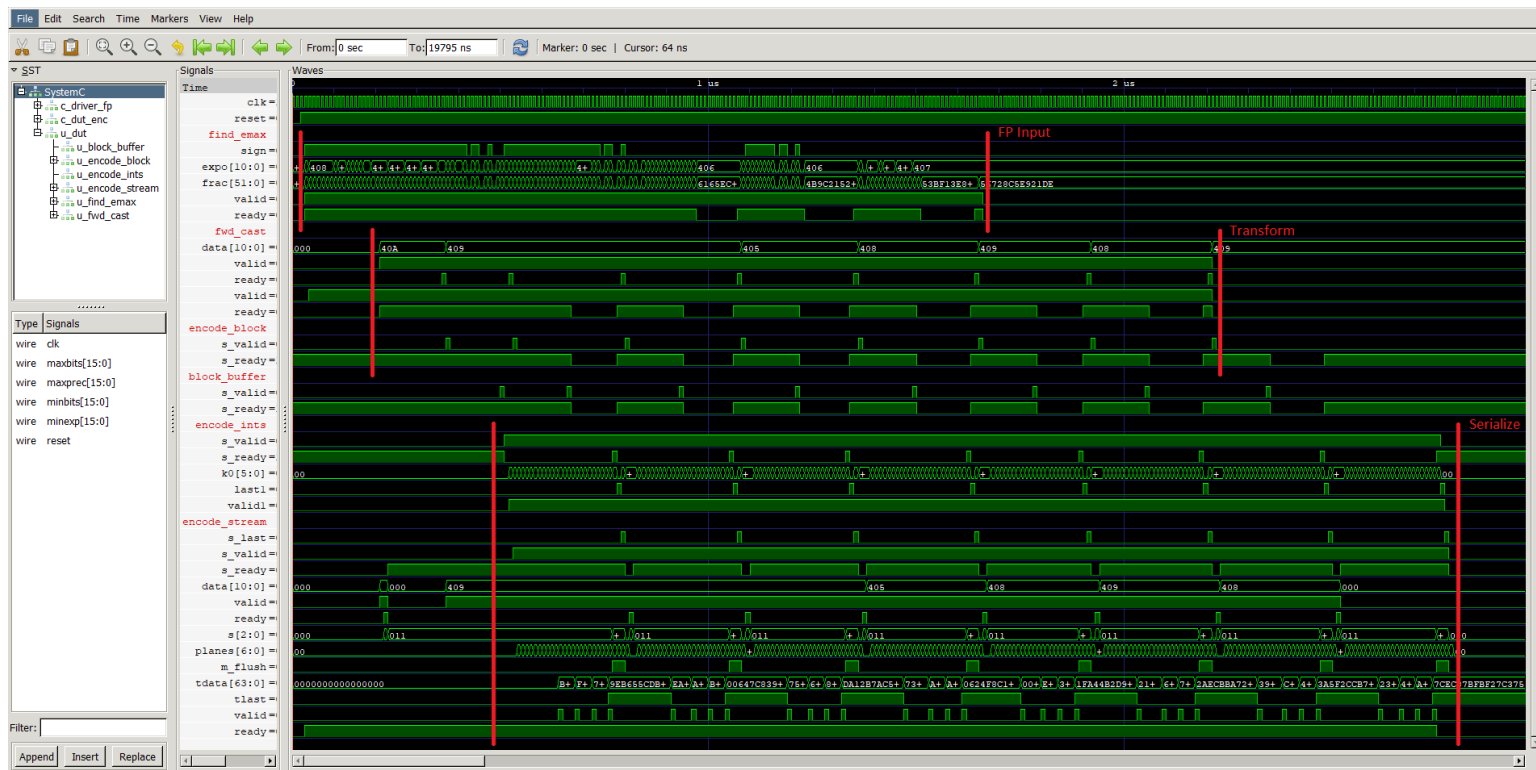


compressed
0.68 bits/value
2.29 hours

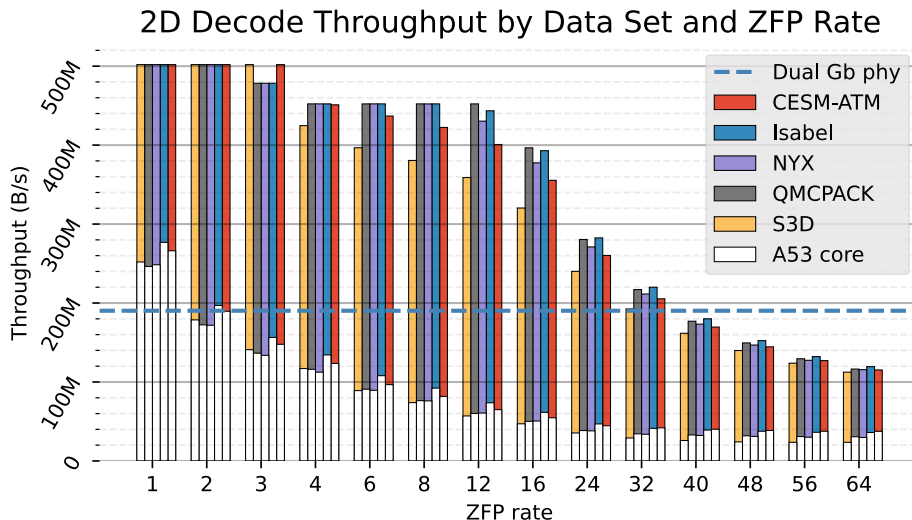
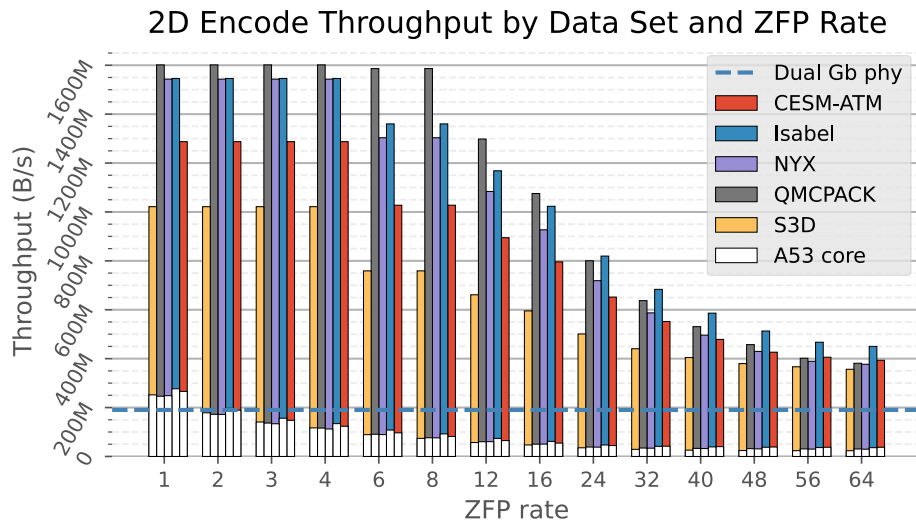


uncompressed
32.00 bits/value
2.44 hours

We have developed zHW, an FPGA implementation of zFP



Our ZHW FPGA encoder outperforms ZFP on a comparable CPU



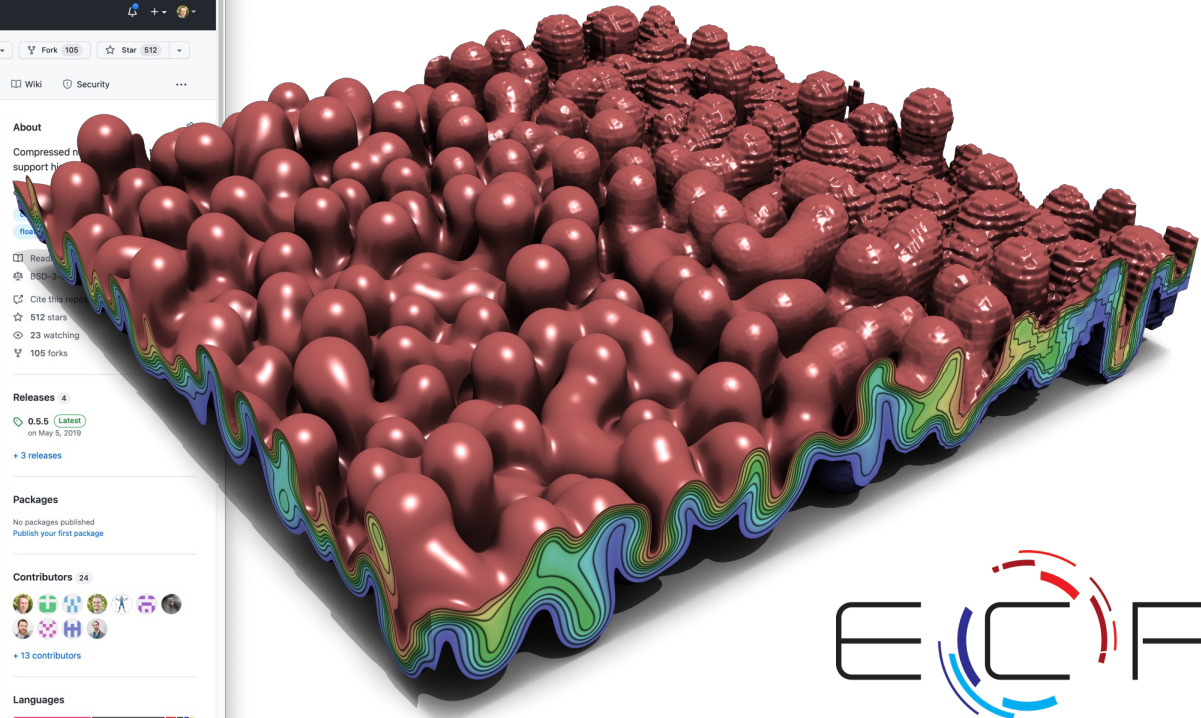
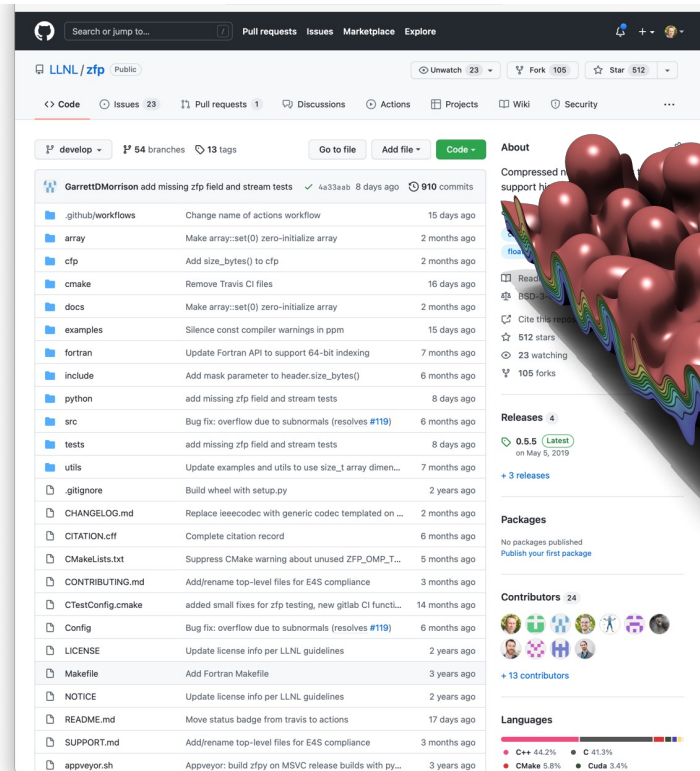
[Barrow et al., “ZHW: A numerical CODEC for big data scientific computation,” FPT 2022]

Conclusions: ZFP makes every bit count

- ZFP compression enables substantial savings in storage
 - **4–10x reduction** in numerical state (e.g., PDEs)
 - **10–250x reduction** in data analysis and visualization
- ZFP provides continuous knob that allows trading accuracy and storage
 - **No half/float/double dichotomy** as in mixed-precision computations
 - **1,000,000x less error** using similar storage
- What about performance?
 - (De)compression **introduces latency**
 - In general, compression **improves** end-to-end **CPU-GPU transfer, communication, I/O** speed
 - In certain cases, compression **improves on-node performance** (we're doing this in software!)
 - Ultimately, we want **compression on a chip** (cf. h/w compression on cell phones, GPUs)
- What about data movement?
 - More work needed to quantify actual data movement + performance models

ZFP is available as open source on GitHub:

<https://github.com/LLNL/zfp>





Acknowledgments

This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research.

Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.