

ZFP: Compressed Floating-Point Arrays for Exascale Computing

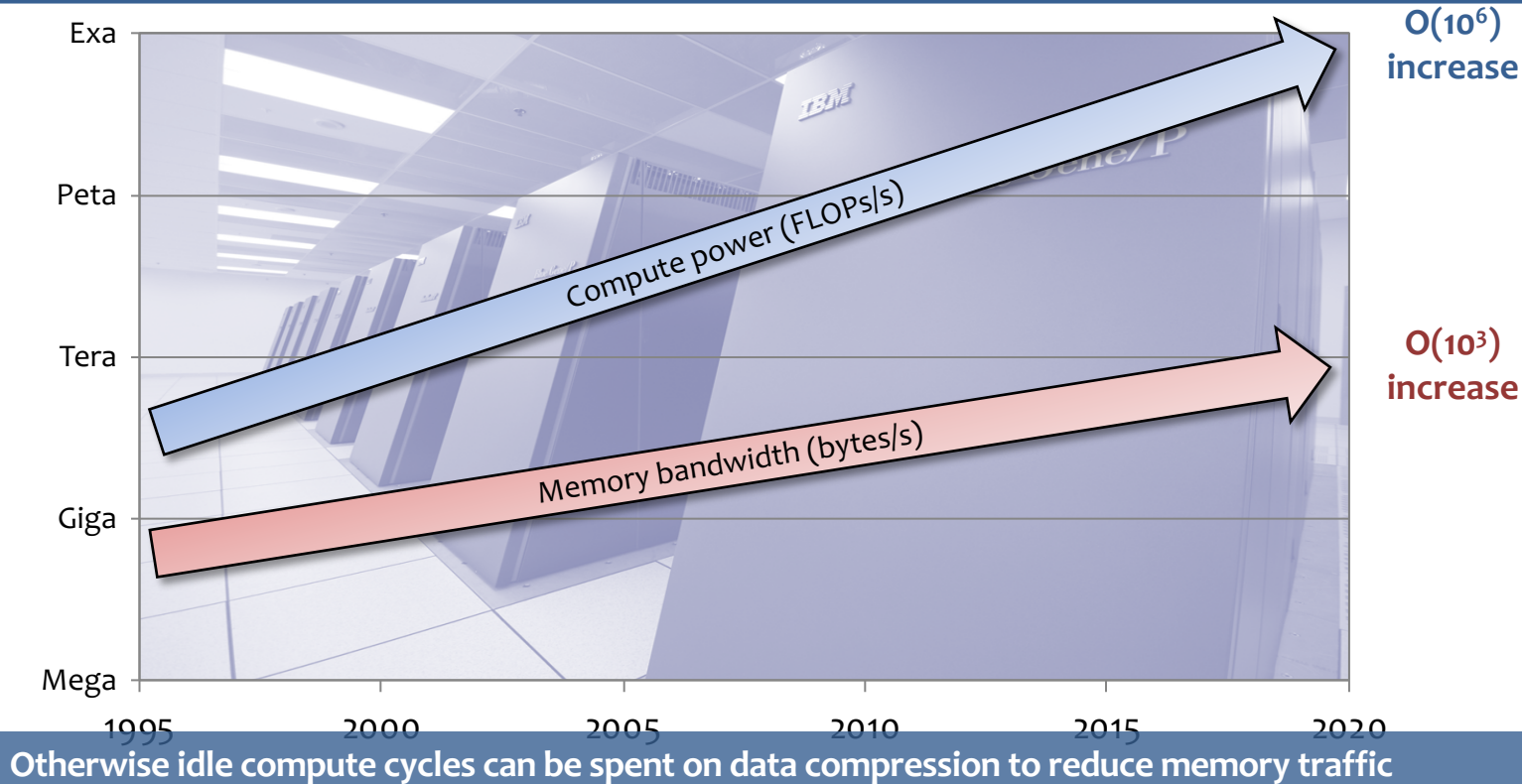
ISC High Performance 2019

Peter Lindstrom

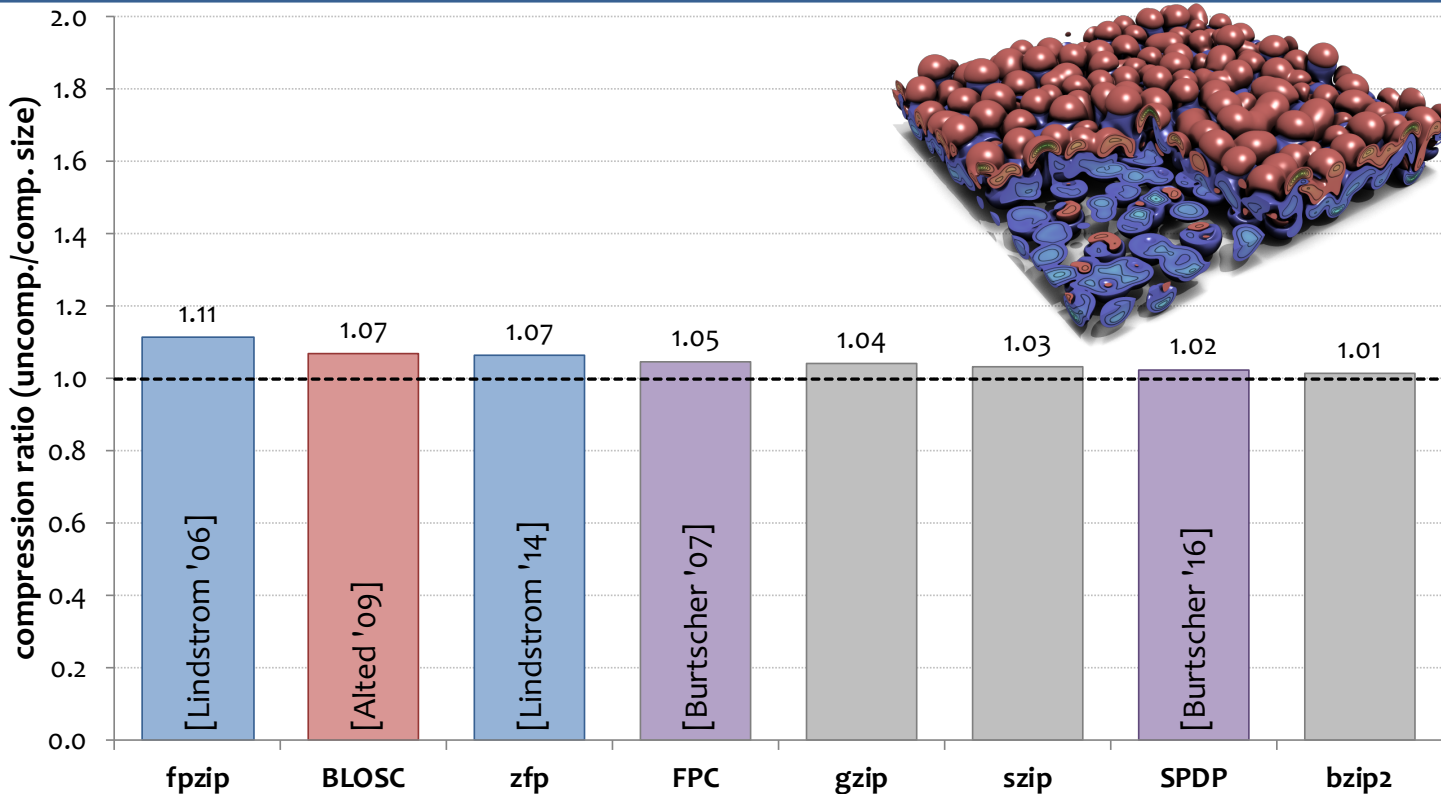
June 19, 2019



Data movement will dictate performance and power usage at exascale

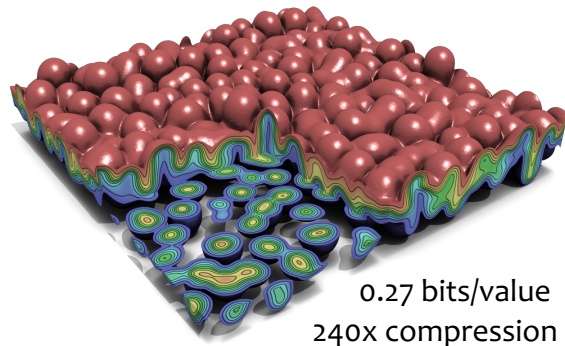


(64-bit) floating-point data does not compress well losslessly

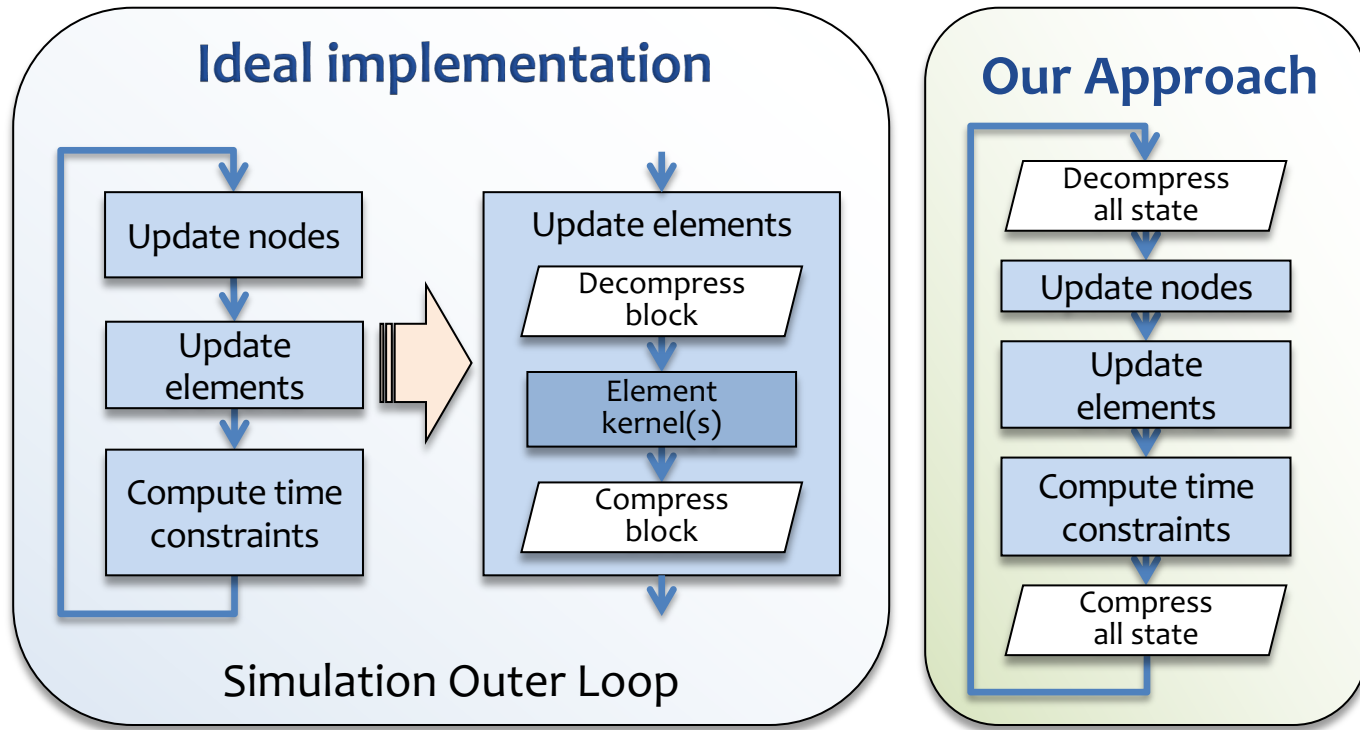


Lossy compression enables greater reduction, but is often met with skepticism by scientists

- Large improvements in compression are possible by allowing even small errors
 - Simulation often computes on meaningless bits
 - Round-off, truncation, iteration, model **errors abound**
 - Last few floating-point bits are effectively **random noise**
- Still, lossy compression often makes scientists nervous
 - Even though lossy **data reduction** is 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 history files)
 - State-of-the-art compressors support **error tolerances**



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

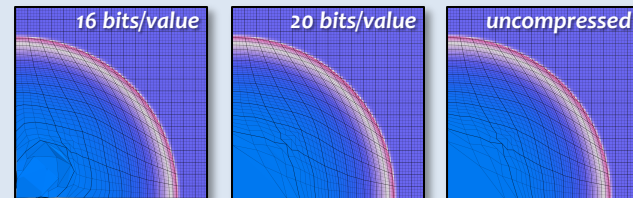


[Laney et al., "Assessing the effects of data compression in simulations using physically motivated metrics," SC 2013]

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

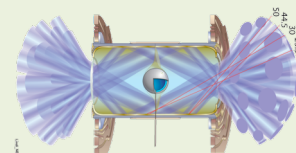
Lagrangian shock hydrodynamics

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



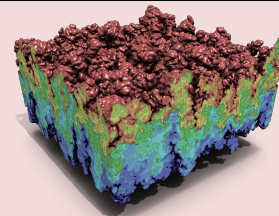
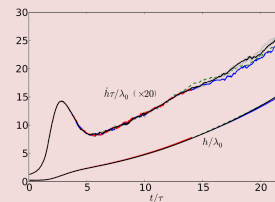
Laser-plasma multi-physics

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



High-order Eulerian hydrodynamics

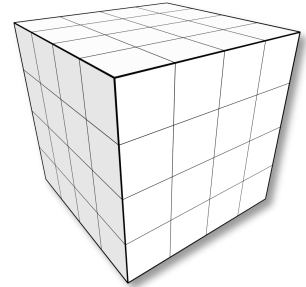
- QoI: *Rayleigh-Taylor mixing layer thickness*
- 10,000 time steps
- At **4x compression**, relative **error < 0.2%**



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

ZFP is an inline compressor for floating-point arrays

- ZFP provides a C++ d -dimensional compressed array primitive
 - Based on decomposition into independent **blocks of 4^d values**
 - $O(1)$ read & write **random access** with user-defined memory footprint
 - **Replaces IEEE** as number format for numerical computations
 - Very fast: up to **150 GB/s** parallel throughput
 - H/w friendly: uses only integer additions and bitwise operations
 - **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)`
- ZFP can also be used for streaming compression to reduce I/O and storage
 - Supports absolute and (local) relative **error tolerances**
 - Supports **spatially adaptive & progressive** compression
 - **Resilient** to data corruption



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];
}
```

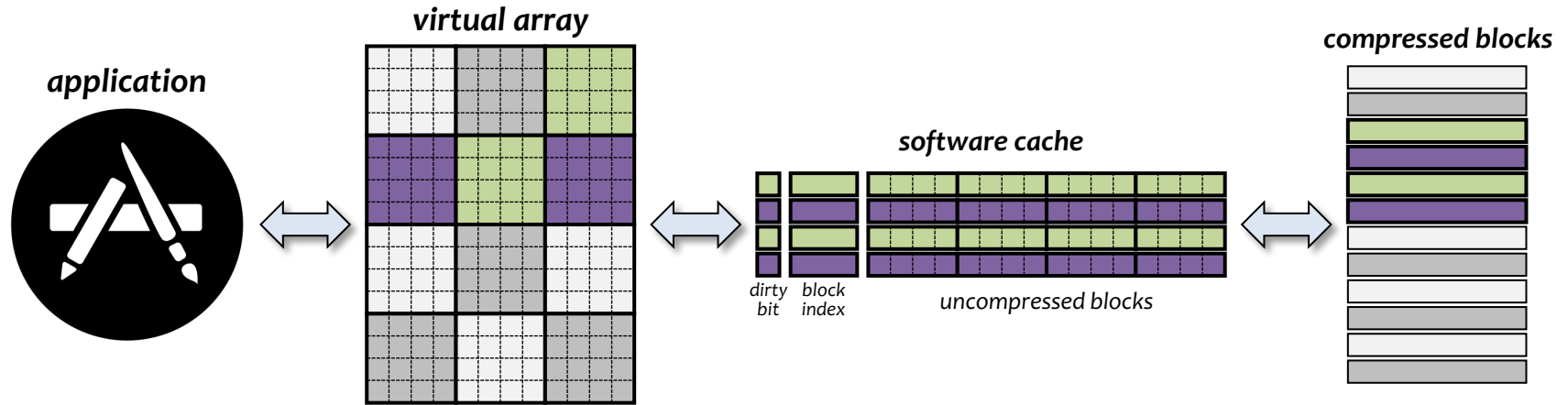
// example using ZFP arrays

```
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];
}
```

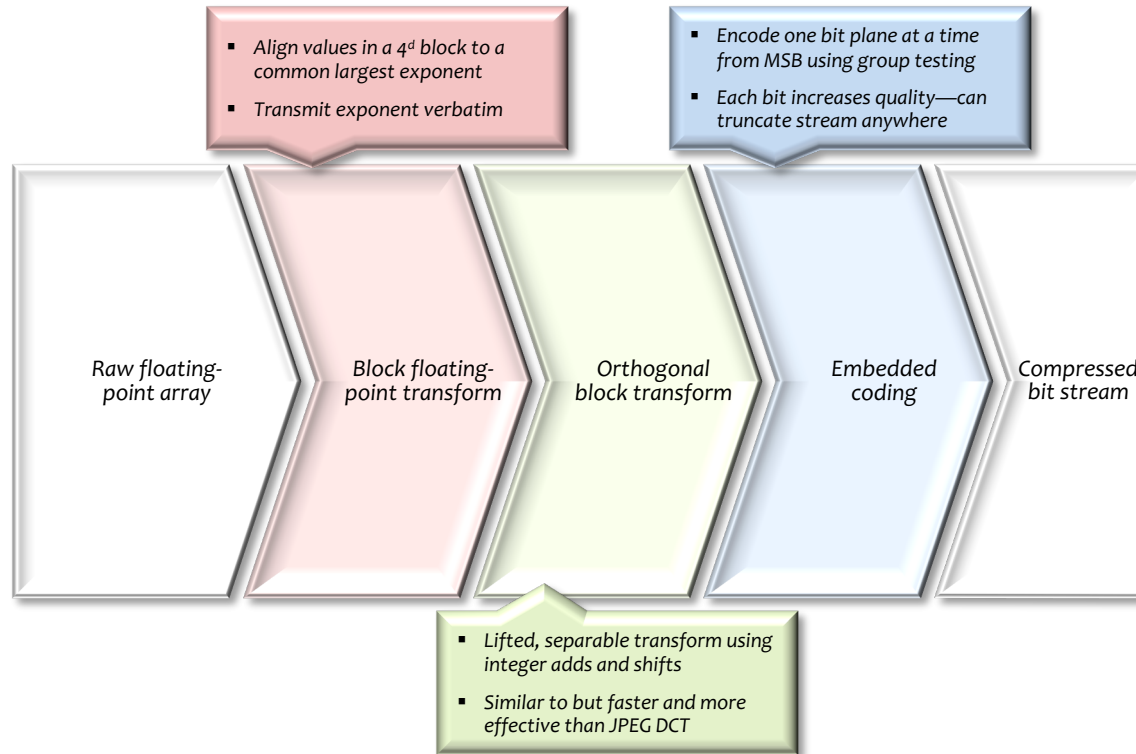
■ required changes

■ optional changes for improved readability

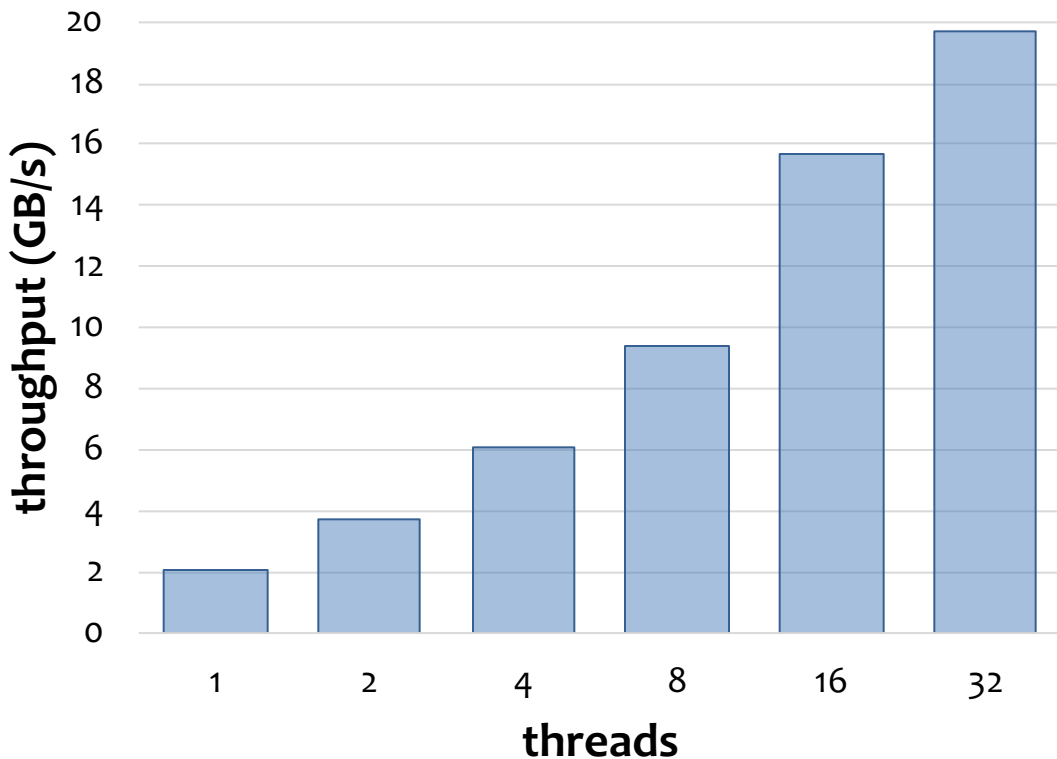
ZFP arrays limit data loss via a small write-back cache



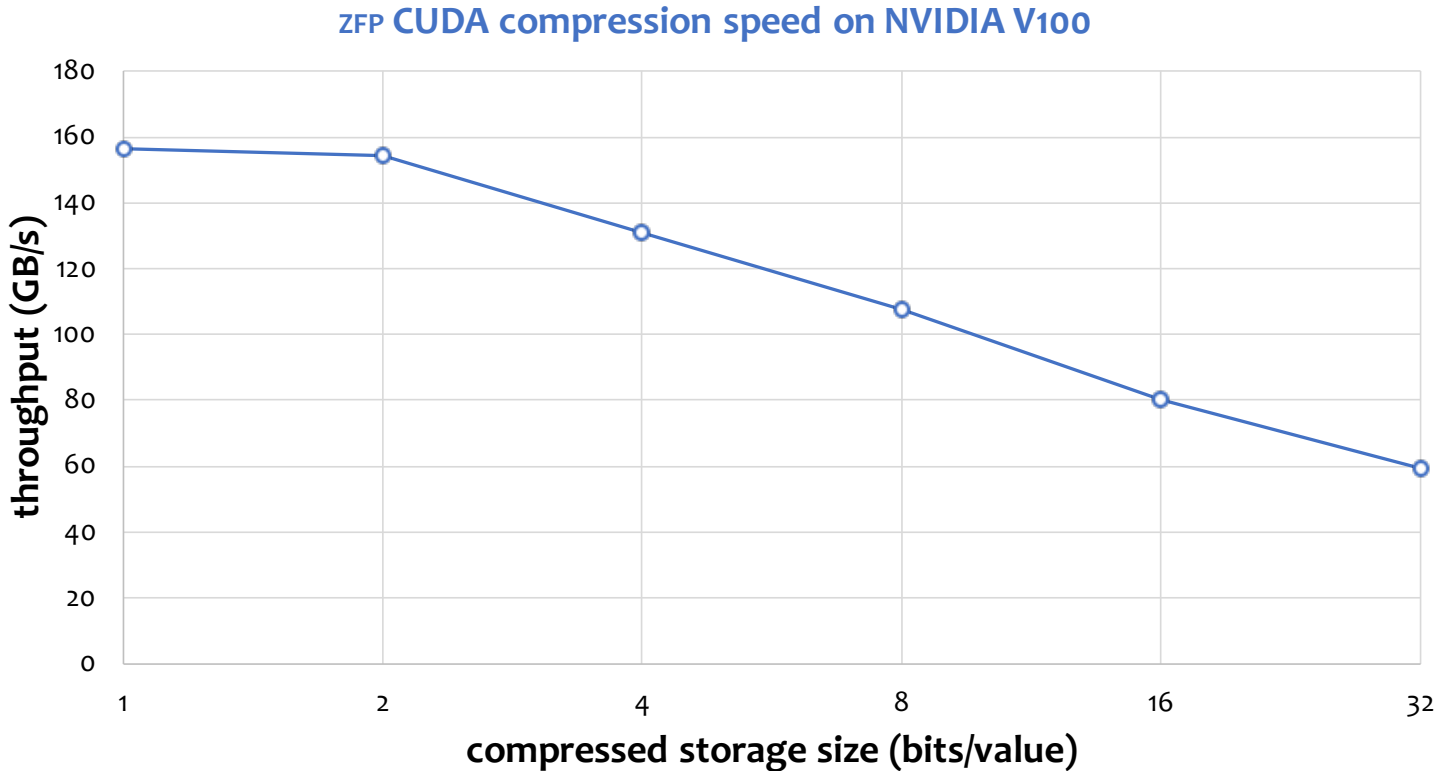
The ZFP compressor is comprised of three distinct components



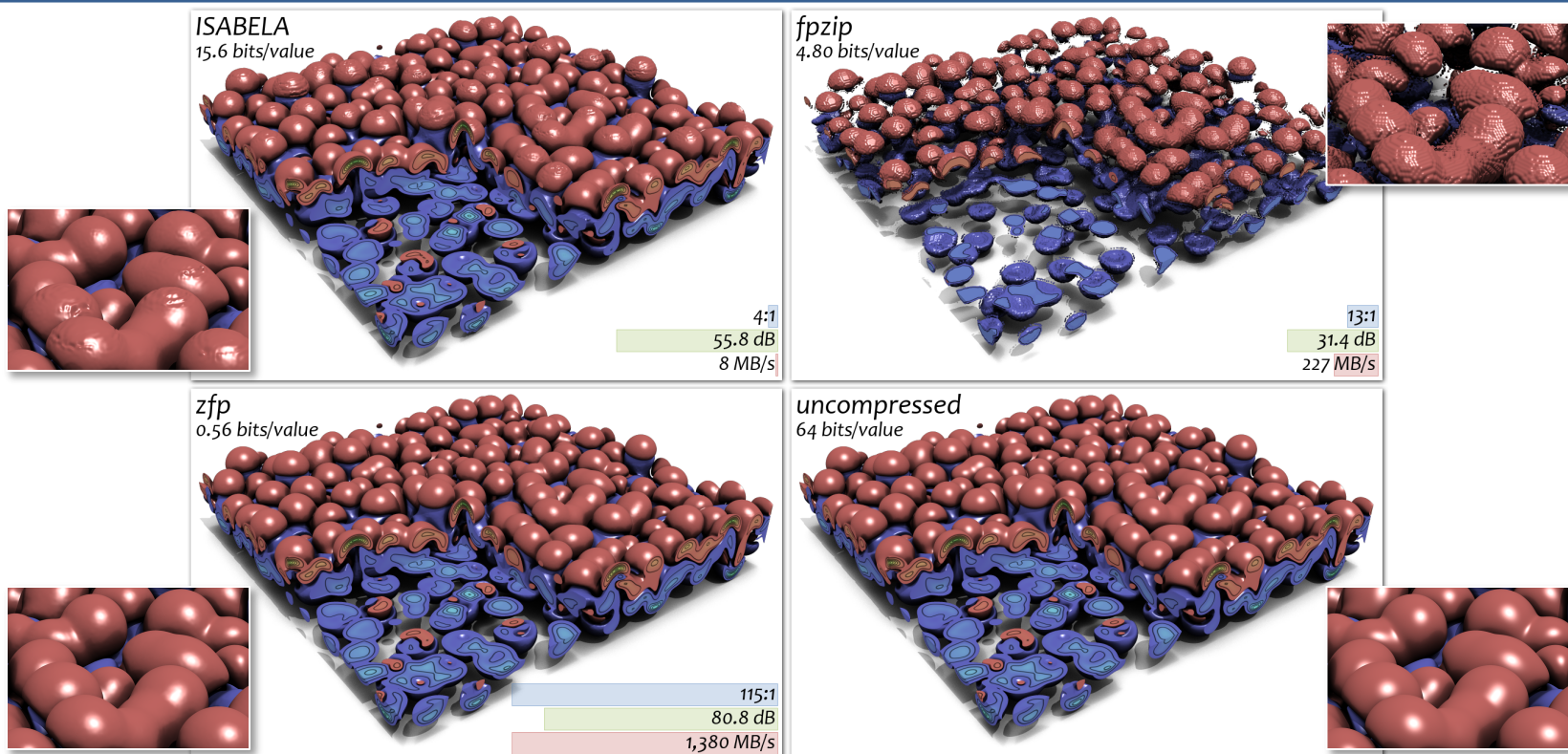
ZFP OpenMP compression achieves up to 20 GB/s throughput



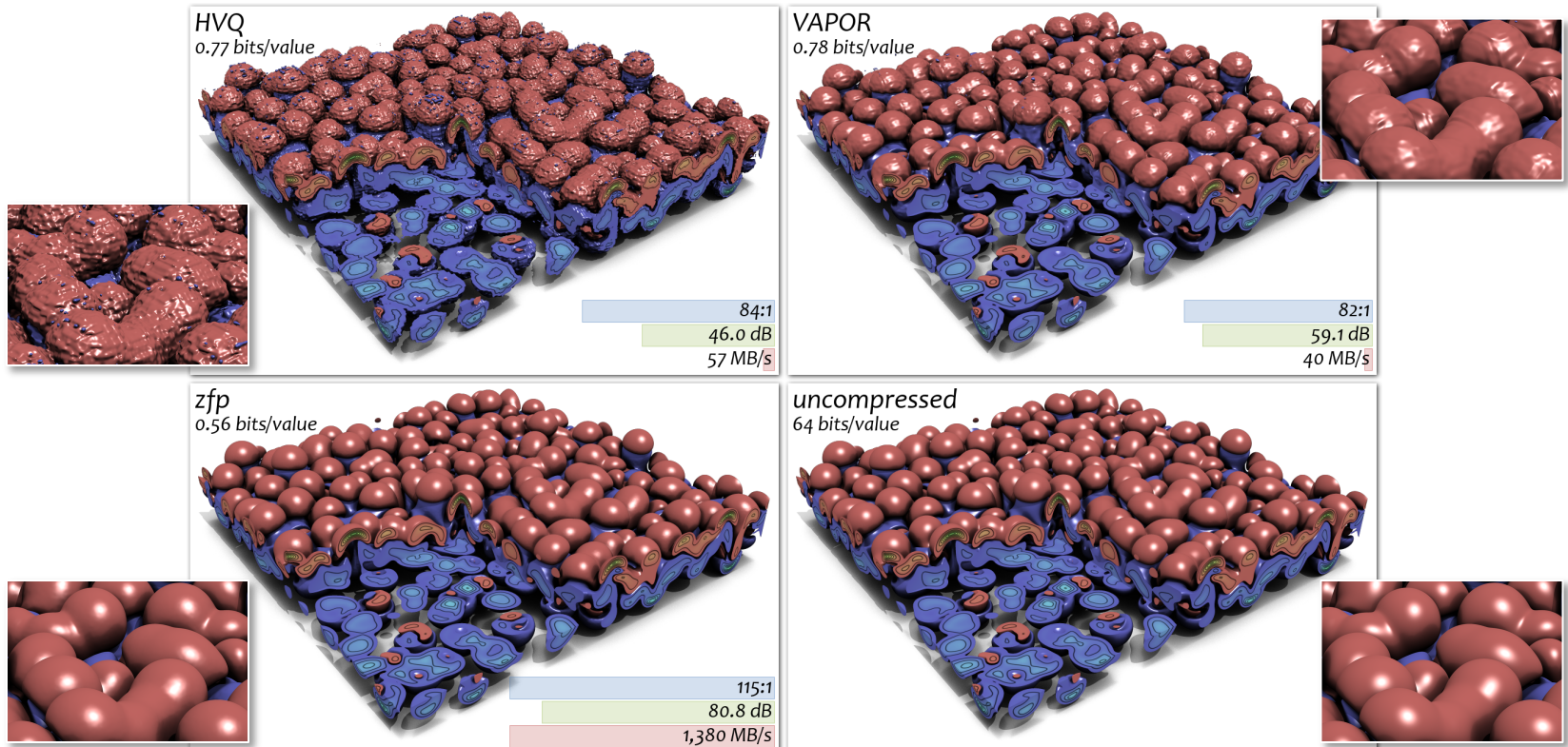
ZFP CUDA compression achieves up to 150 GB/s throughput



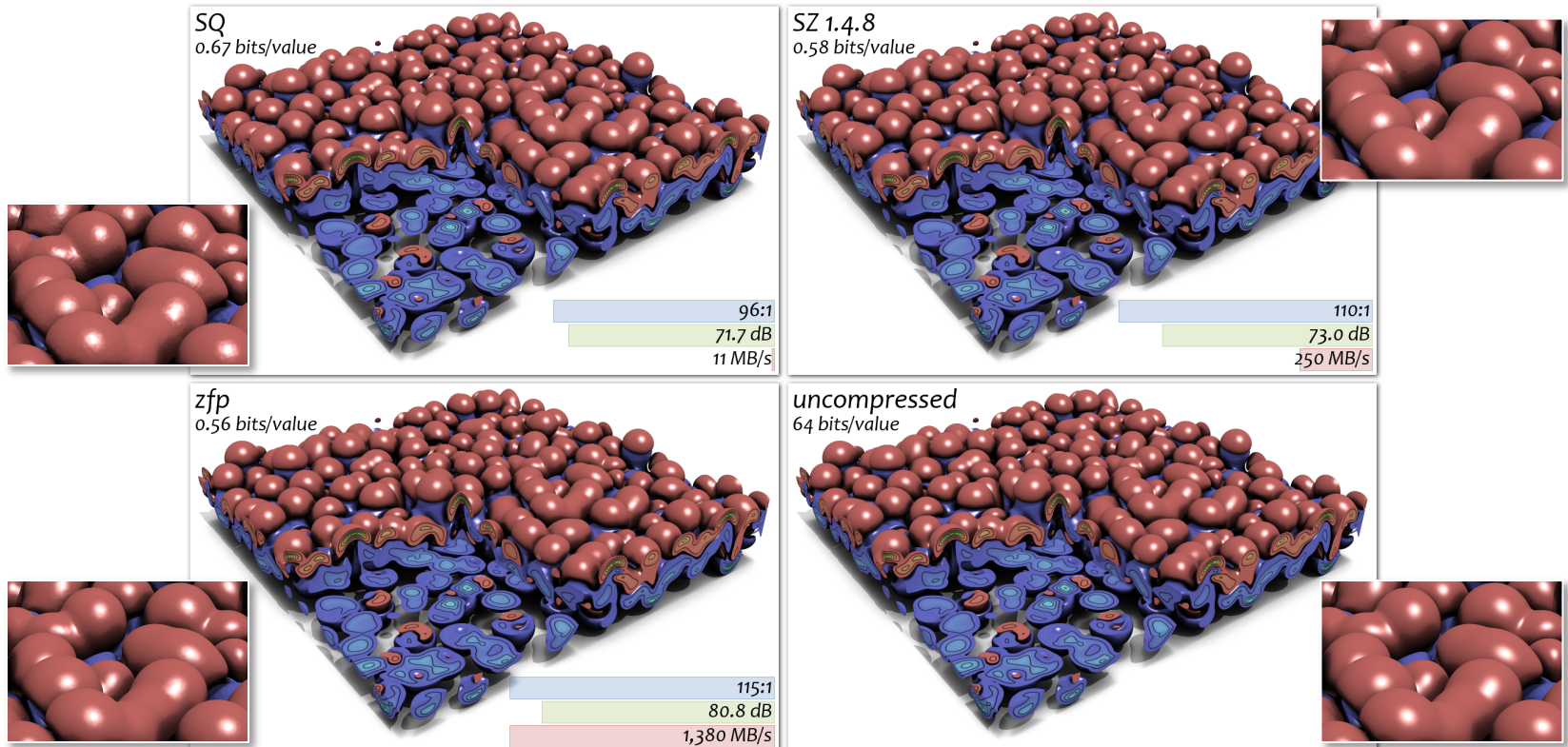
ZFP provides >100x compression with imperceptible loss in visual quality



ZFP provides >100x compression with imperceptible loss in visual quality



ZFP provides >100x compression with imperceptible loss in visual quality



Got artifacts?



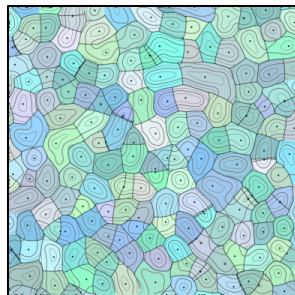
Velocity field Runge-Kutta integration shows good agreement with uncompressed field



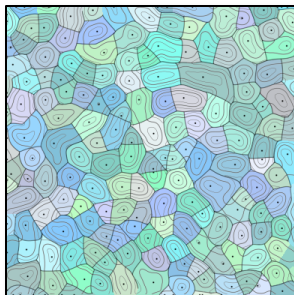
Morse segmentation at 16x compression shows lack of blocking artifacts



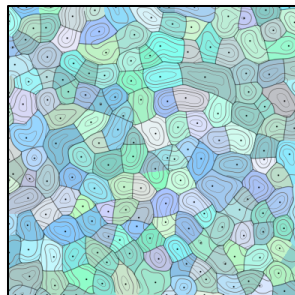
1 bit/value



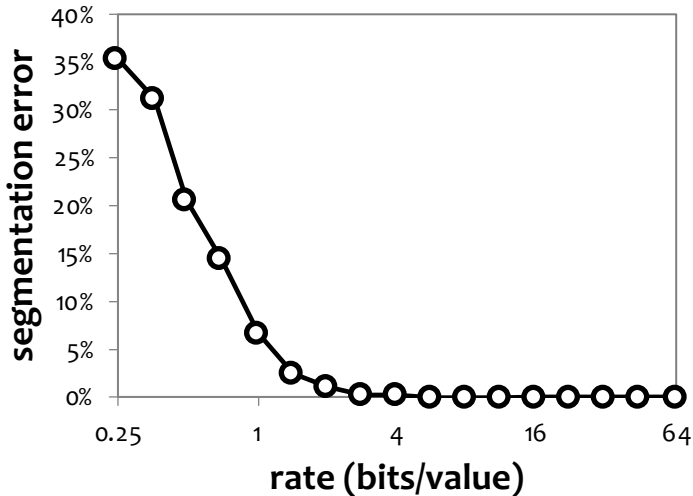
2 bits/value



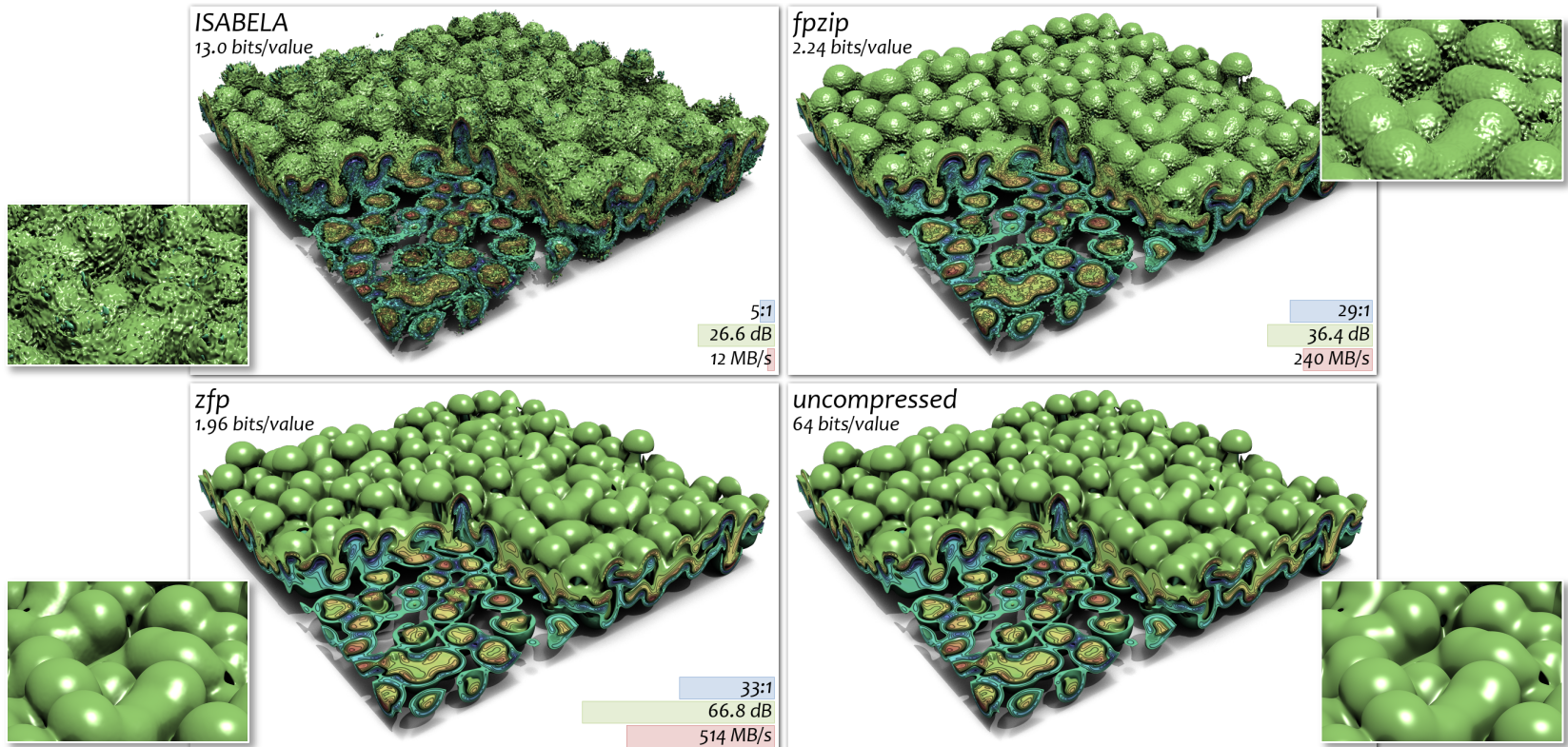
4 bits/value



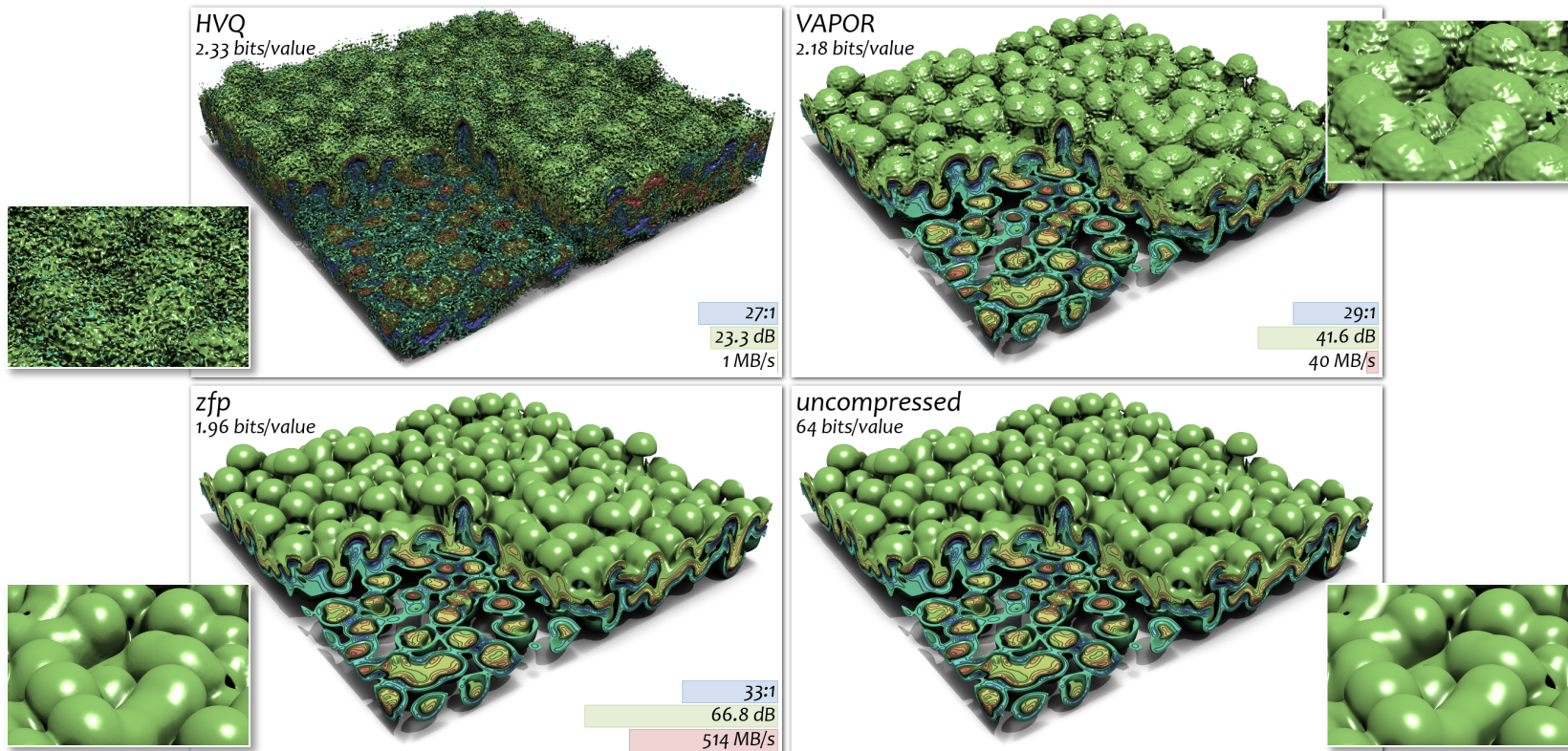
64 bits/value



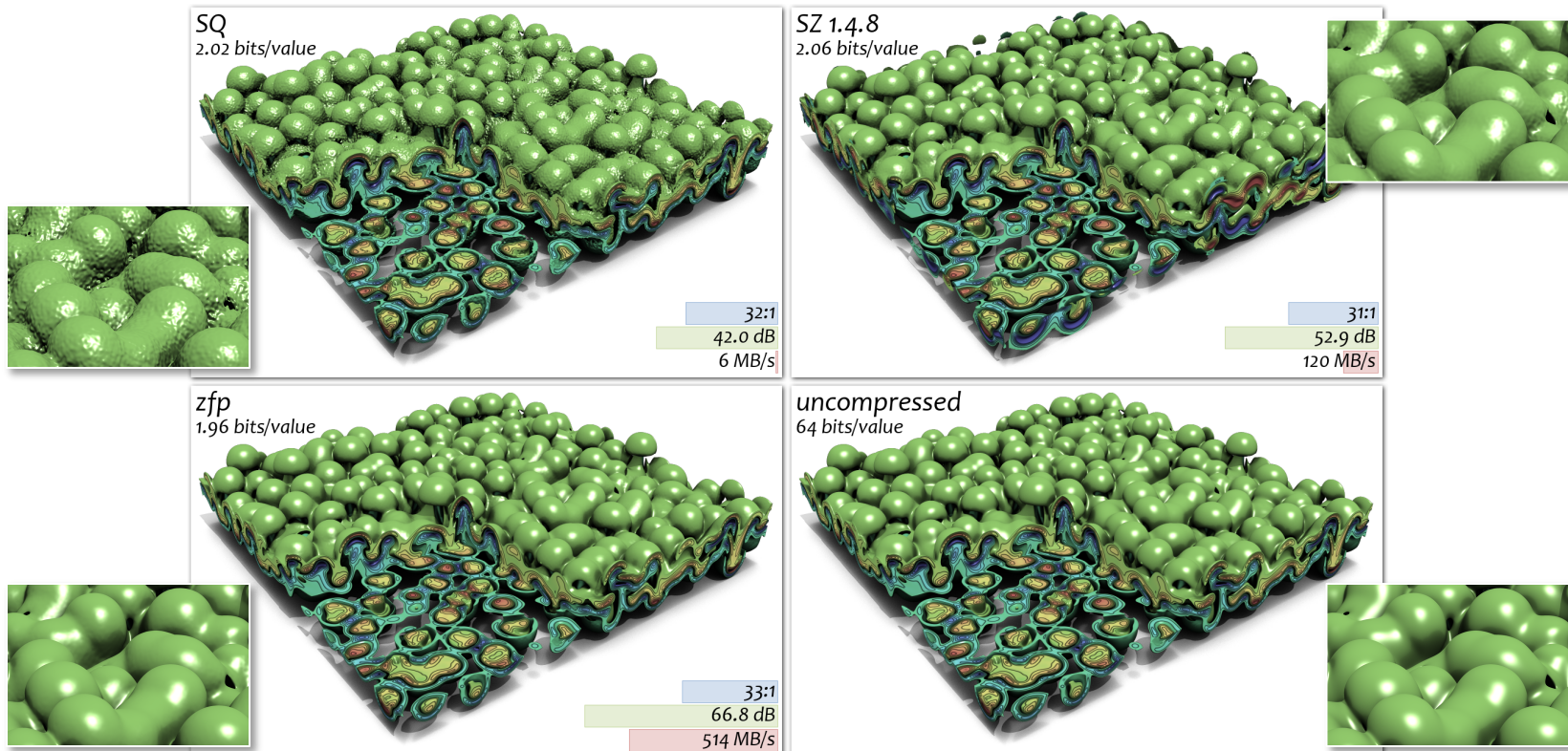
ZFP shows no artifacts in derivative computations (velocity divergence)



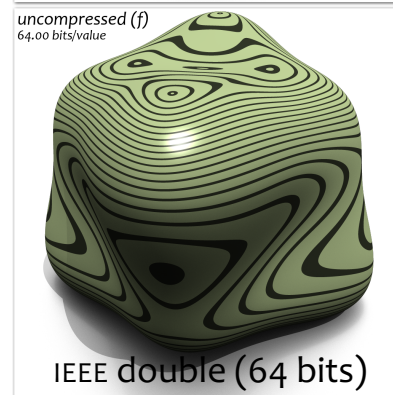
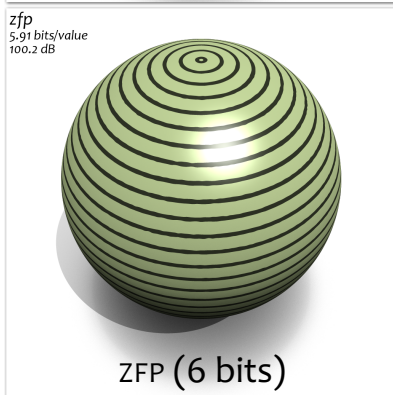
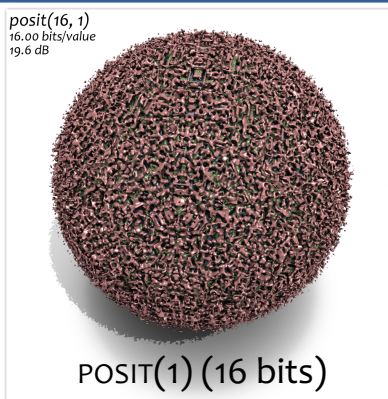
ZFP shows no artifacts in derivative computations (velocity divergence)



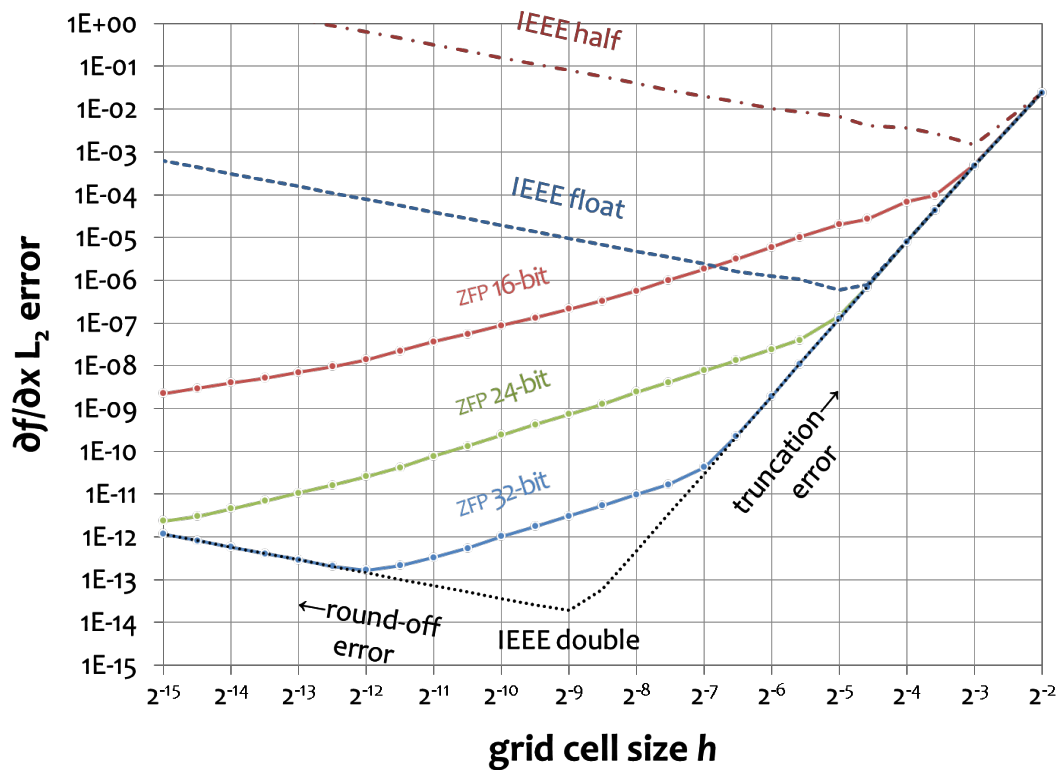
ZFP shows no artifacts in derivative computations (velocity divergence)



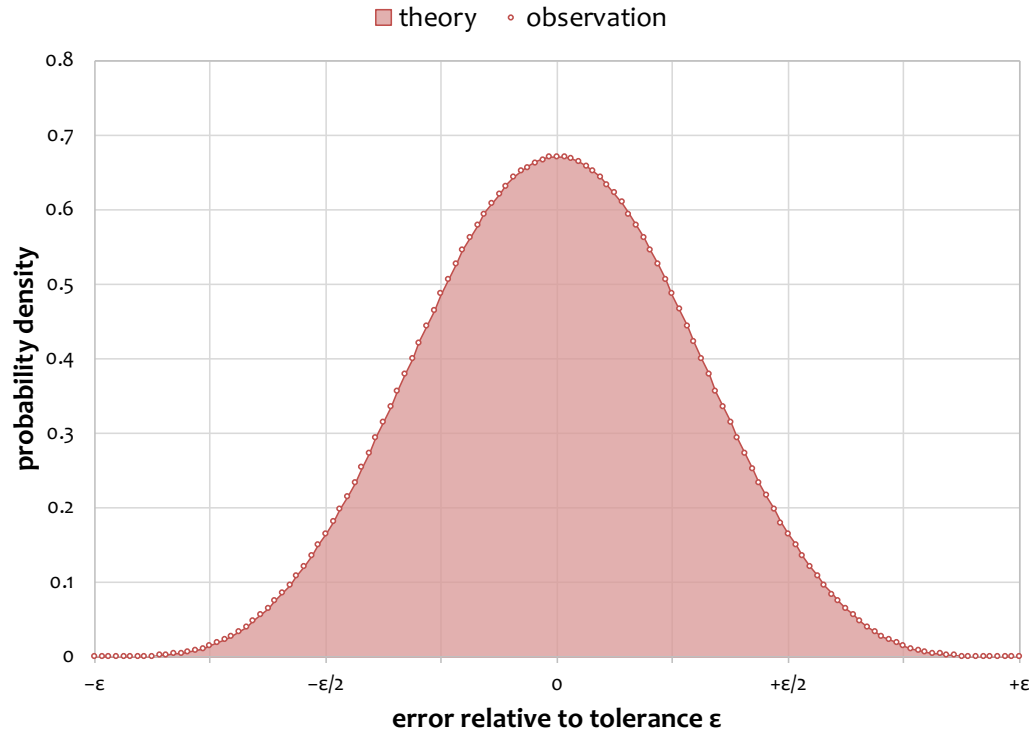
6-bit ZFP gives one more digit of accuracy than 32-bit IEEE in 2nd, 3rd derivative computations (Laplacian and highlight lines)



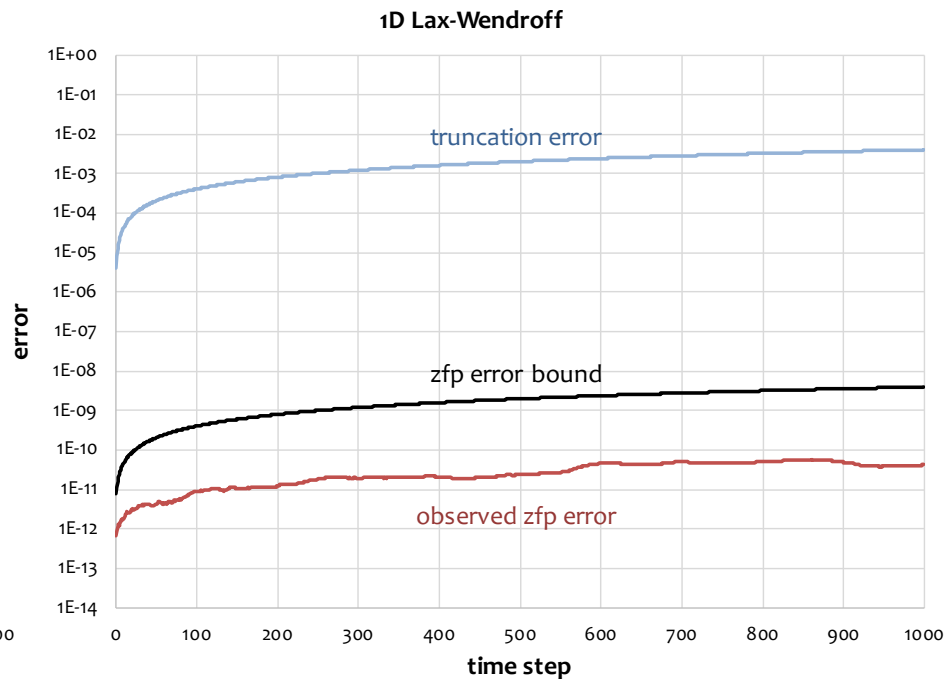
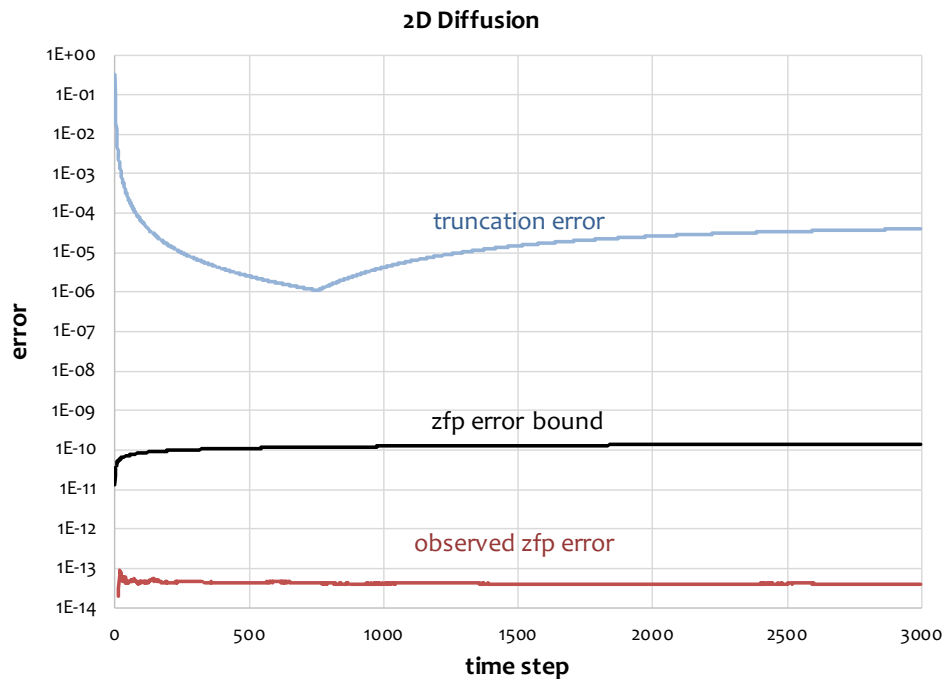
ZFP improves accuracy in finite difference computations using less precision than IEEE



ZFP compression errors are well behaved and follow a normal distribution



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



[Diffenderfer et al., “Error Analysis of ZFP Compression for Floating-Point Data,” SIAM Journal on Scientific Computing, 2019]

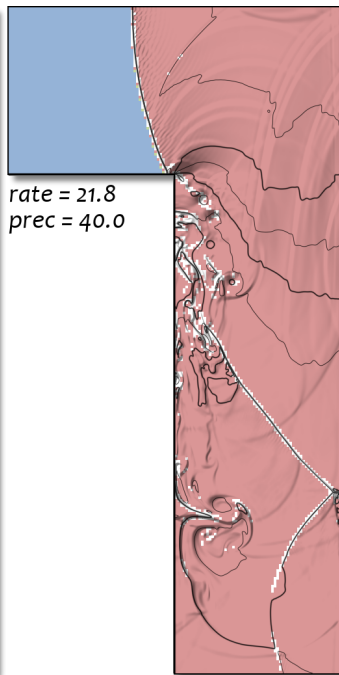
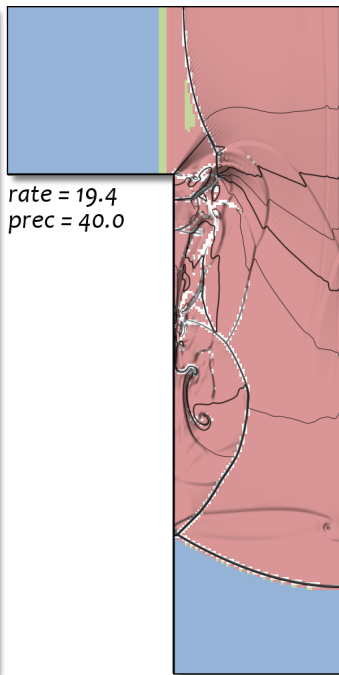
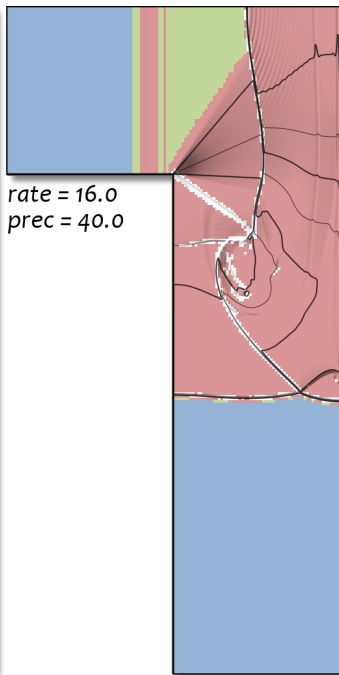
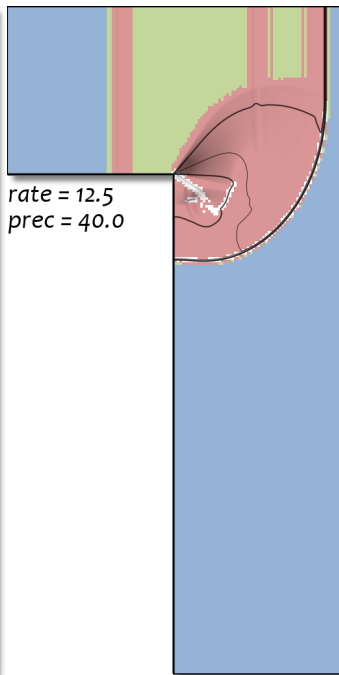
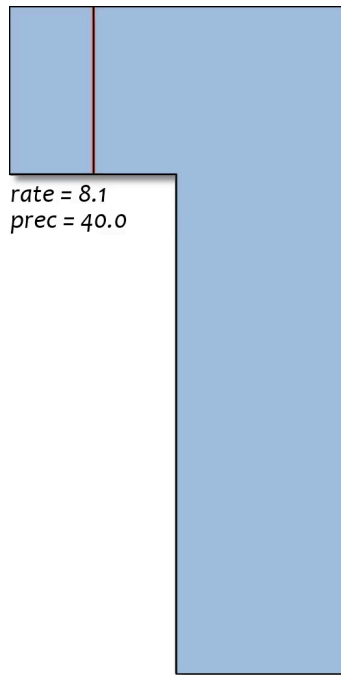
ZFP variable-rate arrays adapt storage spatially and allocate bits to regions where they are most needed

8 bits/value

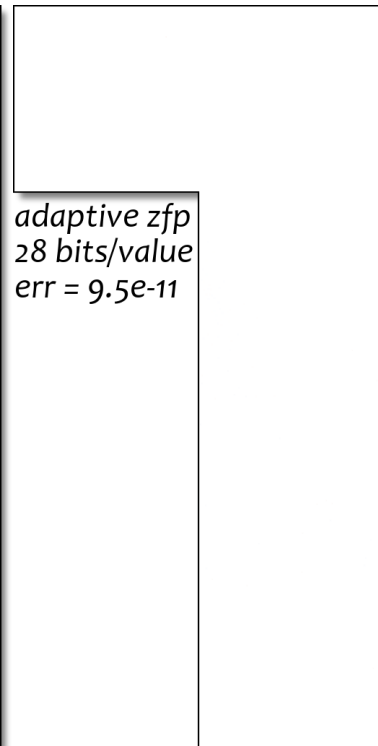
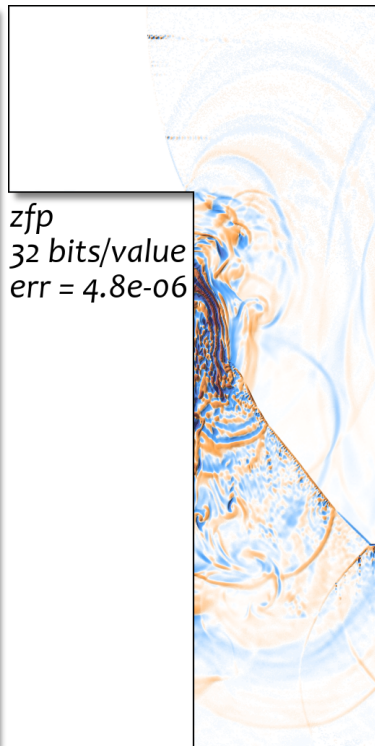
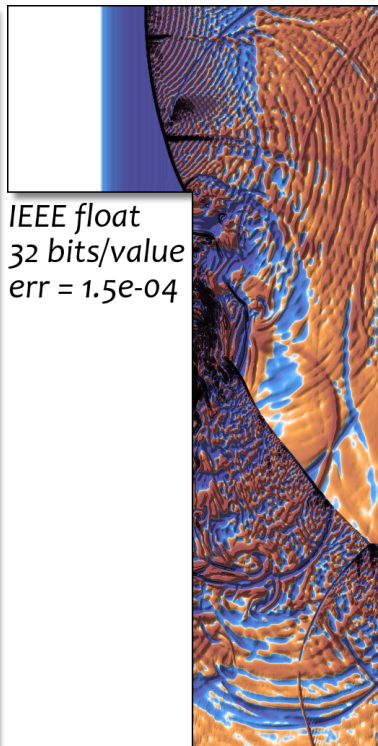
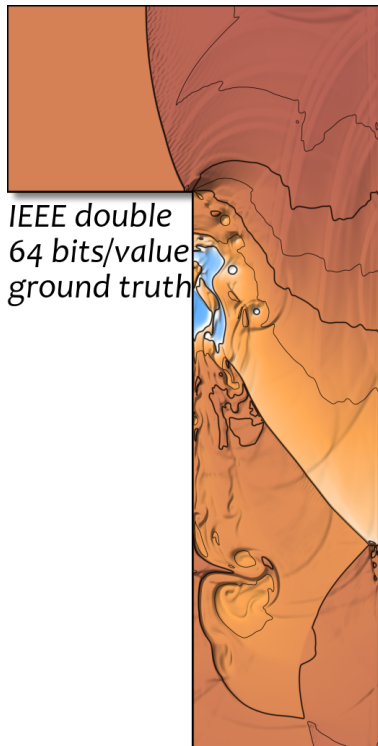
16 bits/value

32 bits/value

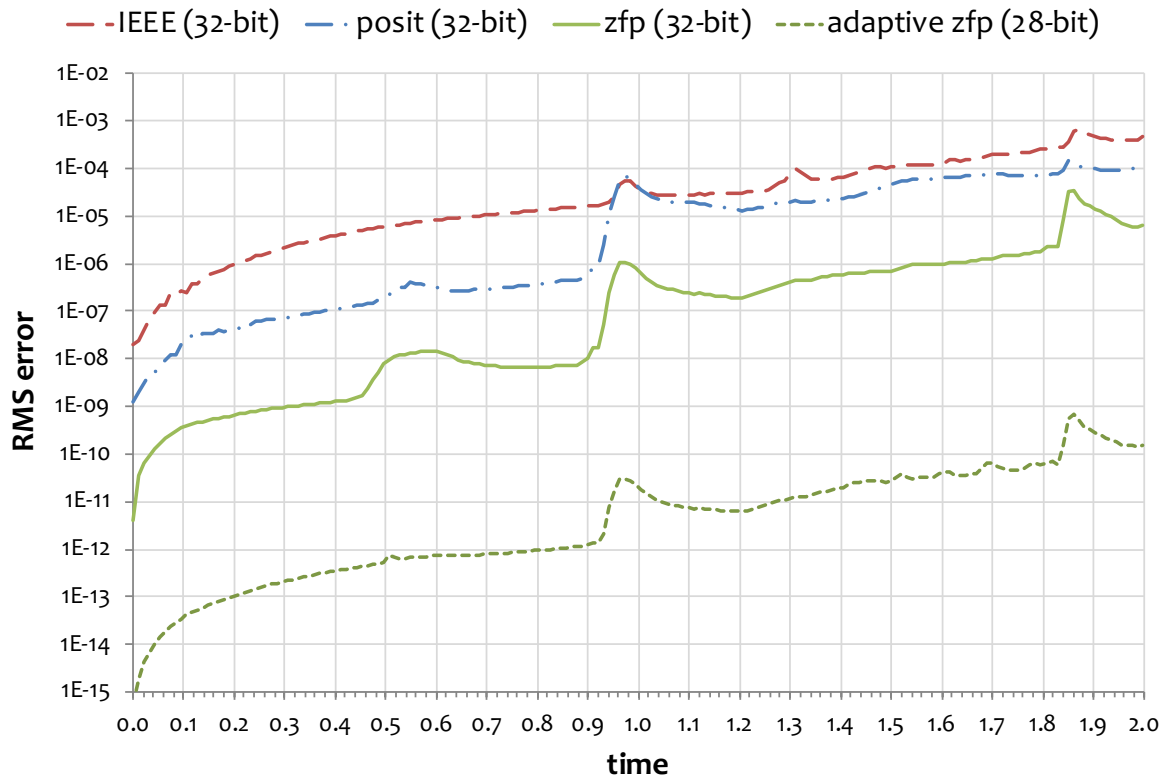
64 bits/value



ZFP adaptive arrays improve accuracy in PDE solution over IEEE by 6 orders of magnitude using less storage

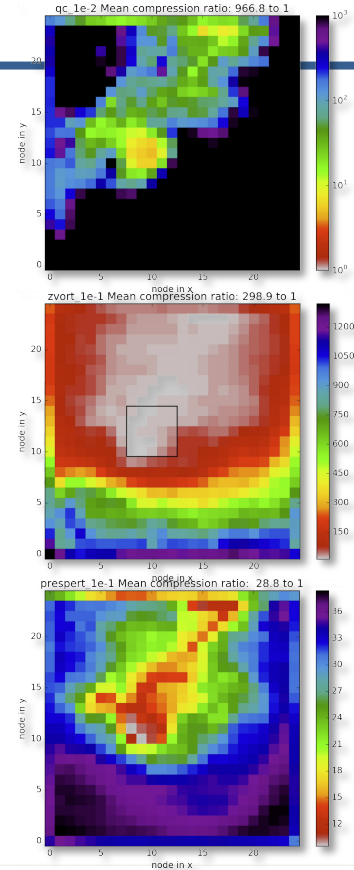


Adaptive-rate ZFP increases accuracy over IEEE float by 6 orders of magnitude while using less storage

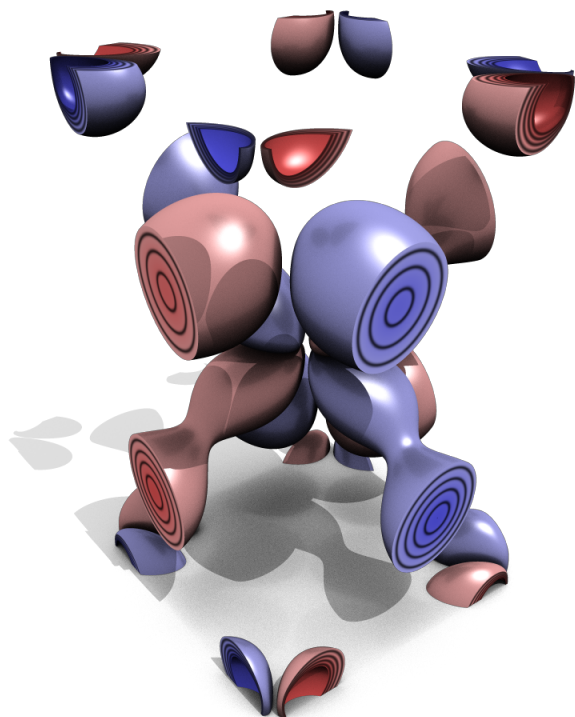


I/O Compression: ZFP reduces I/O by 30x on average in CM1

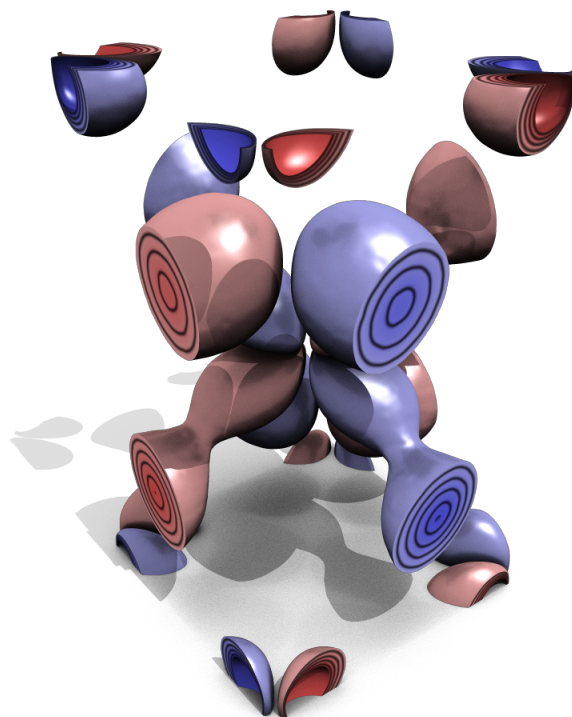
[Work done by Leigh Orf, UW-Madison]



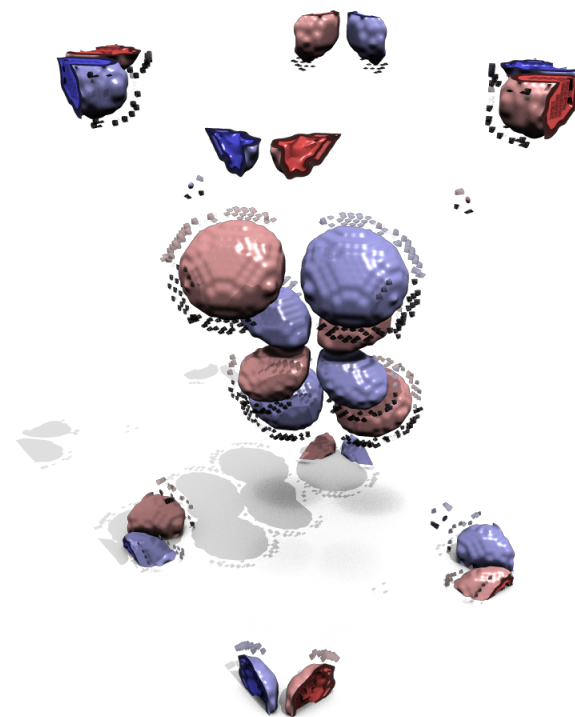
Tabular data: Compressed, pre-computed wavefunctions enable reduced memory footprint in QMCPACK code



uncompressed (32 bits/value)



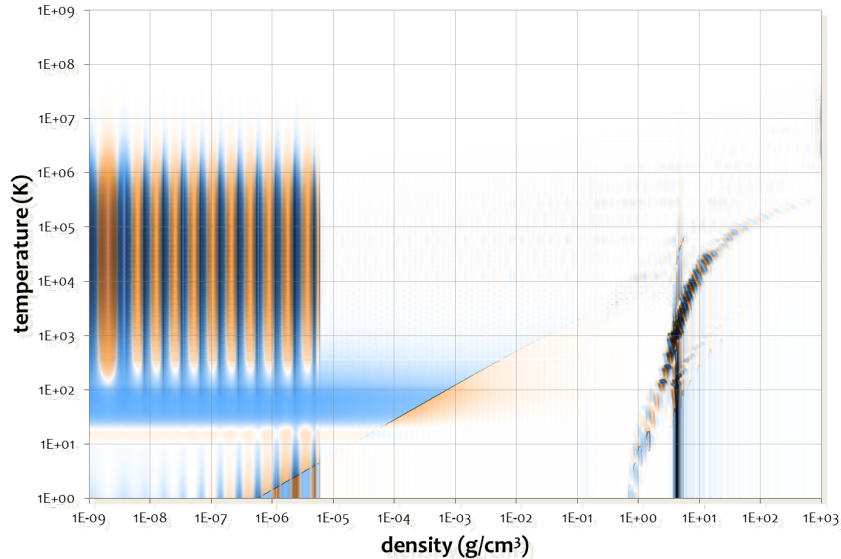
ZFP (1 bit/value)



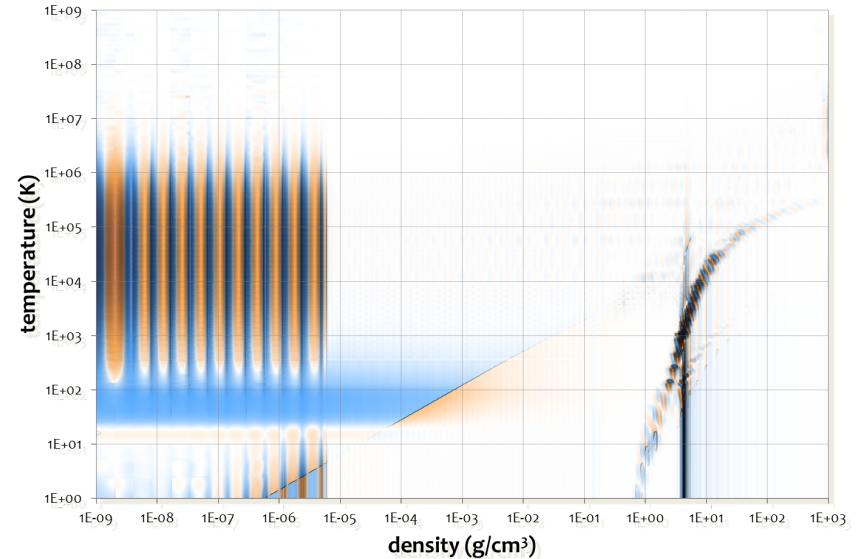
FPZIP (1 bit/value)

Tabular data: LEOS library reduces memory footprint of equation-of-state tables by 4-8x using ZFP

Al 130 thermodynamic consistency (no compression)

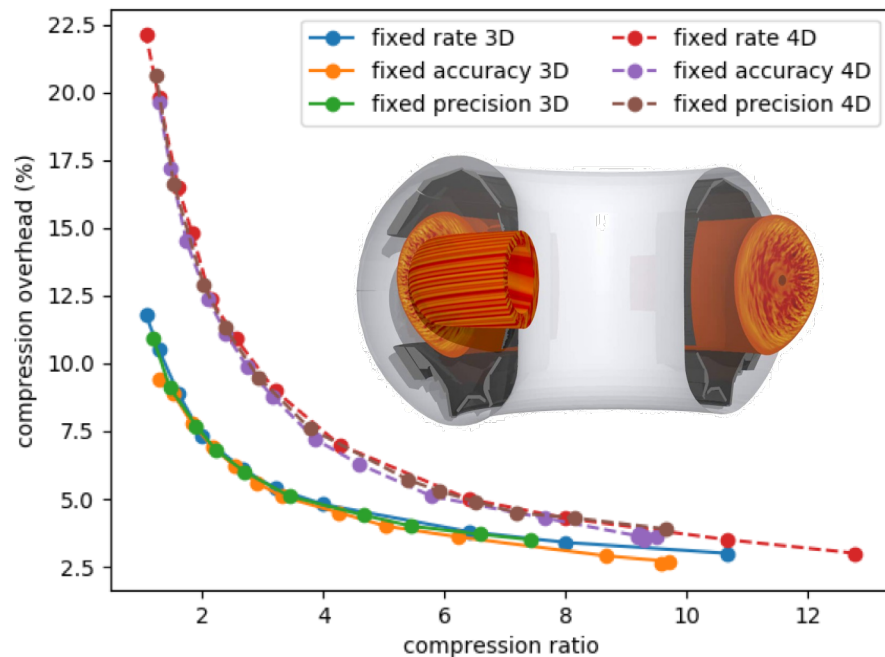
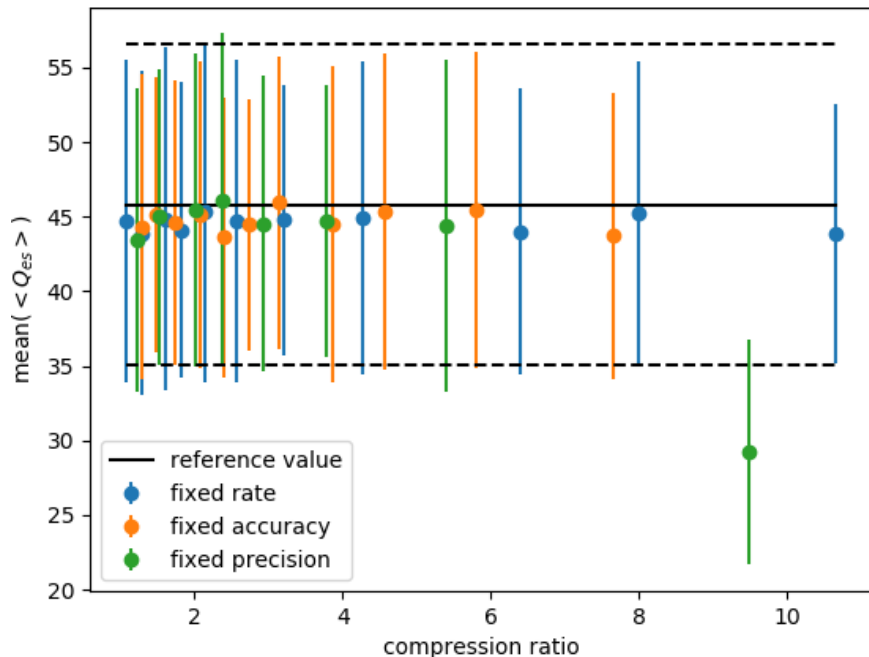


Al 130 thermodynamic consistency (4x compression)



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]

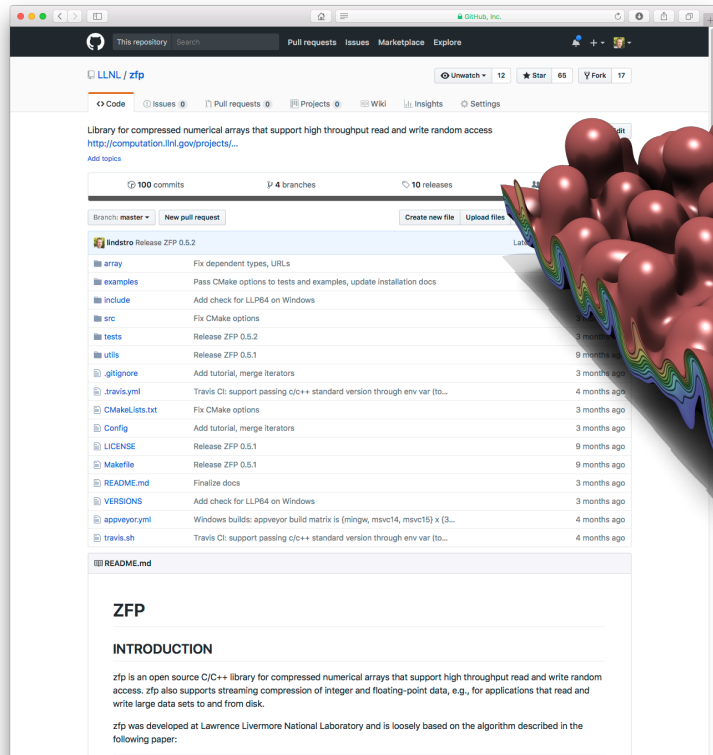


ZFP compression is available in I/O libraries and viz tools

- Registered compression filter in **HDF5**
 - H5Z-ZFP plugin: <https://github.com/LLNL/H5Z-ZFP>
- Available in **ADIOS** I/O library since version 1.11
 - AtoZ = ADIOS + ZFP: <https://github.com/suchyta1/AtoZ>
- Third-party I/O library in **VTK**
 - <https://gitlab.kitware.com/third-party/zfp>
- AVX implementation in **Intel IPP**
 - <https://software.intel.com/ipp-dev-reference>
- Available in **VTK-m** since December 2018
 - <http://m.vtk.org/images/c/c8/VTKmUsersGuide.pdf>



ZFP is BSD licensed and available on GitHub: <https://github.com/LLNL/zfp>



LLNL / zfp

Library for compressed numerical arrays that support high throughput read and write random access
[http://computation.llnl.gov/projects/...](http://computation.llnl.gov/projects/)

100 commits 4 branches 10 releases

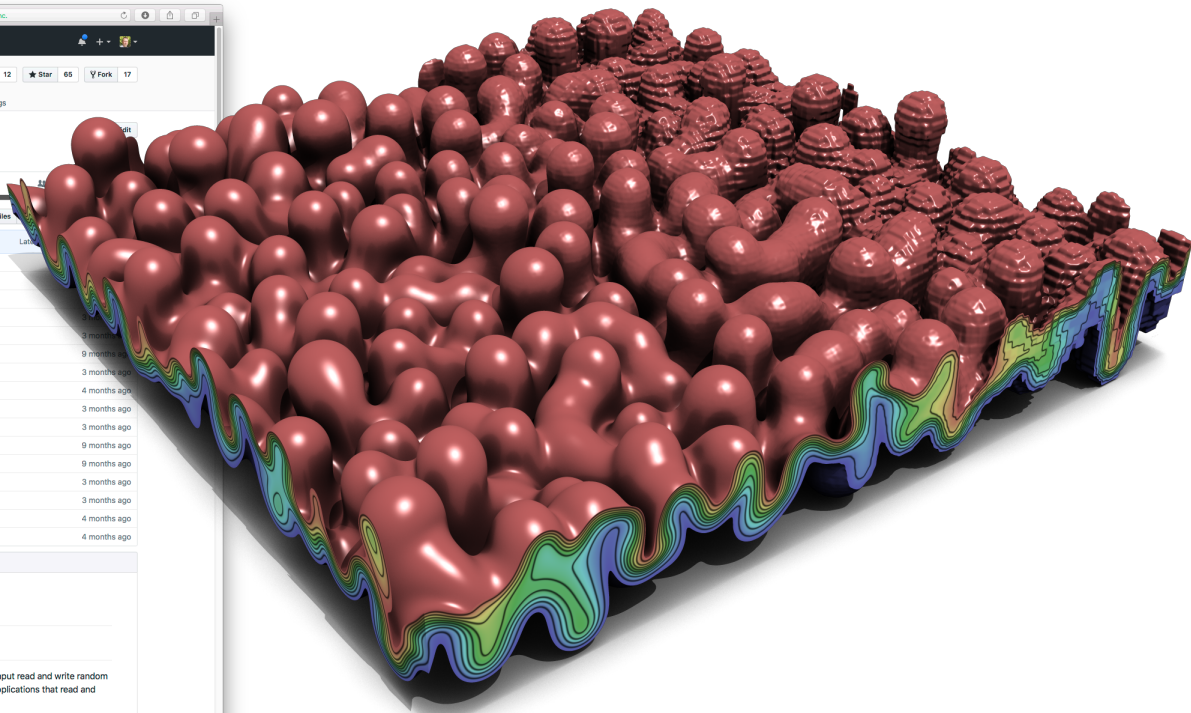
Commit	Message	Time
indstro	Release ZFP 0.5.2	3 months ago
array	Fix dependent types, URLs	9 months ago
examples	Pass CMake options to tests and examples, update installation docs	3 months ago
include	Add check for LLP64 on Windows	4 months ago
src	Fix CMake options	3 months ago
tests	Release ZFP 0.5.2	3 months ago
utils	Release ZFP 0.5.1	9 months ago
ghignore	Add tutorial, merge iterators	3 months ago
travis.yml	Travis CI: support passing c/c++ standard version through env var (to...	4 months ago
CMakeLists.txt	Fix CMake options	3 months ago
Config	Add tutorial, merge iterators	3 months ago
LICENSE	Release ZFP 0.5.1	9 months ago
Makefile	Release ZFP 0.5.1	9 months ago
README.md	Finalize docs	3 months ago
VERSIONS	Add check for LLP64 on Windows	3 months ago
appveyor.yml	Windows builds: appveyor build matrix is (mingw, msvc14, msvc15) x (3...	4 months ago
travis.sh	Travis CI: support passing c/c++ standard version through env var (to...	4 months ago

ZFP

INTRODUCTION

zfp is an open source C/C++ library for compressed numerical arrays that support high throughput read and write random access. zfp also supports streaming compression of integer and floating-point data, e.g., for applications that read and write large data sets to and from disk.

zfp was developed at Lawrence Livermore National Laboratory and is loosely based on the algorithm described in the following paper:





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.