ZFP: compressed floating-point arrays

ECP Annual Meeting 2021

Peter Lindstrom

March 30, 2021

LLNL-PRES-820872

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

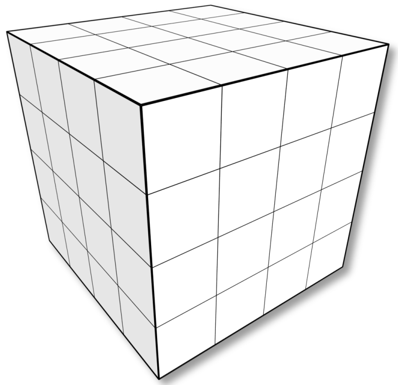
Lawrence Livermore National Laboratory

What is ZFP?

- **Answer #1:** A new, efficient **number format** for small vectors and tensors
  - Alternative to IEEE 754/SSE/AVX/tensor core registers, bfloats, posits, flexpoint, …
- **Answer #2:** An implementation of **multidimensional arrays** with user-specifiable memory footprint or accuracy
  - Alternative to std::vector, Eigen/GSL/Kokkos/NumPy arrays, …
- **Answer #3:** A fast, **streaming compressor** for large floating-point & integer arrays
  - Alternative to gzip, bzip2, blosc, fpzip, JPEG, …

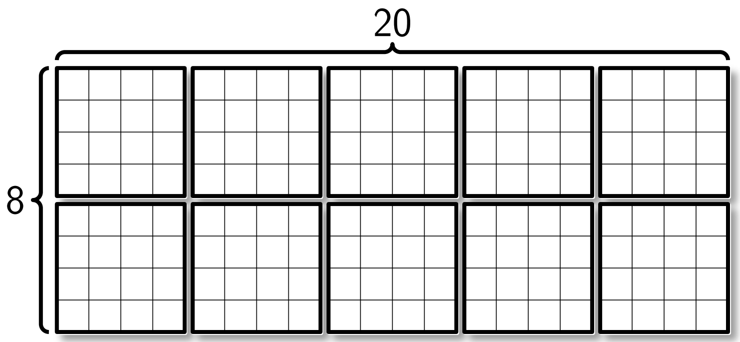
ZFP is a compressed number format for multi-dimensional floating-point arrays

- ZFP compactly represents small vectors and tensors 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 float approximation 1/5 = 0.001100110011… ≈ 0.0011
    - May incur round-off error
    - "Common" blocks have shorter codes ⟹ less or no round-off error
  - **H/W friendly encoder:** integer additions and bitwise operations
  - **Replaces IEEE 754** as number format for numerical computations
    - Usually orders of magnitude **more accurate** than IEEE 754



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] ⇔ std::vector<double> a(n) ⇔ zfp::array<double> a(n, **bits_per_value**)
    - C, experimental NumPy APIs are also 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];
}
```

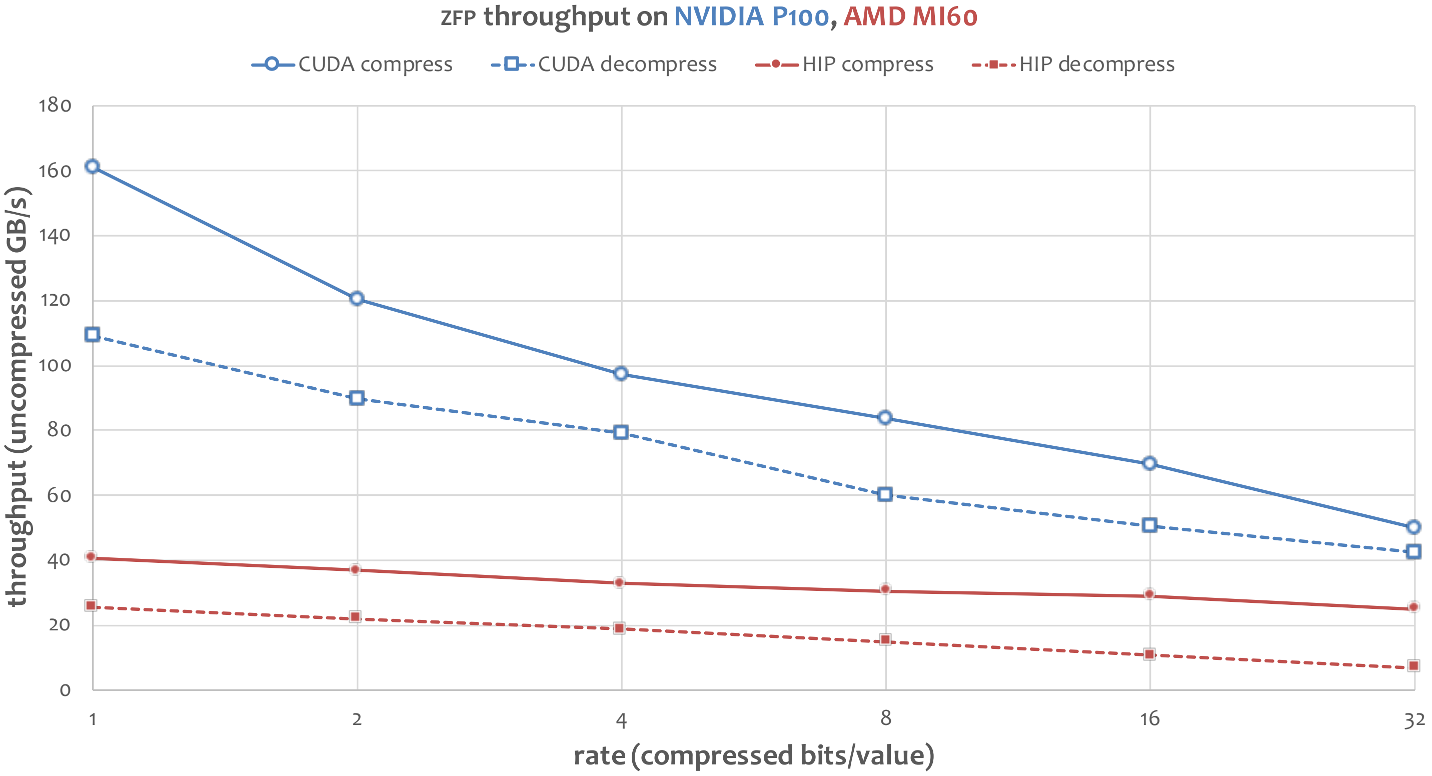
- required changes
- optional changes for improved readability

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

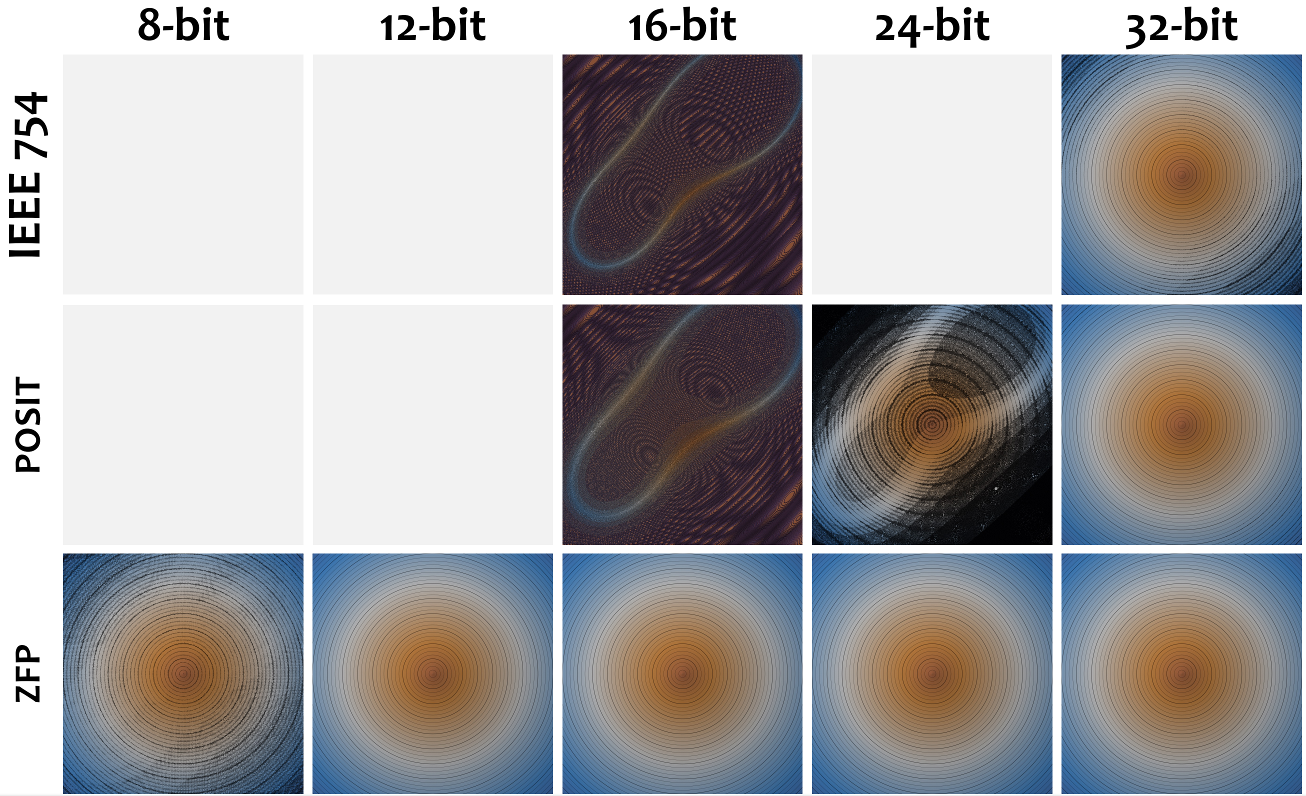
- ZFP also supports streaming compression for I/O, communication, storage
  - Supports absolute and relative **error tolerances** and **lossless** compression
  - Serial, **OpenMP**, **CUDA**, **HIP**, and **FPGA** implementations
    - Up to **160 GB/s** parallel throughput
  - **C**, **C++**, **Python**, **Fortran** bindings
    - 3rd party Julia & Rust bindings available
  - I/O & viz support: **ADIOS**, **Conduit**, **HDF5**, **Intel IPP**, **OpenZGY**, **Silo**, **TTK**, **VTK-m**, …

- ZFP has other nice properties
  - Supports **spatially adaptive** compression
  - Supports **progressive reconstruction** (aka. SNR scalability)
  - **Resilient** to data corruption

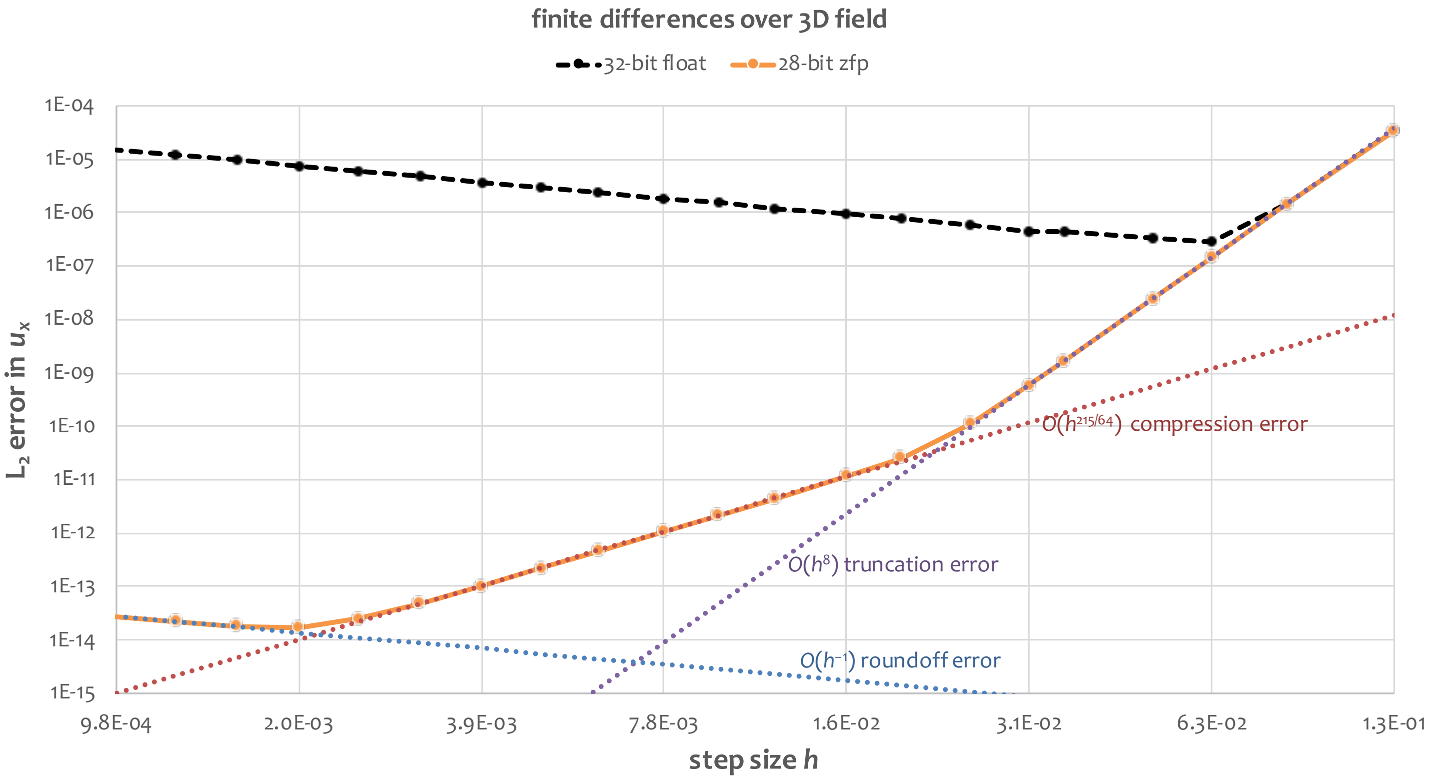
ZFP GPU compression achieves up to 160 GB/s throughput



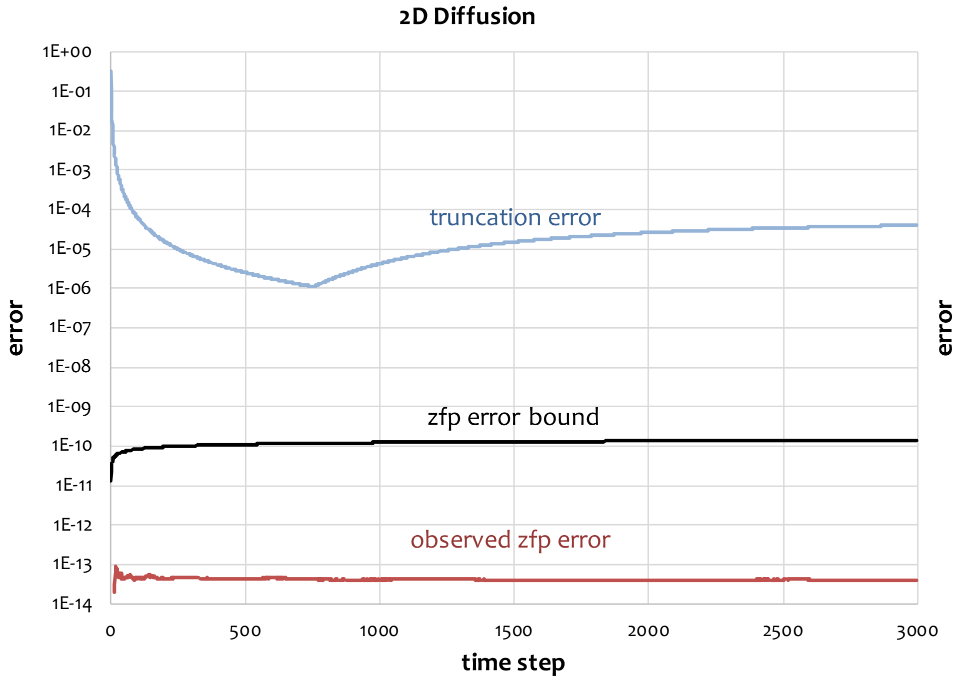
ZFP *improves* accuracy in finite difference computations using less storage than IEEE 754 and POSITS

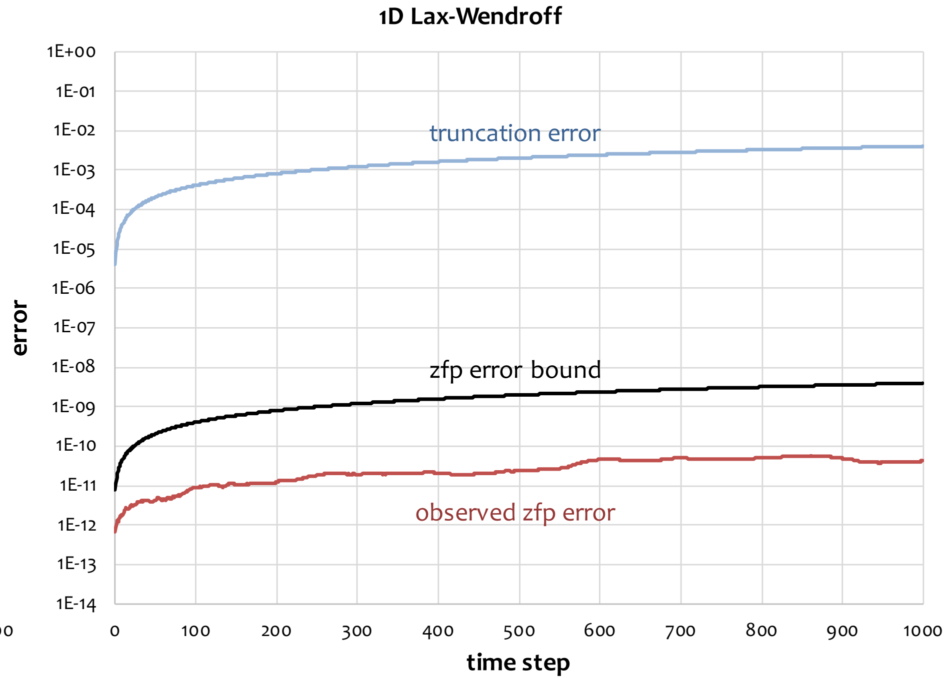


Contrary to conventional floating-point, finite-difference accuracy using ZFP *increases* with grid resolution



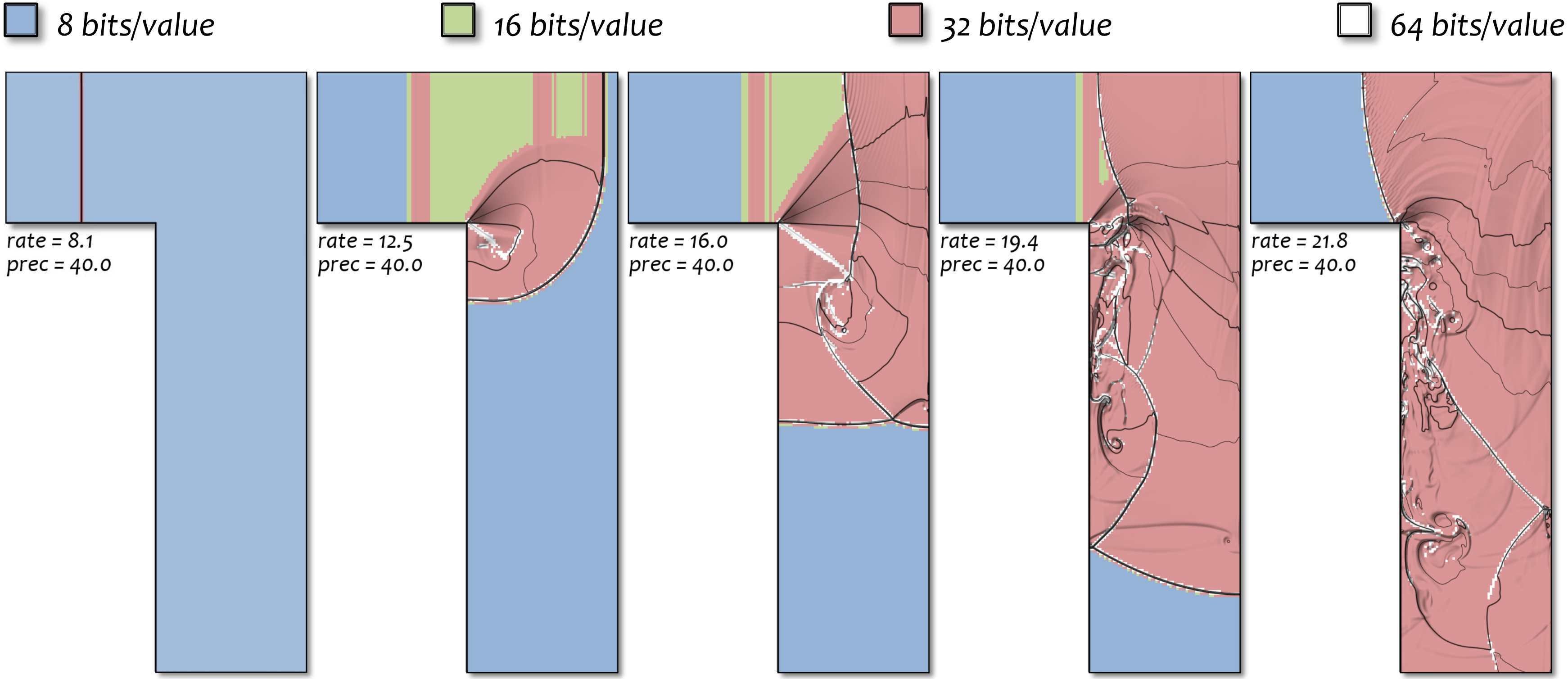
We have developed rigorous error bounds for ZFP, both for static data and in iterative methods



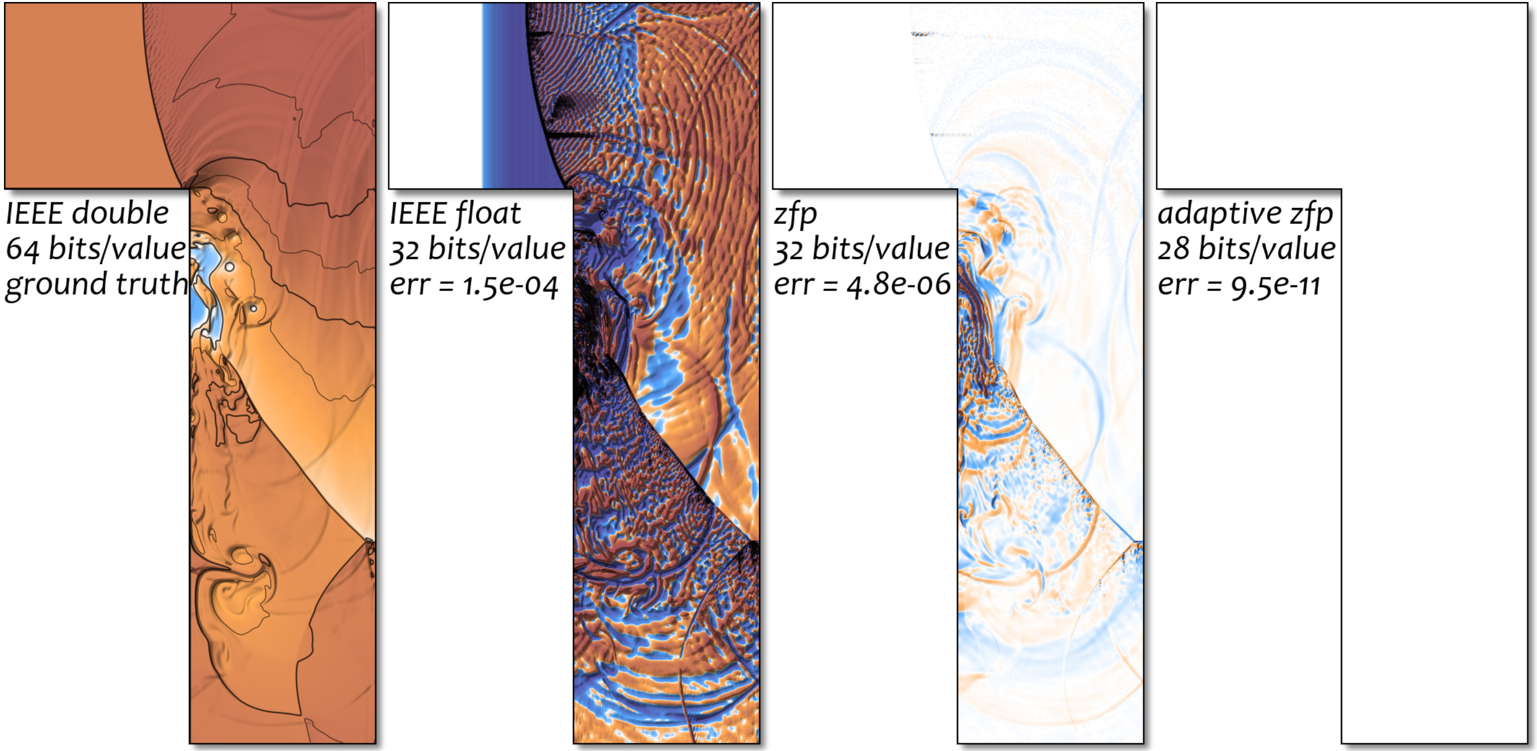


Work by Alyson Fox and James Diffenderfer

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

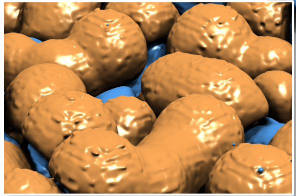
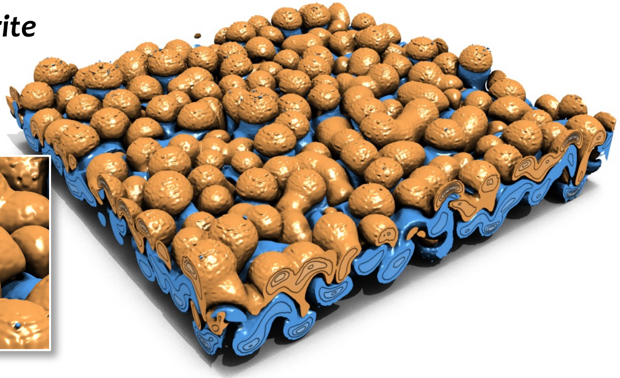


ZFP adaptive arrays improve accuracy in PDE solution over IEEE by 6 orders of magnitude using less storage

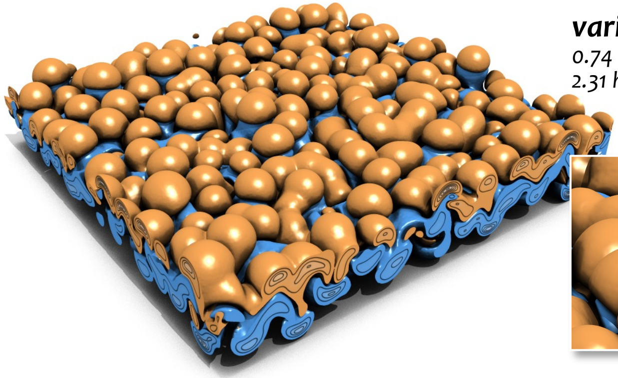
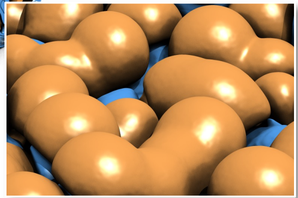


ZFP’s variable-rate arrays improve accuracy per bit stored and in some applications reduce time to solution

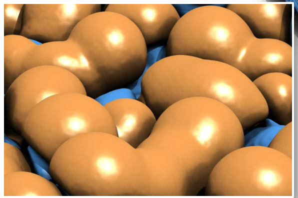
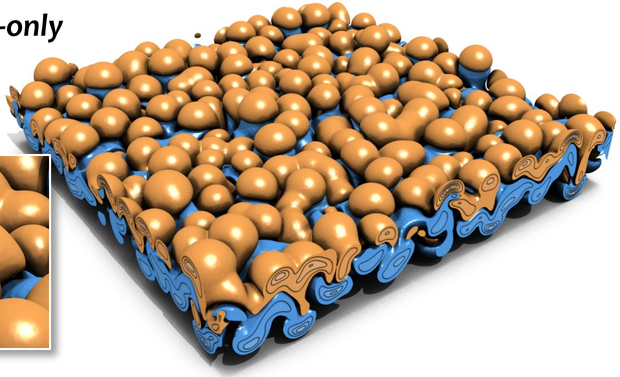
fixed-rate read-write
1.00 bits/value
3.01 hours

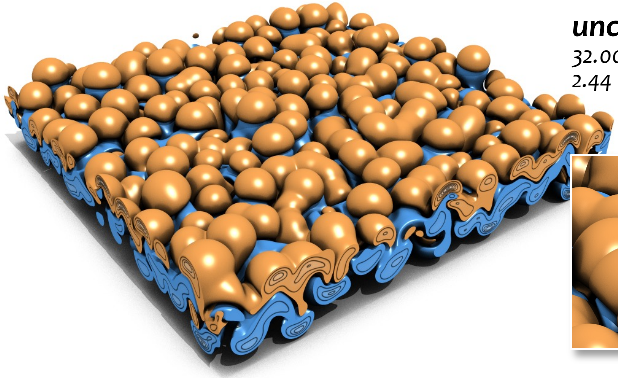
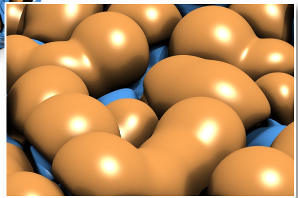
variable-rate read-write
0.74 bits/value
2.31 hours

variable-rate read-only
0.68 bits/value
2.29 hours

uncompressed
32.00 bits/value
2.44 hours

Lawrence Livermore National Laboratory

Acknowledgment
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.