Special Issue Paper

International Journal of HIGH PERFORMANCE COMPUTING APPLICATIONS

The International Journal of High Performance Computing Applications
2025, Vol. 39(1) 104–122

DOI: 10.1177/10943420241284023
journals.sagepub.com/home/hpc
Sage

ZFP: A compressed array representation for numerical computations

Peter Lindstrom[1], Jeffrey Hittinger[1], James Diffenderfer[1], Alyson Fox[1], Daniel Osei-Kuffuor[1] and Jeffrey Banks[2]

Abstract

HPC trends favor algorithms and implementations that reduce data motion relative to FLOPS. We investigate the use of lossy compressed data arrays in place of traditional IEEE floating point arrays to store the primary data of calculations. Simulation is fundamentally an exercise in controlled approximation, and error introduced by finite-precision arithmetic (or lossy compression) is just one of several sources of error that need to be managed to ensure sufficient accuracy in a computed result. We describe ZFP, a compressed numerical format designed for in-memory storage of multidimensional arrays, and summarize theoretical results that demonstrate that the error of repeated lossy compression can be bounded and controlled. Furthermore, we establish a relationship between grid resolution and compression-induced errors and show that, contrary to conventional floating point, ZFP *reduces* finite-difference errors with finer grids. We present example calculations that demonstrate data reduction by 4x or more with negligible impact on solution accuracy. Our results further demonstrate several orders-of-magnitude increase in accuracy using ZFP over IEEE floating point and Posits for the same storage budget.

Keywords

Lossy compression, reduced precision, number representations, error analysis, partial differential equations

1. Introduction

Current trends in computing hardware are diminishing the primacy of hardware clock speed and peak floating-point operations per second (FLOPS) as performance predictors in high-performance computing (HPC) The ASCAC Subcommittee for the Top Ten Exascale Research Challenges (2014); Exascale Mathematics Working Group (2014). The end of Dennard scaling has favored processors with more computing units as clock speeds have stagnated Bohr (2007); Borkar and Chien (2011). The energy associated with floating-point operations has reduced to the point that data movement and volatile storage are now greater concerns in the pursuit of performance Shalf et al. (2011); Michelogiannakis et al. (2014). Performance increases are generally coming from more on-node processing capabilities as opposed to more nodes. For hybrid CPU-GPU architectures, one of the primary challenges is reducing data movement between main and GPU memory, which could be ameliorated if the data could remain resident in the memory of the GPU. With a likely future of more heterogeneous architectures, moving data to the appropriate processor at the appropriate phase of the computation will continue to be a challenge. Many common simulation approaches are already memory bandwidth and/or capacity limited, which means that the peak FLOPS achieved in real calculations is a small fraction of the theoretical peak.

Because of these realities, the HPC physical simulation community has pivoted to consider algorithms with higher arithmetic intensity—the FLOPS per byte of data Williams et al. (2009); Exascale Mathematics Working Group (2014). For simulations modeled by partial differential equations (PDEs), high-order discretizations execute more useful FLOPS per byte because they require both fewer degrees of freedom (DOFs) for a given level of accuracy and more operations per degree of freedom. Directly reducing the number of DOFs is another strategy enabled by adaptive mesh refinement and/or by moving mesh with features of

[1]Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Livermore, CA, USA
[2]Rensselaer Polytechnic Institute, Troy, NY, USA

Corresponding author:
Peter Lindstrom, Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, 7000 East Ave, L-478, Livermore, CA 94550, USA.
Email: pl@llnl.gov

the solution, both of which reduce the resolution requirements by concentrating resolution only where it is needed. An approach that has seen a resurgence is to use less data by reducing the precision of representations used in the calculation. There are numerous examples of mixed-precision and reduced-precision implementations to reduce data storage and motion and to leverage specialized hardware Abdelfattah et al. (2021). Of course historically, we know that, with care, many scientific calculations can be done in single precision, but the potential savings in terms of data reduction can be no higher than a factor of two going from double to single precision. Interest in half precision Markidis et al. (2018) and BFLOAT Burgess et al. (2019); Kalamkar et al. (2019) has been driven by data science applications where there may be different trade-offs with respect to the number of bits used to represent the mantissa or exponent. Provided that calculations on these lower-precision types can be stabilized for poorly conditioned systems, venerable techniques like iterative refinement can be used to improve the precision of a result while computing with lower-precision data types Haidar et al. (2017, 2020); Abdelfattah et al. (2020).

Ideally, we would use the minimum number of bits per floating-point value to store only as much useful information as is present. In fact, while double precision may provide 15 or more "significant" digits, in reality, many of these digits do not possess useful information about the desired solution; in practice, we compute to this precision because it helps alleviate finite-precision issues. Physical simulation is the art of controlled approximation: sets of equations approximate physical theories (to various degrees of fidelity), discrete systems of equations approximate the continuous model, these discrete systems often require iterative schemes to approximate the system solution, and these systems are solved on digital computers that approximate real numbers with finite precision numbers. Several approximation errors typically corrupt the majority of the "significant" digits: truncation error from the discrete approximation, iteration error from the approximate iterative solver, and roundoff error from the use of finite precision. Add to this the model form errors and uncertainties in model parameters and initial and boundary conditions, and a case can be made that, for many scientific and engineering calculations, it is folly to trust more than 2–3 digits even in double precision.

Thus, for many problems that have and will continue to motivate the need for HPC, we are storing, moving, and computing on large amounts of data that have low information content. We need to be more bit-efficient, but in a way that still guarantees correctness and robustness to roundoff accumulation. There has been a recent revival in the study of alternative floating-point representations Gustafson and Yonemoto (2017); Lindstrom et al. (2018); De Dinechin et al. (2019); Lindstrom (2019); Buoncristiani et al. (2020). We propose, instead, the use of arrays compressed in small chunks such that memory-efficient random access is effectively preserved, but similarities in nearby data can be leveraged for more efficient storage. Lossy compression can provide significant data reductions while not sacrificing accuracy if the "lost" data had low information content, for example, it represented errors from any of the other approximations involved in the simulation. Effectively, we would add a new type of error that would be dominated by one or more of the other error terms. [1] Furthermore, if the data are stored in a compressed form, decompressed incrementally as needed into IEEE double precision in cache for calculation, and re-compressed before being returned to main memory, one gains the benefit of storing and moving much less data while preserving the calculation benefits of double precision. Of course, decompression can be done into any floating-point representation, so this process separates the concern of storage size from the concern of finite-precision arithmetic in an easily managed way. Thus, we demonstrate that an approach based on a compressed data array type, with calculations in double precision, is more *bit efficient* than using single precision without a significant loss of accuracy or the complications of computing at lower precision.

Of course, multiple issues must be addressed to justify broad adoption of compressed data types. Floating-point data is not easily compressed, so a suitable lossy floating-point compression algorithm with rapid random access is needed. Practitioners need confidence that repeated compression and decompression of solution data within the core numerics of a calculation can be done with controllable error that ensures both accuracy and stability. Compressed data arrays have a clear benefit by, in effect, increasing the memory capacity, which is particularly beneficial for GPUs with limited high-bandwidth memory. We have more ambitious aspirations, however, to overcome bandwidth limitations, which will require large data reductions while, at the same time, the overhead of (de)compression must not offset potential gains. We will review and extend progress in each of these areas to justify our primary thesis of this paper: *Compressed data types are a technology suitable and with substantial benefits for a significant class of HPC applications, which warrants additional attention and investment from the community.*

Specifically, we focus on the ZFP floating-point compressed array Lindstrom (2014), which is an efficient storage representation well-suited to the task. Unlike most contemporary floating-point compressors Lindstrom and Isenburg (2006); Burtscher and Ratanaworabhan (2007); Lakshminarasimhan et al. (2011); Yang et al. (2015); Liang et al. (2018); Ainsworth et al. (2019); Ballester-Ripoll et al. (2020); Li et al. (2023), which sequentially compress whole or large portions of arrays, ZFP supports on-demand random access to small blocks of data that are (de)compressed

independently inline with the computation. ZFP allows these blocks to be compressed to a user-specified number of bits, which simplifies random access and memory management and gives the user a knob to tune the accuracy needed for each array in the application. Development of ZFP originated in an earlier study that demonstrated the viability of using a lossy-compressed representation of simulation state Laney et al. (2013). While this study demonstrated empirically that lossy compression could be used with acceptable levels of error, this work relied on conventional streaming compressors Lindstrom and Isenburg (2006); Wegener (2013) to perform decompression of whole arrays, to advance the solution one time step, and then to re-compress the state.

Although motivated by the potential for compression to reduce memory bandwidth usage, streaming compression when used this way *increases* data movement, as large arrays are (de)compressed from RAM to RAM with little or no cache reuse. In contrast, ZFP supports fine-grained random access to small chunks of uncompressed data on the order of one or a few hardware cache lines, which makes it possible to limit data motion to between RAM and cache—in compressed form—and to realize a net decrease in overall data movement. Such inline compression is relatively new in HPC. We are aware of only three competing approaches: the inline compression scheme of Fu et al. Fu et al. (2017), which in effect amounts to turning 32-bit floats into 16-bit fixed-point values via a normalization step (a form of scalar quantization); the vector quantization approach of Trojak and Witherden Trojak and Witherden (2021), which is limited to 3-component vector-valued data and a fixed 1.5x compression ratio; and Blaz Martel (2022), a block-based compression scheme for 2D arrays that fixes storage at 360 bits per block of 8×8 values (or 5.625 bits/value). We instead propose the use of more general floating-point compressors that allow for random access with incremental (de)compression and arbitrary compression ratios specified by the user, and we restrict our investigation to ZFP because of its data locality property and because of recent work that established theoretical errors bounds for lossy ZFP compression Diffenderfer et al. (2019); Fox et al. (2020).

In the following sections, we review recent results of the mathematical analysis of the ZFP compression algorithm, including an analysis that demonstrates the stability and control of the error from lossy (de)compression done repeatedly in the core of two common classes of numerical algorithms, and provide new results on ZFP error convergence. These theoretical results are supported by several numerical experiments involving ZFP compressed arrays in physical simulation codes that show that data reductions by factors of four or higher can be achieved without appreciable loss of accuracy. This collection of results makes a strong case that compressed data representations are a feasible and potentially significant technology for better data efficiency in the inner numerical loops in HPC simulation.

2. Compressed floating-point arrays

ZFP implements multidimensional compressed arrays for representing predominantly continuous scalar fields, such as those that typically arise in PDE solutions. Although discontinuities like shocks and material interfaces may be present in such fields, they tend to be confined to lower-dimensional manifolds, with most of the field being at least C^0 continuous and spatially correlated. Autocorrelation—correlation between adjacent values on a grid—is a source of redundancy for conventional uncompressed number representations, where adjacent scalar values often share sign, exponent, and several leading mantissa bits. ZFP achieves compression by finding and removing such correlations and by discarding in a controlled way trailing mantissa bits often contaminated with error.

ZFP allows applications to store arrays in memory in compressed form with fine control over memory footprint or accuracy, for example, for representing the state arrays in simulation, data analysis, and visualization. ZFP decompresses data on demand in small chunks to a conventional number type like IEEE floating point, on which computations can be done, and compresses modified chunks back to memory. ZFP supports both lossy and lossless compression but has been tuned for the lossy use case with the assumption that compression-induced errors—which are analogous to floating-point roundoff errors—can be kept significantly smaller than other sources of error. As we shall see, ZFP typically provides a level of accuracy per bit stored that far exceeds conventional number representations like IEEE 754. As such, ZFP offers both reduced storage and bandwidth and improved accuracy for a given memory footprint. Moreover, ZFP provides a "continuous" storage size knob for mixed-precision implementations through a single data type and API, unlike mixed-precision implementations that require specialized code to handle heterogeneous scalar types.

ZFP has evolved substantially since its first release Lindstrom (2014). In this section, we describe ZFP CODEC 5—the compressed representation supported in ZFP since version 0.5.0 (released in 2016). We first give a brief overview of the ZFP compressed block representation and then describe its fixed-rate array data structures.

2.1. Compression scheme

Unlike early compression work done in the visualization community Ning and Hesselink (1992); Schneider and Westermann (2003), which focused on the use case of write-once, read-many, ZFP was designed to support fine-

grained random access reads and writes with symmetric performance. Toward this end, ZFP partitions d-dimensional floating-point arrays into blocks of 4^d values each, for example, a 3D block holds $4 \times 4 \times 4$ values. Each block is encoded independently to a finite-length bit string in constant time, allowing random access reads and writes at block granularity. For arrays whose dimensions are not multiples of four, ZFP pads any partial blocks with values from the same block in a manner that promotes compression. Thus, we focus our discussion on the encoding of individual blocks.

The ZFP compression scheme for floating-point blocks can be described as a sequence of distinct steps, which are outlined in detail in Diffenderfer et al. (2019). First, ZFP employs a block-floating-point transform, where each value within a block is represented as a $(q + 1)$-bit two's complement signed integer that is scaled by a single per-block exponent, e. Here $q \in \{30, 62\}$, depending on whether single- or double-precision data with $k \in \{24, 53\}$ mantissa bits is being encoded. The block-floating-point transform is essentially lossless, though the alignment to a common exponent may incur loss of some least significant bits if the values in the block differ sufficiently in magnitude. [2]

For smooth data, the integer values are often highly correlated and share several leading bits. Such redundancy is eliminated using a decorrelating linear transform that is applied once along each dimension. The goal of this transform is to eliminate any covariance between the values in a block and to produce *transform coefficients* that are small in magnitude, allowing a more compact encoding. Several such transforms have been proposed in the compression literature, of which the *discrete cosine transform* (DCT) is perhaps the best known. In Lindstrom (2014), it was shown how many of these transforms are part of a family that can be characterized by a single parameter. ZFP CODEC 5 is based on a parameter choice that gives rise to orthogonal basis functions that are tensor products of Gram polynomials Barnard et al. (1998) of increasing degree. For computational efficiency, ZFP uses a slight modification to this basis that allows the transform to be implemented using only integer additions, subtractions, and arithmetic right shifts via a sequence of *lifting steps* Daubechies and Sweldens (1998). This choice of basis was made with care to ensure that the transform coefficients decay rapidly with grid resolution—a property we will explore further below.

Transform coefficients indexed in 3D by (i, j, k) are then sequentially ordered by expected magnitude using a predetermined ordering. As discussed below, we expect the coefficient magnitude to decrease with increasing $i + j + k$, which we use as a sort key. This reordering essentially provides a generalization of the 2D zig-zag ordering employed by JPEG Wallace (1992).

Following reordering, the two's complement coefficients are converted to their *negabinary* representation. Negabinary uses as base -2 instead of the usual base $+2$ employed in binary representation. The purpose of this conversion is to ensure that small-magnitude coefficients are represented with many leading zero-bits, whereas two's complement uses leading zeros or leading ones depending on sign.

The binary coefficient matrix of 4^d $(q + 2)$-bit negabinary values is then transposed so that each row corresponds to a *bit plane*—a set of bits for the 4^d values that have the same place value—with each column representing a coefficient. Due to the previous coefficient ordering and negabinary conversion, small coefficients appear on the left and large ones on the right in this matrix, such that rows near the top tend to have many leading zero-bits. The matrix is then encoded losslessly one bit plane at a time, from top to bottom and from right to left, using a simple variable-length code. This code represents any sequence of leading zero-bits using a single bit, which enables compression. The leading one-bit and any trailing bits are encoded verbatim and are interleaved with a small number of additional control bits needed for the variable-length code.

Finally, the resulting bit stream for the block is optionally truncated to meet a termination criterion, such as a prescribed compressed size or error tolerance. Any truncated bits are replaced with zeros. The effect of this truncation is analogous to rounding in the IEEE floating-point representation and is the main source of loss in accuracy when converting numerical data to ZFP. Next, we will discuss one of several different approaches to bit stream truncation.

2.2. Compressed-array data structure

Based on the ZFP block representation, ZFP provides C++ classes for in-memory compressed storage of multidimensional arrays. These arrays appear and behave much like conventional uncompressed representations (e.g., C/C++ arrays, STL vectors). Using operator overloading, the details of compression and decompression are entirely hidden from the user, who interacts with the arrays as though each array element were directly addressable. ZFP arrays provide access to elements in several ways: via multidimensional indexing, via linear (flattened) sequential indexing, and via iterators, proxy pointers, and views. Thus, converting an existing application to make use of ZFP arrays can be as simple as replacing standard array storage with this alternative type.

Under the hood, ZFP makes use of a small write-back software cache of decompressed blocks in IEEE format to avoid the need to compress and decompress blocks upon every read or write access. If the application access pattern exhibits reasonable locality, as in stencil operations and streaming passes over the data, the need for compression (which may introduce loss) can be kept to a minimum. Note

that a block is compressed back to persistent storage when evicted from the cache only if it has been modified, that is, following one or more write accesses. For all results presented in this paper, we use the default cache size of $\sqrt{B}$ blocks, where B is the total number of blocks in the array.

As exploited in Lindstrom (2014), bit stream truncation to a fixed number of bits (as specified by the user) allows for straightforward random access to blocks with no additional data structures required to track their memory locations. ZFP *fixed-rate arrays* allow the user to specify the *rate*—the number of bits of compressed storage per array element—in increments of as little as 2^{3-2d}, for example, 1/8 bit for 3D ($d = 3$) arrays. Although ZFP has been extended to also support variable-rate arrays that allocate bits more intelligently over the domain, we here focus only on its fixed-rate arrays.

ZFP's support for random access is due to a unique combination of design choices that sets it apart from other compressors: (1) ZFP is nonstatistical and therefore does not require learning, encoding, and updating data statistics (e.g., probability table, Huffman tree, dictionary, data-dependent basis, etc.) when array elements change in computations. (2) ZFP operates on tiny, independent blocks of data smaller than filter stencils commonly used in wavelet and other multiresolution compressors. (3) ZFP uses embedded bit plane coding that allows it to truncate per-block compressed streams at any point to fit within a prescribed bit budget, thus simplifying random access by fixing the compressed block size. Such truncation is analogous to rounding in conventional floating point.

The use of very small data blocks—a distinguishing feature of ZFP—is also of key importance in achieving a net reduction in data movement. If small enough, decompressed blocks can fit in hardware registers or cache after being fetched in compressed form from main memory or higher-level caches. For example, a 2D ZFP block decompressed to double precision occupies 128 bytes—one or two cache lines. Streaming compressors usually employ far larger chunks (if at all) to hide per-chunk overhead in computation (e.g., thread creation, (de)compressor initialization, data transfer) and storage (to index variable-length chunks). Example default or suggested chunk sizes range from 1 MB in ZARR Miles et al. (n.d.) to 2 MB in IDX2 Hoang et al. (2021) to 128 MB in SPERR Li et al. (2023). At best, such large chunks limit caching to second or higher levels of cache; at worst, chunks are decompressed to main memory, thus only *increasing* data movement.

3. Lossy compression error behavior

A natural concern about the use of a lossy compressed representation is whether the repeated actions of decompression and recompression introduce a compounding error that corrupts the solution, causes the simulation to become unstable, or systematically biases results in some way. To ensure the reliable use of ZFP in HPC simulation, it is essential to establish bounds for the compression error incurred by encoding blocks of floating-point numbers and to demonstrate that the error in the arithmetic operations performed while computing with the decoded ZFP arrays is also bounded. For functions typical for physical simulations, this latter requirement ensures that the fundamental axiom of floating-point arithmetic Trefethen and Bau (1997) is satisfied with respect to using ZFP.

3.1. Bounding ZFP compression errors

It is already established that the floating-point representation of some real number a, denoted $fl(a)$, satisfies the absolute error bound $|fl(a) - a| \leq \varepsilon_{mach}|a|$, where ε_{mach} is the machine epsilon. It follows that the error introduced by representing a d-dimensional array of real numbers, $\boldsymbol{x} \in \mathbb{R}^{n^d}$, satisfies $\|fl(\boldsymbol{x}) - \boldsymbol{x}\|_\infty \leq \varepsilon_{mach}\|\boldsymbol{x}\|_\infty$, which describes the maximum absolute error in the floating point representation of the array data. Equivalent error bounds have been developed for compressing blocks of floating point data using ZFP. Let $\boldsymbol{x}_{\text{ZFP}}$ denote the decoded compressed block array. The bound then takes the form

$$\|\boldsymbol{x}_{\text{ZFP}} - \boldsymbol{x}\|_\infty \leq \varepsilon_{\text{ZFP}}\|\boldsymbol{x}\|_\infty, \tag{1}$$

where ε_{ZFP} is a parameter, analogous to ε_{mach}, that bounds the relative compression error. It corresponds to the parameter K_β from Diffenderfer et al. (2019) and is dependent on the dimensionality of the input data, d; the number of bit planes encoded by ZFP, β; and the Lipschitz constant of the near-orthogonal transform used by ZFP, $k_L = (7/4)\,(2^d - 1)$:

$$\varepsilon_{\text{ZFP}} = \left(\frac{15}{4}\right)^d \left(\delta_k \left[\frac{8}{3}\epsilon_\beta + \epsilon_q\left(1 + \frac{8}{3}\epsilon_\beta\right)(1 + k_L\delta_q)\right] + \epsilon_k\right), \tag{2}$$

where $\epsilon_m = 2^{1-m}$, $\delta_m = 1 + \epsilon_m$, and k and q are fixed by the number of bits in the data type to be compressed.

As $q > k$ and as $k > \beta$ almost always, ZFP errors tend to be dominated by the $\mathcal{O}(\epsilon_\beta)$ term associated with bit stream truncation, with β either specified directly by the user or controlled indirectly through a prescribed bit rate. The $\mathcal{O}(\epsilon_q)$ error term associated with block-floating-point conversion from IEEE FP and roundoff error in the decorrelating transform, and the larger $\mathcal{O}(\epsilon_k)$ term for conversion back to IEEE FP are usually many orders of magnitude smaller.

Computing with ZFP arrays requires performing floating-point arithmetic operations on the decoded compressed floating-point arrays. The entries of a decoded array are just floating-point numbers. However, the result of the arithmetic operation is some real number that needs to be

representable in floating-point form. The fundamental axiom of floating-point arithmetic requires that the error of the floating-point arithmetic operation is no worse than the error incurred in representing its result as a floating-point number. In other words, the relative error in floating-point arithmetic operations is bounded by ε, where ε depends on the implementation used for representing floating-point numbers. If the result of some arithmetic operation, $*$, acting on decoded ZFP arrays $\boldsymbol{x}$ and $\boldsymbol{y}$, is stored in a ZFP parray $\boldsymbol{w}$, then from equation (1), we have that:

$$\|\boldsymbol{w}_{\mathrm{ZFP}} - (\boldsymbol{x} * \boldsymbol{y})\|_\infty \leq \varepsilon_{\mathrm{ZFP}} \|(\boldsymbol{x} * \boldsymbol{y})\|_\infty. \tag{3}$$

In other words, all floating-point arithmetic operations on ZFP arrays are exact up to a relative error of $\varepsilon_{\mathrm{ZFP}}$.

As the error bound in Equation (1) represents a single instance of using ZFP arrays for data representation and the error bound in Equation (3) represents a single operation on ZFP arrays, they provide a basis for error analysis of the use of ZFP arrays in simulations. HPC simulations typically require the repeated application of a linear or nonlinear advancement operator, say g, to a state variable in an iterative manner defined by $\boldsymbol{x}^{i+1} = g(\boldsymbol{x}^i)$, given an initial state $\boldsymbol{x}^0$. When the state variable $\boldsymbol{x}^i$ is represented using ZFP arrays, this iterative process needs to result in a bounded sequence of state variables. Leveraging Equations (1) and (3), theoretical bounds for the i^{th} iterate of sequences computed with ZFP arrays, $\{\boldsymbol{x}^i_{\mathrm{ZFP}}\}_{i\geq 0}$, have been established Fox et al. (2020). Under the hypothesis that g is either Lipschitz continuous with Lipschitz constant in (0, 1) or a Kreiss bounded matrix, two common properties for advancement operators, these bounds have the form

$$\|\boldsymbol{x}^{i+1}_{\mathrm{ZFP}} - \boldsymbol{x}^{i+1}\|_\infty \leq \mathcal{O}(\varepsilon_{\mathrm{ZFP}}) \sum_{j=0}^{i} \|\boldsymbol{x}^j\|_\infty. \tag{4}$$

Note that, if the iterates $\boldsymbol{x}^i_{\mathrm{ZFP}}$ are replaced by a sequence computed using a floating-point representation, such as IEEE 754, ε_{mach} would replace $\varepsilon_{\mathrm{ZFP}}$ in the bound equation (4). The validity of these theoretical bounds has been empirically tested and verified in Diffenderfer et al. (2019); Fox et al. (2020), providing not only assurances on the reliable use of ZFP arrays in HPC simulations, but a means to tune the compression error to below the level of other numerical errors (typically truncation error, but any well-bounded error would suffice). In this way, maximum compression can be achieved without loss of meaningful data, and for deterministic solutions, quantities of interest can be computed to the same formal level of accuracy with lossy compression as without.

3.2. Bias in ZFP compression errors

Empirical studies on the distribution of error introduced by ZFP revealed that, by implementing a minor modification in ZFP, the error distributions are effectively unbiased Hammerling et al. (2019). For example, the data in Figure 1 demonstrate how this minor modification, denoted ZFP-ROUND in Hammerling et al. (2019), essentially eliminates bias in the error distribution of climate data when using ZFP. Theoretical results on bias in the error distribution of ZFP, based on the distribution of the input data, would instill further confidence by establishing that any bias is bounded near zero. Further, these efforts could lead to probabilistic error bounds that are tighter than the existing worst-case error bound, in Equation (1), at some confidence level. By extending the error bound analysis framework from Diffenderfer et al. (2019) to account for the distribution of the input data, theoretical bounds on the expected value of the difference between the original data and the same data represented using ZFP can be established, thereby theoretically bounding the bias. This approach has been pursued in a separate study Fox and Lindstrom (2024), where more details can be found. The bias result is important because it ensures accuracy for problems that exhibit chaotic behavior, where small differences lead to exponentially diverging solution trajectories. In such calculations, the principal quantities of interest are computed from ensembles of calculations and are statistical in nature, so accuracy would suffer from systematic lossy compression biases.

3.3. ZFP error convergence

The error analysis so far has been independent of the discretization of the domain. As the mesh size is reduced and the number of degrees of freedom increases, the number of data accesses and lossy operations (cf. rounding operations) will increase, so it is informative to understand how this error grows with system size. Thus, we provide a new result on the convergence properties of ZFP errors for functions defined on uniform grids of increasing resolution.

Consider a 1D block

$$\mathbf{f} = \left(f\left(-\frac{3}{2}h\right) \;\; f\left(-\frac{1}{2}h\right) \;\; f\left(+\frac{1}{2}h\right) \;\; f\left(+\frac{3}{2}h\right) \right)^T, \tag{5}$$

where $f(x)$ is a thrice-differentiable function discretized on a uniform grid with step size h. The third-order Taylor expansion of $f(x)$ around $x = 0$ gives $\mathbf{f} = T\mathbf{d} + \mathcal{O}(h^4)$ with

$$T = \frac{1}{48}\begin{pmatrix} 48 & -72 & 54 & -27 \\ 48 & -24 & 6 & -1 \\ 48 & 24 & 6 & 1 \\ 48 & 72 & 54 & 27 \end{pmatrix}, \mathbf{d} = \begin{pmatrix} f(0) \\ f'(0)h \\ f''(0)h^2 \\ f'''(0)h^3 \end{pmatrix}. \tag{6}$$

We now apply the ZFP decorrelating transform given by

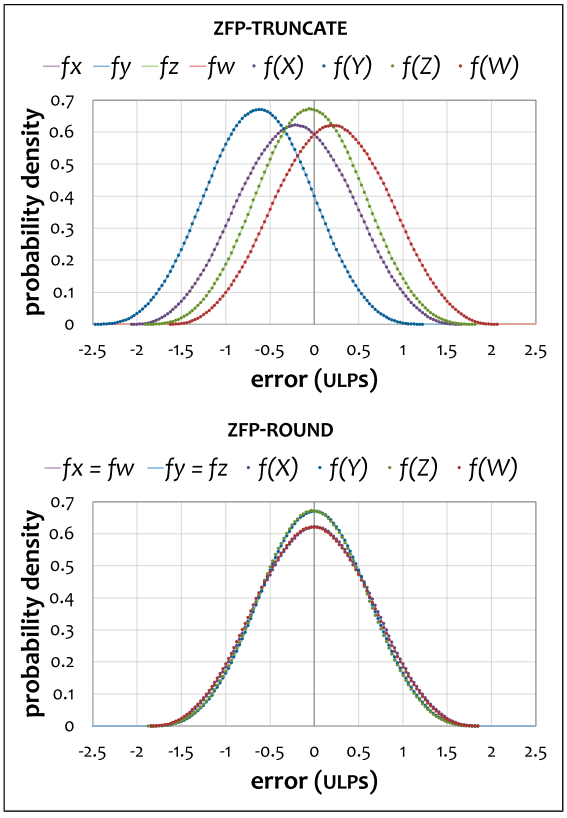

Figure 1. Predicted (curves) and observed (dots) error distributions for 1D compression of climate data (daily averages of surface temperature). *X*, *Y*, *Z*, and *W* denote four random variables, each of which corresponds to one of the four locations within a 1D ZFP block. When using ZFP-ROUND Hammerling et al. (2019), the error distribution is centered around zero. Figure source: Fox and Lindstrom (2024).

$$A = \frac{1}{16}\begin{pmatrix} 4 & 4 & 4 & 4 \\ 5 & 1 & -1 & -5 \\ -4 & 4 & 4 & -4 \\ -2 & 6 & -6 & 2 \end{pmatrix} \tag{7}$$

to compute the transform coefficients $\mathbf{c} = A\mathbf{f} = AT\mathbf{d}$:

$$\mathbf{c} = \frac{1}{48}\begin{pmatrix} 48 & 0 & 30 & 0 \\ 0 & -48 & 0 & -17 \\ 0 & 0 & -24 & 0 \\ 0 & 0 & 0 & 6 \end{pmatrix}\begin{pmatrix} f(0) \\ f'(0)h \\ f''(0)h^2 \\ f'''(0)h^3 \end{pmatrix}. \tag{8}$$

Because AT is upper triangular (a consequence of our chosen basis, A), we have as i^{th} transform coefficient $|c_i| = \mathcal{O}(h^i)$ as $h \to 0$ for $i \in \{0, 1, 2, 3\}$. Due to ZFP's separable basis in higher dimensions, this result generalizes in a straightforward way via tensor products. For example, in three dimensions, $|c_{i,j,k}| = \mathcal{O}\left((\Delta x)^i(\Delta y)^j(\Delta z)^k\right) = \mathcal{O}(h^{i+j+k})$, assuming $\Delta x = \Delta y = \Delta z = h$.

As a result of our ordering of coefficients, for example, by $i + j + k$ in 3D, coefficients will appear sorted by magnitude for small enough h (assuming nonvanishing derivatives of f). This, in turn, implies that each row of the binary coefficient matrix being encoded begins with some number of zeros, with fewer zeros for successive rows. Each such leading sequence of zeros, regardless of length, is encoded using exactly one bit that "terminates" the row (recall that rows are encoded from right to left).

Now consider what happens when we halve h. In d dimensions, this implies a scaling of a coefficient $|c_{i_1,i_2,\ldots,i_d}| = \mathcal{O}(h^{i_1+i_2+\cdots+i_d})$ by $2^{-(i_1+i_2+\cdots+i_d)}$. Consequently, we expect $n_{i_1,i_2,\ldots,i_d} = i_1 + i_2 + \cdots + i_d$ fewer significant bits in its negabinary representation. If coefficients already appear in sorted order, those new $n_{i_1,i_2,\ldots,i_d}$ zero-bits are each absorbed into a single row terminator bit and incur no coding cost, except for the $n_{3,3,\ldots,3} = 3\ d$ bits (corresponding to the leftmost column in the coefficient matrix), since a single leading zero still incurs a one-bit cost. These bits that have been "freed" when h was halved can thus be used to increase the precision by encoding additional bit planes up to the bit budget implied by the prescribed rate.

Let us now determine the number of freed bits, $n(d)$, per d-dimensional block:

$$\begin{aligned} n(d) &= \left(\sum_{i_1=0}^{3}\sum_{i_2=0}^{3}\cdots\sum_{i_d=0}^{3} i_1 + i_2 + \cdots + i_d\right) - 3d \\ &= \frac{3}{2}d4^d - 3d = 3d\left(\frac{4^d}{2} - 1\right). \end{aligned} \tag{9}$$

Amortized over 4^d coefficients in a block, each coefficient consequently gains a precision of $3d(2^{-1} - 4^{-d})$ on average when h is halved, and the compression error therefore decays as $\mathcal{O}(h^{3d(2^{-1}-4^{-d})})$ as $h \to 0$. Below, we demonstrate the agreement of our theory with empirical observations.

4. Numerical results

To illustrate the effectiveness of the ZFP data compression approach for PDE discretization, we focus here on the total error in a variety of PDE-based simulations. The discussion begins by investigating the error in a high-order finite difference approximation of the first derivative, and proceeds to results of solving various PDEs (some using high-order methods) with ZFP and other number formats. To probe the class of physically important problems whose solutions contain singular features, we then investigate a problem of a shock wave in air impacting a cylindrical inclusion of helium. Note that wave and shock propagation problems are challenging tests because the resulting numerical methods have very low dissipation, which would otherwise damp and, thus, help to control errors.

4.1. Experimental setup

We used public implementations of ZFP 1.0.0 Lindstrom and Morrison (2022), Blaz 1.1 Martel (2021), and SoftPosit 0.4.1 Leong (2018). For the serial CPU performance results, we used a MacBook Pro with Apple M2 Max cores and DDR5 RAM, a Linux desktop equipped with Intel Xeon E5-2680 CPUs and DDR4 RAM, and LLNL's Lassen supercomputer, with IBM POWER9 cores and DDR4 RAM. For these experiments, we used ZFP's fixed-rate capability via its compressed-array C++ classes with double-precision arithmetic and default software cache size of $2^d\sqrt{N}$ array elements for a d-dimensional array with N total elements. The CUDA experiments were run on LLNL's Pascal and Lassen systems with NVIDIA P100 and V100 GPUs, respectively, and on NERSC's Perlmutter with A100 GPUs. We used ZFP's default rounding mode, which as previously discussed may slightly bias and inflate compression errors.

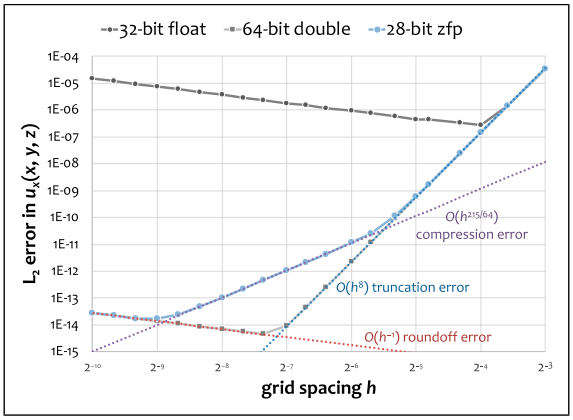

Figure 2. Finite difference error in first partial derivative of $u(x, y, z) = \sin 2\pi x \sin 2\pi y \sin 2\pi z$ as a function of grid spacing, h, for 32-bit IEEE and 28-bit ZFP fixed-rate arrays. The total ZFP error is the sum of truncation error, compression error due to truncating the ZFP stream, and roundoff error in the conversion to IEEE double precision.

4.2. Notation

We use subscripts for partial derivatives, for example, $u_x = \partial u/\partial x$. $\nabla^2 u = u_{xx} + u_{yy}$ denotes the Laplacian (sum of second partial derivatives). As should be clear from context, we also use subscripts to index values on a Cartesian grid and superscripts to denote iteration number, for example, $u^n_{i,j}$ is the value at grid location (i, j) in the n^{th} iteration. For isotropic grids, we use h to denote grid spacing.

4.3. Relative behaviors of errors

Our discussion begins by investigating the fundamental operation of derivative approximation via differences. Regardless of the particular flavor of PDE discretization technique (e.g., FEM, FD, FV), the end effect is to replace a differential operator with a difference operator applied to data. Therefore, consider data given by $u(x, y, z) = \sin 2\pi x \sin 2\pi y \sin 2\pi z$ on a uniform spatial grid $x_j = jh, y_k = kh, z_l = lh$, for $(x, y, z) \in [0, 1)^3$. The x-derivative, $\partial_x u$, is then approximated on the dual grid (i.e., the "half points") using 8th order centered finite differences and the L_2-error is computed over the domain. All arithmetic is done using 80-bit extended precision regardless of storage type. The results are presented in Figure 2. Also included in the figure are results using standard 64-bit doubles and 32-bit floats. The usual analysis with standard floating point axioms indicates that the error will decay as $\sim h^8$, until the condition of the problem, which grows as $\sim h^{-1}$, becomes large enough that roundoff errors dominate ($h \approx 2^{-7}$ for double precision). Reference lines indicating these growth rates are included in the figure, and the observations for standard floating point calculations are exactly in line with this theory. However, for the ZFP result we see a third growth rate corresponding to ZFP-induced "compression error" associated with bit stream truncation (see previous section). The compression error decays as $\sim h^{3d(2^{-1}-4^{-d})}$ in the ZFP representation of u itself, and as $\sim h^{3d(2^{-1}-4^{-d})-1}$ in its first derivative estimate. In 3D ($d = 3$), the compression error in $\partial_x u$ decays as $\sim h^{215/64}$, which is in excellent agreement with the observed error. This is a remarkable result—whereas IEEE roundoff error in derivative estimates *increases* with grid resolution, ZFP compression error *decreases*. Because computations with ZFP currently rely on conversion to an arithmetic type like IEEE, this error reduction is eventually reversed once IEEE double-precision roundoff error dominates ($h \approx 2^{-9}$).

4.4. Example: The Poisson equation

Having established the error behavior of ZFP for a single roundtrip of compression and decompression, we now evaluate how compression errors impact the solution accuracy in iterative solvers and time-dependent PDEs, where such errors may cascade over time. Particularly, we focus on roundoff and compression errors involved in finite-difference computations of first up to fourth derivatives.

We begin by comparing ZFP accuracy with alternative number representations of the iterative solution $u(x, y)$ to the Poisson equation

$$u_{xx} + u_{yy} = f(x, y) = \sqrt{x^2 + y^2} \tag{10}$$

on $[-1, 1]^2$, where we have chosen $f(x, y)$ to be a radially symmetric function that increases linearly by radius. Hence, its contours are uniformly spaced concentric circles. Absent boundary conditions, the solution is

$$u(x,y) = \frac{1}{9}(x^2+y^2)^{\frac{3}{2}}, \tag{11}$$

as can easily be verified. We use this as ground truth and then impose matching Dirichlet boundary conditions. With prescribed boundaries, we then initialize the iterative solver on the interior $(-1,\ 1)^2$ of the domain using transfinite interpolation

$$u^0(x,y) = u(x,1) + u(1,y) - u(1,1). \tag{12}$$

This yields a rough approximation to the solution. To discretize the PDE, we use second-order finite differences

$$u_{xx} \approx \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h^2}, \tag{13a}$$

$$u_{yy} \approx \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{h^2}. \tag{13b}$$

Our approach is to solve for $u_{i,j}$ by fixing u at all other grid points. For simplicity, we use a Jacobi update scheme where we solve for u^{n+1} at each grid point by fixing the values u^n in the previous iteration, n:

$$u^{n+1}_{i,j} = \frac{1}{4}\left(u^n_{i-1,j} + u^n_{i+1,j} + u^n_{i,j-1} + u^n_{i,j+1} - h^2 f(x,y)\right). \tag{14}$$

While such a scheme exhibits very slow convergence, we favor a simple solver to allow for a straightforward comparison between number representations. For example, a more sophisticated multigrid solver would require making numerous decisions about coarsening, smoothing, coarse-grid solver, etc., plus auxiliary numerical data structures.

For comparison with Blaz, which does not expose an API for accessing individual array elements or for evaluating the stencils in Equation (14), we here take the approach of decompressing u^n to a temporary double-precision array, advancing the solution one iteration using double-precision arithmetic, and then compressing the result, u^{n+1}. (As discussed in the introduction, this approach was also taken in Laney et al. (2013), which we will depart from below when evaluating ZFP's compressed-array primitives.) Thus, compression and rounding errors will be introduced by each approximate number representation once per iteration.

4.4.1. Stationary compression. Before evaluating the accuracy of iterates, we first examine how the roundoff errors associated with reduced precision or compression of the ground-truth solution u impact the accuracy not only of u but also the L_2 norm of its gradient, $\|\nabla u\|$, and its Laplacian, $\nabla^2 u$, computed using finite differences. We note that while u is C^2 continuous, it is highly nonlinear and not necessarily easily compressible. Still, we expect u to be fairly well represented in reduced precision, with challenges posed by the increasing condition numbers associated with the first and second differential operators. This is confirmed by Figure 3, which for several representations shows distorted (noisy) contours and, for $\nabla^2 u$, complete loss in accuracy for the 16-bit scalar formats and for Blaz. This failure is simply due to insufficient precision to resolve differences.

To compare with Blaz's 5.625 bits/value, we evaluate ZFP at 5.5 bits/value—the closest ZFP rate supported. Here the ZFP Laplacian exhibits some slight asymmetry. We attribute this to the asymmetry in the negabinary representation (f and $-f$ differ in precision by one bit) and to the necessary linear ordering of coefficients that favors one dimension over the other. A similar but far more pronounced asymmetry is also seen in Blaz.

Figure 4 quantitatively plots as a function of grid size the error in $\nabla^2 u$ estimated using fourth-order accurate differencing. We here expect ZFP compression errors to follow $\mathcal{O}(h^{5/8})$. We generally observe a shallow but slightly positive error slope, whereas roundoff errors for the IEEE and Posit scalar types follow $\mathcal{O}(h^{-2})$. These results clearly demonstrate the benefit of using ZFP for finite differences, with even 8-bit ZFP surpassing both 32-bit floats and Posits in accuracy for fine enough grids (large N).

4.4.2. Iterative solution. Based on the above results, we expect several formats to be unusable for solving this second-order PDE. This is also confirmed by Figure 5, which visualizes approximations to $\nabla^2 u$ after a little more than 2,000 iterations, well before convergence. We note that, as in the static compression case of Figure 3, 12-bit ZFP appears visually identical to full 64-bit double precision and further improves on 32-bit single precision, which again exhibits grainy noise-like artifacts in contours. We note that the grid has here been reduced to 1024×1024 to speed up computations for the many iterations needed.

4.5. Example: The heat equation

Our Jacobi-based iterative approach to solving the Poisson equation bears a strong resemblance to the heat equation,

$$u_t = \nu(u_{xx} + u_{yy}), \tag{15}$$

with explicit time integration. Here time, t, takes the place of iteration number, but with the added complication of having to take small enough time steps to satisfy the CFL stability condition. Using fourth-order accurate derivatives in space and second-order time derivatives, the PDE is discretized as

$$\begin{aligned} \frac{u^{n+1} - u^n}{\Delta t} &= u_t + \frac{\Delta t}{2} u_{tt} + \cdots \\ &= \nu\left(u_{xx} + u_{yy}\right) + \\ &\quad \frac{\Delta t}{2}\nu^2\left(u_{xxxx} + 2u_{xxyy} + u_{yyyy}\right) + \cdots, \end{aligned} \tag{16}$$

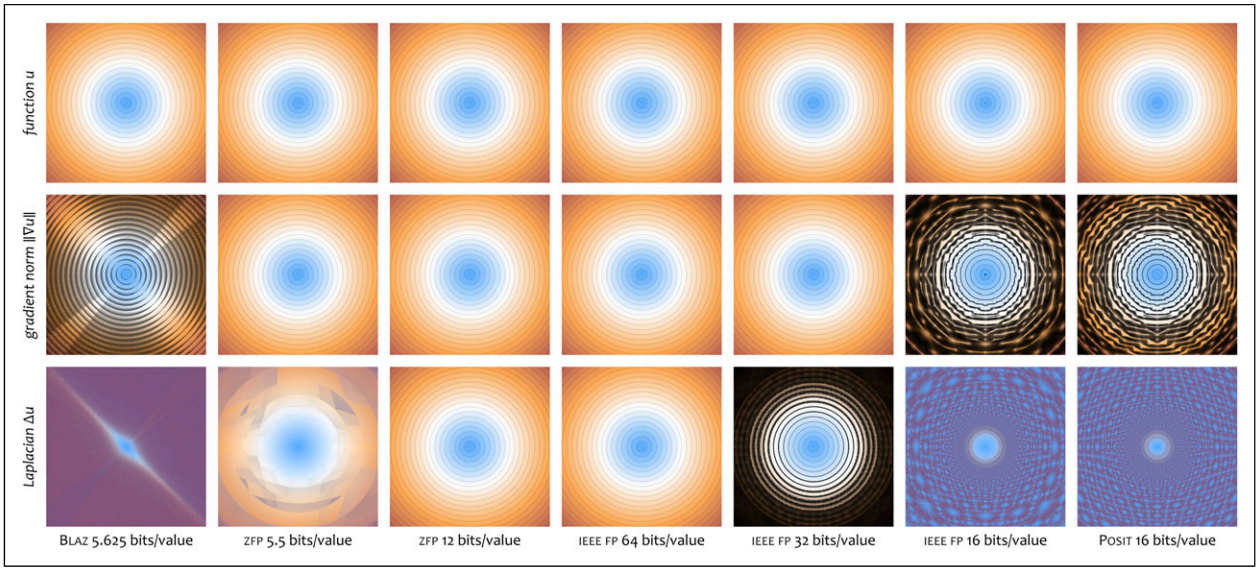

Figure 3. Illustration of the impact of rounding/compression errors on finite-difference derivative estimates of the function $u(x,y) = 1/9(x^2+y^2)^{3/2}$. The plots show $(9u)^{1/3} = \sqrt{3\|\nabla u\|} = \nabla^2 u = \sqrt{x^2+y^2}$ based on second-order finite differences computed for various numerical representations of u on $[-1,1]^2$ with uniform, isotropic grid spacing $h = 2^{-11}$. The increasing condition numbers, $\{\mathcal{O}(1), \mathcal{O}(h^{-1}), \mathcal{O}(h^{-2})\}$ (top to bottom), of the differential operators cause rounding errors to be magnified and reveal the benefits of 12-bit ZFP even over 32-bit single precision. The ideal solution has concentric, evenly spaced circular contours (contours are not shown for the worst approximations).

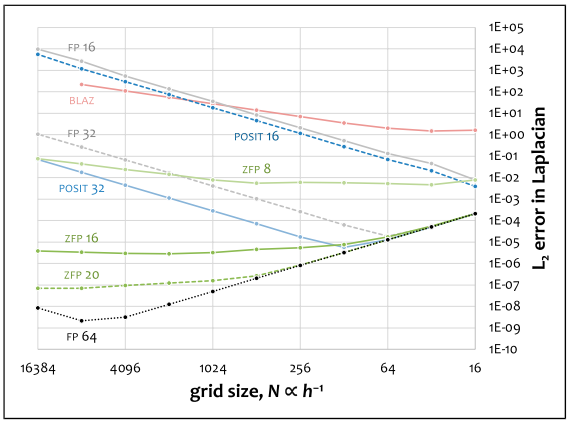

Figure 4. Error in 4th-order Laplacian estimate versus grid size for various number representations.

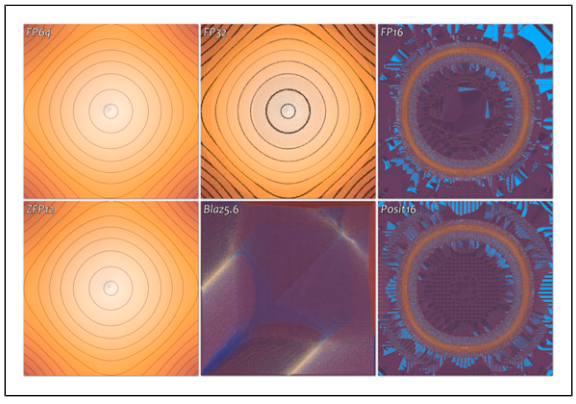

Figure 5. Contour plots of $\nabla^2 u$ corresponding to an intermediate (not-yet-converged) solution to the Poisson equation $\nabla^2 u(\mathbf{r}) = \|\mathbf{r}\|$ based on an iterative Jacobi solver. 12-bit ZFP is qualitatively identical to 64-bit double precision; alternative representations exhibit moderate to severe artifacts.

where we use standard central difference stencils Fornberg (1988). We use Neumann boundary conditions,

$$u_x(-1,y) = u_x(1,y) = u_y(x,-1) = u_y(x,1) = 0, \quad (17)$$

enforced by reflecting u across the boundary, and initial conditions

$$u(x,y,0) = \sin(k_x x)\sin(k_y y) \quad (18)$$

on the domain $\Omega = [-1, 1]^2$. We obtain as analytical solution

$$u(x,y,t) = u(x,y,0)\exp\left(-\nu(k_x^2 + k_y^2)t\right). \quad (19)$$

In other words, the shape of $u(x, y, t)$ does not change; only the amplitude does. Using higher-order finite differencing, roundoff error dominates even for modest grid resolutions. We use 256×256 grid points with 8 ghost layers on each side to accommodate Blaz and ZFP. This gives isotropic grid spacing $h = 1/128$. We set $\nu = 1$, $k_x = k_y = \pi/2$.

As with the Poisson equation, the state u is represented using various number representations and is (de)compressed once per time step. All arithmetic is performed in full double precision, and we also keep the time advanced solution u^{n+1} in double precision before converting it to the storage representation at the end of each iteration.

Figure 6 shows two snapshots of u at early and late times. Even though the increment Δu and u^n are each easily represented in double and even reduced precision, their magnitudes differ greatly. Because the CFL condition requires a small time step $\Delta t < h^2/(4\nu) = 2^{-16}$, the ratio $\Delta u/u < (\pi^2/2)\Delta t < 7 \times 10^{-5}$. In other words, when advancing the solution, the representation requires at least $\log_2(|u|/|\Delta u|)$ bits of mantissa, or about 14 bits, to resolve *any* differences between u^n and u^{n+1}. As 16-bit IEEE and Posits both provide less accuracy, $u^{n+1} = u^n + \Delta u^n$ rounds to u^n, and the solution is never advanced, as is evident from the figure. On the contrary, even at a rate of 5.5 bits/value (slightly less than Blaz), ZFP provides a reasonable solution after as many as 20,000 time steps (bottom panel) and in spite of accumulated roundoff error. Blaz, on the other hand, introduces diagonal drift in the solution early on (perhaps due to truncation of higher-order cross terms), until around 7,600 time steps, when it introduces NaNs. We have no explanation for this behavior, as all computations involved are well-conditioned.

Figure 7 plots the L_2 error in u over time for various representations. Notably, 16-bit ZFP is more accurate than 32-bit IEEE and Posits; 32-bit ZFP coincides with double precision.

Whereas the Poisson equation illustrates challenges in estimating accurate second derivatives using reduced-precision representations, the heat equation demonstrates how lack of precision causes difficulties with time integration. Among the low-precision representations, only ZFP at 8 bits/value handles this second challenge gracefully.

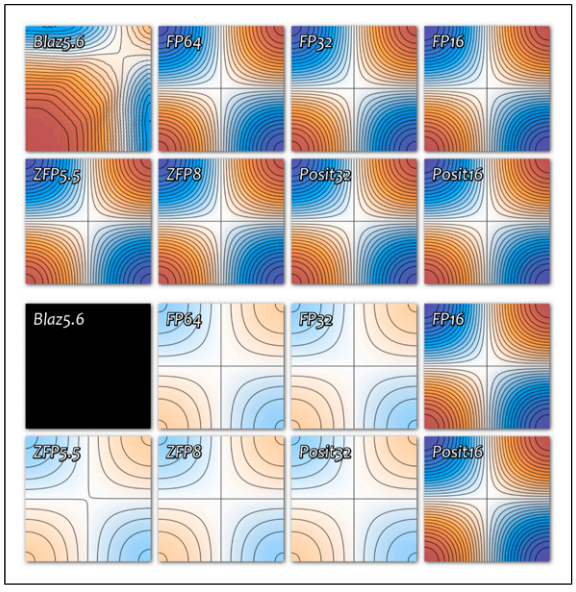

Figure 6. Solutions to the heat equation at early (top) and late (bottom) times. Blaz introduces NaNs early, while 16-bit IEEE and Posits cannot resolve the small temporal changes required for solver stability, effectively "freezing" the solution.

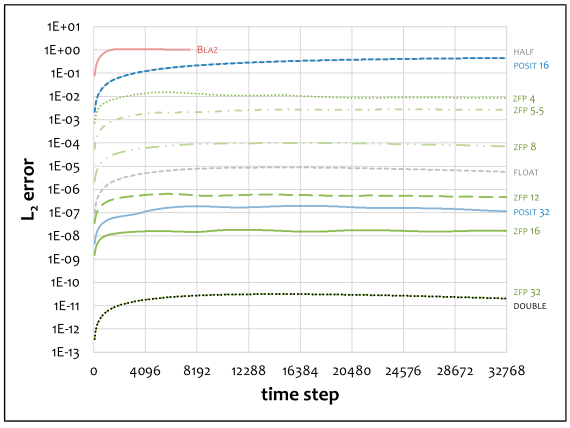

Figure 7. Error in heat equation solution over time with respect to the continuous analytical solution.

4.6. Example: The wave equation

The approach taken so far of decompressing the entire state before advancing and compressing it once per iteration actually *increases* data movement. Not only do we incur that same data movement to and from DRAM as we would when representing the solution uncompressed, but we also have to account for the additional data movement incurred by the (de)compression steps. The ZFP representation was designed specifically to reduce data movement by (de)compressing tiny blocks of data on demand that uncompressed occupy a single or a few hardware cache lines, thereby avoiding subsequent accesses to DRAM. Below, we evaluate ZFP's compressed-array data structures that perform on-demand compression, decompression, and software caching of uncompressed data, with lossy compression occurring only when a previously modified block is evicted from the software cache, for example, due to a cache conflict. For an $N_x \times N_y$ 2D array, we use the default software cache size of $4\sqrt{N_x N_y}$ elements rounded up to the next integer power of two.

We begin our evaluation by solving the wave equation. Here, we use the 6th-order accurate discretization described in Banks and Henshaw (2012) and adopt the surface wave test problem from the same. Specifically consider the wave equation in two space dimensions,

$$u_{tt} = {c_x}^2 u_{xx} + {c_y}^2 u_{yy}, \tag{20a}$$

$$u(x, y, 0) = u^0(x, y), \quad u_t(x, y, 0) = v^0(x, y), \tag{20b}$$

$$u_y(x,0,t) = \alpha u(x,0,t), \tag{20c}$$

where c_x and c_y are real valued wave speeds, and $\alpha \in \mathbb{R}$. This system supports traveling surface wave solutions,

$$u(x,y,t) = Ae^{\alpha y}\cos(kx \pm \omega t), \tag{21}$$

where A is the wave amplitude, and ω satisfies the dispersion relation $\omega = \sqrt{c_x{}^2k^2 - c_y{}^2\alpha^2}$. Note the additional constraint $\int_{-\pi}^{\pi} u(x,0,t)\,dx = 0$ is enforced to eliminate the exponentially growing mode with $k = 0$. As in Banks and Henshaw (2012), the problem is specified by taking $k = 1$, $c_x = 1/2$, $c_y = 1$, $\alpha = 0.4$ and $A = 1$. Computations are performed in double precision using the 6th-order accurate upwind scheme described in Banks and Henshaw (2012) and using a square grid with $(x, y) \in (-\pi, \pi) \times (-2\pi, 0)$ with N points in each direction. The L_2 error norm of the approximate solutions at the final time $t_f = 5$ using various levels of ZFP compression, as well as data stored in standard IEEE half, single, and double precision, are given in Figure 8. Here it is apparent that the error in the half-precision result is essentially dominated by roundoff error already with a grid of 16×16, demonstrating that half-precision is not a viable storage representation in this case. Error in the single-precision result becomes roundoff-dominated near 10^{-7}, a similar level of error to that obtained by 20-bit ZFP. At the finest resolution, double precision gives roughly 14 correct digits, and 40-bit ZFP is nearly identical. Note that, as in the case of the simple derivative operator (results shown in Figure 2), the ZFP results appear to attain an error floor before leveling off, which is in contrast to typical IEEE floating point, where the roundoff error dominates and grows as $\mathcal{O}(h^{-2})$, at least initially. Because the ZFP results here use double precision arithmetic, such growth will be observed for sufficiently small h and will follow the trend inherent to double-precision arithmetic.

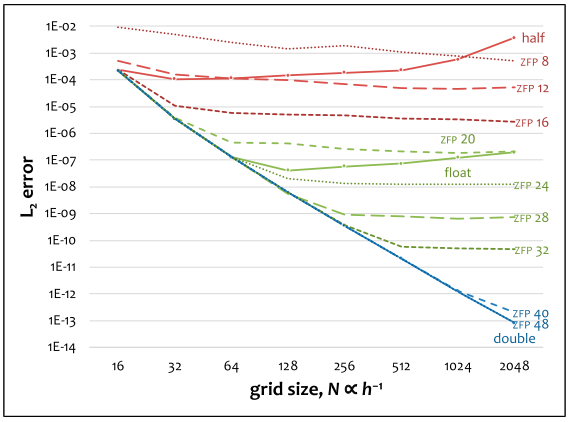

Figure 8. Error in final-time solution to the wave equation.

4.7. Example: Compressible flow

Many PDE applications involve solutions with singular features. For example in conservation laws, solutions with discontinuities (shock waves) often arise from smooth data in finite time. Singular solutions also commonly occur near jumps in material properties, such as along an interface between disparate gasses. The implications of adopting ZFP arrays for such problems is important to understand.

As a more complicated test, consider a Mach-1.22 shock in air impacting a cylindrical inclusion (bubble) of helium gas. This problem was first considered experimentally in Haas and Sturtevant (1987) and in many subsequent numerical studies including Schwendeman (1988); Quirk and Karni (1996); Banks et al. (2007). In the present manuscript, the numerical method and setup from Banks et al. (2007) are adopted. In particular, air is modeled as an ideal gas with ratio of specific heats $\gamma = 1.4$, specific heat at constant volume $C_v = 0.720$, and initial density $\rho_0 = 1$. Helium is modeled as an ideal gas with ratio of specific heats $\gamma = 1.67$, specific heat at constant volume $C_v = 3.11$, and initial density $\rho_0 = 0.138$. The domain is a 2D rectangle $(x, y) \in (0, 0.75) \times (-0.25, 0.25)$. The initial conditions correspond to a planar shock wave in air traveling left to right and initially located at $x = 0.05$ and to a circular inclusion of helium of radius $r_0 = 0.1$ and centered at $(x_c, y_c) = (0.2, 0)$. Runs were made with solution-adaptive time-stepping with a constant CFL number of 0.8.

Similar to Figure 8, in Figure 9, we present the L_1 error norm (L_1 is the preferred error norm for problems with discontinuities) in the solution at the final time $t = 0.40$ (after the shock wave has traversed the bubble) versus the grid size for the shock-bubble interaction problem with results computed in IEEE double but stored in different data types. The error is measured with respect to a solution computed with 80-bit extended precision storage. Five grid resolutions are considered, with N points in y and $1.5\,N$ points in x. Again, one sees that, for the standard IEEE types, domination by a growing roundoff error for sufficiently small h, immediately in the cases of IEEE half and single precision and after $N = 256$ for double precision. For all IEEE types, the growth follows the predicted $\mathcal{O}(h^{-2})$ rate, at least initially. As before, the ZFP curves at first decrease with increasing grid size, but all eventually begin to increase, with the higher-rate ZFP more closely following the double-precision trend. Notably, however, the ZFP results at an equivalent number of bits have orders of magnitude lower error than their IEEE counterparts: roughly four orders for IEEE half relative to the 16-bit ZFP and slightly more than four orders for IEEE single relative to the 32-bit ZFP. Put another way, for a desired level of accuracy, ZFP arrays can provide significant savings: if an accuracy of 10^{-3} was sufficient, 12-bit ZFP is a suitable replacement for both IEEE single and IEEE double, with storage savings of factors of 2.7 and 5.3, respectively, and without any special numerical algorithm considerations, which could be a necessity for computing directly with IEEE single. These

results are consistent with tests on other 2D problems not shown here (i.e., 16-bit ZFP can replace IEEE double with less than a 1% loss in accuracy). In 3D, the savings are potentially even greater.

Figure 10 shows a visualization of the density field for different data representations with $N_y = 1024$ at final time $t = 0.40$. Clearly, the results using 16-bit IEEE half precision and Posits exhibit significant artifacts, and we conclude that 16-bit precision is not sufficient in this case. On the other hand, the 12-bit ZFP results, which are roughly on par with IEEE single precision, are nearly indistinguishable from the double precision calculation, but with more than 5x reduction in storage.

4.8. Performance

We conclude this section with some brief comments on performance. While ZFP has been designed to be highly performant, the reduction in data movement it affords also comes at the expense of additional computations to perform both compression and decompression. In the case of ZFP's compressed-array classes, additional overhead is incurred in the index translations associated with array tiling and cache lookup and management.

Figure 11 plots the serial CPU execution time for the shock wave problem using both 16-bit ZFP and 64-bit IEEE. As expected, execution time increases as $\mathcal{O}(N^3)$ due to finer resolution in x and y and correspondingly smaller time steps to meet the CFL condition, resulting in linear curves in this log-log plot. In spite of (de)compression being done in software, ZFP performance is here only 2–4 times lower than running with conventional floating-point arrays. With the recent development of ZFP FPGA Barrow et al. (2022); Sun and Jun (2019); Habboush et al. (2022); Sun et al. (2020, 2022); Lim and Jun (2022) and ASIC Liu et al. (2023) designs, we expect this performance gap to narrow and possibly reverse with ZFP hardware support.

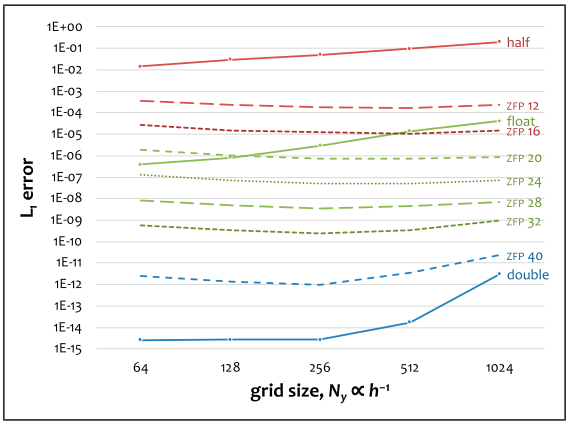

Figure 9. Final-time error for the shock wave problem.

Figure 12 plots, for the same problem, the serial execution time dependence on both rate and grid size, N_y, relative to FP 64 execution time. These runs were done on LLNL's Lassen supercomputer.[3] This figure shows roughly a linear relationship between execution time and rate, with better ZFP performance for the smallest and largest grids. We conjecture that this is due to better software cache reuse for small grids, with $\mathcal{O}(N_y)$ cache size and $\mathcal{O}(N_y^2)$ problem size, and better hardware cache reuse relative to FP 64 when grids are large. Here, 16-bit ZFP is roughly 2.1–2.5 times slower than FP 64.

Whereas keeping up with memory bandwidth is challenging, data movement occurs throughout the memory hierarchy: between CPU and GPU Bamakhrama et al. (2019), between compute nodes using communication Ramesh et al., 2022; Zhou et al., 2021, 2022b, 2022, and between compute nodes and disk in I/O operations Langer et al. (2016); Lindstrom et al. (2016); Triantafyllides et al. (2019); Orf (2019); Margetis et al. (2021); McCallen et al. (2022). Here, ZFP fares more favorably and often dramatically improves overall performance. Moreover, in these situations it is common to compress larger pieces of an array than single blocks at a time, allowing overhead to be amortized over many blocks and data movement to be accelerated using parallel (de)compression.

ZFP's decomposition of arrays into small, independently compressed blocks exposes opportunities for massive data parallelism, as exploited by the ZFP CUDA back-end. Figure 13 plots the ZFP compression and decompression throughput in gigabytes per second on three generations of NVIDIA GPUs. As data source, we used 3D double-

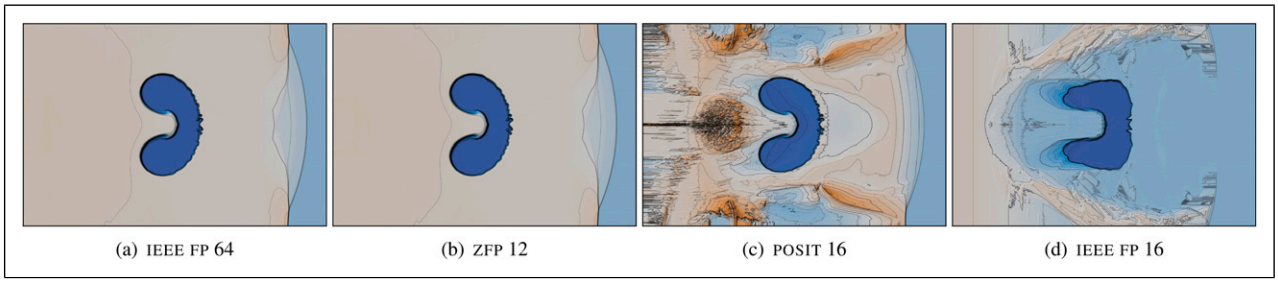

Figure 10. Density contours for the shock wave/bubble interaction at time $t = 0.40$.

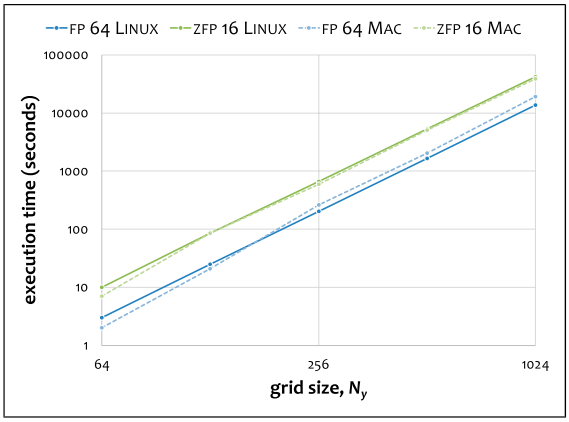

Figure 11. Performance comparison of using uncompressed FP 64 arrays and compressed ZFP 16 arrays for the simulation state in the shock wave problem. In this application, ZFP arrays are 2–4 times slower.

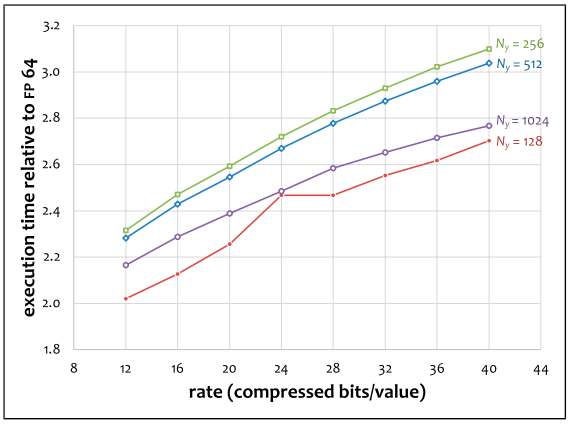

Figure 12. ZFP execution time relative to IEEE double precision versus rate and vertical grid size, N_y, for the shock wave problem.

precision arrays from the Miranda hydrodynamics code available from SDRBench Zhao et al. (2020).[4] Here throughput peaks around 700 GB/s on an A100 GPU. Performance is strongly dictated by bit rate—at lower rates, fewer bits need to be output or processed, and (de)compression time roughly depends linearly on rate.

Because of ZFP's decomposition into independently compressed blocks, distributed-memory parallelism scales as one would expect. That is, no cross-node communication or synchronization is needed for (de)compression itself. On the other hand, other studies have highlighted the benefits of using ZFP to perform node-to-node communication in compressed form Zhou et al. (2021); Ramesh et al. (2022).

At throughputs like these, ZFP becomes an attractive approach to accelerating HPC applications with minimal impact on accuracy, as we have demonstrated here. Compression errors introduced by 16-bit ZFP usually fall

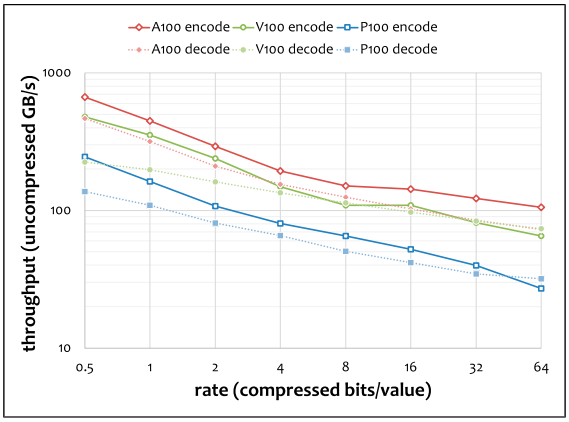

Figure 13. 3D fixed-rate double-precision parallel throughput on three generations of NVIDIA graphics cards. Notice how (de)compression time is roughly proportional to bit rate.

somewhere in between single and double precision roundoff errors, and sometimes greatly improve on single precision. Furthermore, as ZFP provides a near continuous knob on either rate or accuracy, it allows balancing accuracy and performance to meet the application's needs.

Finally, for memory capacity limited computations, ZFP provides an alternative to double precision that enables computations to be completed that otherwise could not due to insufficient memory.

5. Future work and conclusions

Memory usage, both capacity and bandwidth, have become major concerns in HPC, motivating new approaches to the intelligent storage of information. Reduced precision representations are undergoing a renaissance as computational scientists seek to balance information content with the number of bits used. Here, we have demonstrated that an approach based on lossy compressed floating-point arrays is a viable strategy worthy of further consideration by the HPC community. A drop-in compressed array representation has obvious advantages over mixed or lower precision approaches, both in terms of ease of implementation and, if computing in double precision, no need for the specialized solvers robust to roundoff accumulation and overflow that are a necessity for single (or lower) precision calculation. In addition, the ZFP compression error in derivative estimates actually decreases with increasing number of degrees of freedom (i.e., larger condition numbers) as opposed to the standard IEEE floating-point behavior, where roundoff error in derivative estimates increases.

We discussed and added to the growing theoretical and empirical evidence supporting the use of compressed data arrays in certain classes of HPC simulation. Recent theoretical results provide a solid foundation that the errors of lossy compression, like their cousins from traditional finite-

precision arithmetic, can be controlled to ensure stability, accuracy, and effectively unbiased approximation. Experiments with simulations based on PDEs demonstrate a factor of four reduction in memory requirements over IEEE double for less than a 1% change in the L_1 norm of the computed results, which is comparable to the accuracy of single precision. We note that, as with IEEE floating-point precision, choosing a suitable ZFP rate for a given problem is mostly an exercise in trial and error, although the error bounds and convergence theory presented here may inform such decisions.

It is important to identify areas where an approach based on ZFP arrays will likely not benefit. ZFP obtains its highest compression, like most compressors, when data are locally well-correlated (similar or "smooth"). Particle data, for instance, only has such locality if ordered in high-dimensional phase space, which would defeat a major advantage of particle-based methods; thus, ZFP arrays provide little gain for these problems. Depending on how data is organized, unstructured data, for example, from a tetrahedral mesh or arrays of structures, are generally not ordered in such a way as to provide the local correlations needed for high compression. Clever refactoring can reduce this concern. Problems with extreme nonlinearities and data ranges may also present a challenge for a compressed data type. One may object to lossy compression used to represent chaotic solutions like turbulence, but the compression errors are no more serious than the other discretization errors, and one should never focus on a particular instantiation of a chaotic process; only the statistics are valuable quantities of interest for these problems, and the point of eliminating bias in ZFP is to ensure that we can compute correct statistics for the system. Even in problems where extremely high accuracy is required and the truncation error may be driven down to the level of roundoff error, ZFP arrays still provide an advantage because ZFP's wider mantissas actually improve accuracy for the same storage.

Fixed-rate ZFP is but one compression strategy, and there are other options to consider for future work. In particular, a promising direction would be to use variable-rate arrays as the compression strategy. In this case, only as many bits to achieve a desired level of accuracy within each block would be used, so each compressed block would have a different compression rate. Initial results indicate that this choice of compression strategy is even more efficient than fixed-rate ZFP because it "locally adapts" the compression rate to the data, and overall compression factors above four in 2D can be achieved while maintaining less than 1% compression error in the solution.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 and was supported by the Exascale Computing Project [17-SC-20-SC], the LLNL LDRD Program [17-SI-004], and the Office of Science, Office of Advanced Scientific Computing Research.

ORCID iDs

Peter Lindstrom https://orcid.org/0000-0003-3817-4199
Jeffrey Hittinger https://orcid.org/0000-0002-8229-6516
James Diffenderfer https://orcid.org/0009-0004-8641-3275
Alyson Fox https://orcid.org/0000-0001-7140-3614
Daniel Osei-Kuffuor https://orcid.org/0000-0002-6111-6205
Jeffrey Banks https://orcid.org/0000-0001-6413-5113

Notes

1. Negative connotations associated with "lossy" compression are undeserved; IEEE floating-point is itself lossy, rounding away mantissa that it cannot represent. Roundoff error is literally the error associated with lossy finite-precision arithmetic, demonstrating that lossy errors can be managed.
2. Such loss also occurs in usual floating-point arithmetic such as addition.
3. Some of the $N_y = 1024$ runs exceeded Lassen's 12-h job time limit. Thus, we timed only the first 75% of the simulation at this grid size.
4. https://sdrbench.github.io

References

Abdelfattah A, Tomov S and Dongarra J (2020) Investigating the benefit of FP16-enabled mixed-precision solvers for symmetric positive definite matrices using GPUs. *Computational Science (ICCS)*. Berlin: Springer, 237–250. DOI: 10.1007/978-3-030-50417-5_18.

Abdelfattah A, Anzt H, Boman EG, et al. (2021) A survey of numerical linear algebra methods utilizing mixed-precision arithmetic. *International Journal of High Performance Computing Applications* 35(4): 344–369. DOI: 10.1177/10943420211003313.

Ainsworth M, Tugluk O, Whitney B, et al. (2019) Multilevel techniques for compression and reduction of scientific data—the multivariate case. *SIAM Journal on Scientific Computing* 41(2): A1278–A1303. DOI: 10.1137/18M1166651.

Ballester-Ripoll R, Lindstrom P and Pajarola R (2020) TTHRESH: tensor compression for multidimensional visual data. *IEEE Transactions on Visualization and Computer Graphics* 26(9): 2891–2903. DOI: 10.1109/TVCG.2019.2904063.

Bamakhrama MA, Arrizabalaga A, Overman F, et al. (2019) GPU acceleration of real-time control loops. ArXiv Preprint arXiv: 1902.08018.

Banks JW and Henshaw WD (2012) Upwind schemes for the wave equation in second-order form. *Journal of Computational Physics* 231(17): 5854–5889. DOI: 10.1016/j.jcp.2012.05.012.

Banks JW, Schwendeman DW, Kapila AK, et al. (2007) A high-resolution Godunov method for compressible multi-material flow on overlapping grids. *Journal of Computational Physics* 223(1): 262–297. DOI: 10.1016/j.jcp.2006.09.014.

Barnard RW, Dahlquist G, Pearce K, et al. (1998) Gram polynomials and the Kummer function. *Journal of Approximation Theory* 94(1): 128–143. DOI: 10.1006/jath.1998.3181.

Barrow M, Wu Z, Lloyd S, et al. (2022) ZHW: a numerical CODEC for big data scientific computation. In 2022 International Conference on Field-Programmable Technology (ICFPT), Hong Kong, 05-09 December 2022, 1–9. DOI: 10.1109/ICFPT56656.2022.9974258.

Bohr M (2007) A 30 year retrospective on Dennard's MOSFET scaling paper. *IEEE Solid-State Circuits Society Newsletter* 12(1): 11–13. DOI: 10.1109/N-SSC.2007.4785534.

Borkar S and Chien AA (2011) The future of microprocessors. *Communications of the ACM* 54(5): 67–77. DOI: 10.1145/1941487.1941507.

Buoncristiani N, Shah S, Donofrio D, et al. (2020) Evaluating the numerical stability of Posit arithmetic. IEEE International Parallel and Distributed Processing Symposium (IPDPS), New Orleans, LA, USA, 18-22 May 2020, 612–621. DOI: 10.1109/IPDPS47924.2020.00069.

Burgess N, Milanovic J, Stephens N, et al. (2019) Bfloat16 processing for neural networks. 26th IEEE Symposium on Computer Arithmetic (ARITH), Kyoto, Japan, 10-12 June 2019, 88–91. DOI: 10.1109/ARITH.2019.00022.

Burtscher M and Ratanaworabhan P (2007) High throughput compression of double-precision floating-point data. Data Compression Conference, Snowbird, UT, USA, 27-29 March 2007, 293–302. DOI: 10.1109/DCC.2007.44.

Daubechies I and Sweldens W (1998) Factoring wavelet transforms into lifting steps. *Journal of Fourier Analysis and Applications* 4(3): 247–269. DOI: 10.1007/BF02476026.

de Dinechin F, Forget L, Muller JM, et al. (2019) Posits: the good, the bad and the ugly. Conference for Next Generation Arithmetic, Singapore, March 13-14, 2019, 1–10. DOI: 10.1145/3316279.3316285.

Diffenderfer JD, Fox AL, Hittinger JAF, et al. (2019) Error analysis of ZFP compression for floating-point data. *SIAM Journal on Scientific Computing* 41(3): A1867–A1898. DOI: 10.1137/18M1168832.

Exascale Mathematics Working Group (2014) Applied mathematics research for exascale computing: U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research Program. Technical Report LLNL-TR-651000. DOI: 10.2172/1149042.

Fornberg B (1988) Generation of finite difference formulas on arbitrarily spaced grids. *Mathematics of Computation* 51: 699–706. DOI: 10.1090/S0025-5718-1988-0935077-0.

Fox A and Lindstrom P (2024) Statistical analysis of ZFP: understanding bias. Lawrence Livermore National Laboratory. Technical Report LLNL-JRNL-858256. DOI: 10.48550/arXiv.2407.01826.

Fox AL, Diffenderfer JD, Hittinger JAF, et al. (2020) Stability analysis of inline ZFP compression for floating-point data in iterative methods. *SIAM Journal on Scientific Computing* 42(5): A2701–A2730. DOI: 10.1137/19M126904X.

Fu H, He C, Chen B, et al. (2017) 18.9-Pflops nonlinear earthquake simulation on Sunway TaihuLight: enabling depiction of 18-Hz and 8-meter scenarios. *International Conference for High Performance Computing, Networking, Storage and Analysis*, Denver, CO, USA, November 12-172, 2017, :1–12. DOI: 10.1145/3126908.3126910.

Gustafson JL and Yonemoto IT (2017) Beating floating point at its own game: posit arithmetic. *Supercomputing Frontiers and Innovations* 4(2): 71–86.

Haas JF and Sturtevant B (1987) Interaction of weak shock waves with cylindrical and spherical gas inhomogeneities. *Journal of Fluid Mechanics* 181: 41–76. DOI: 10.1017/S0022112087002003.

Habboush M, El-Maleh AH, Elrabaa ME, et al. (2022) DE-ZFP: an FPGA implementation of a modified ZFP compression/decompression algorithm. *Microprocessors and Microsystems* 90: 104453. DOI: 10.1016/j.micpro.2022.104453.

Haidar A, Wu P, Tomov S, et al. (2017) Investigating half precision arithmetic to accelerate dense linear system solvers. *8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA)*, Denver, CO, USA, November 12-17, 2017, 1–8. DOI: 10.1145/3148226.3148237.

Haidar A, Bayraktar H, Tomov S, et al. (2020) Mixed-precision iterative refinement using tensor cores on GPUs to accelerate solution of linear systems. *Proceedings of the Royal Society A* 476(2243): 20200110. DOI: 10.1098/rspa.2020.0110.

Hammerling DM, Baker AH, Pinard A, et al. (2019) A collaborative effort to improve lossy compression methods for climate data. IEEE/ACM 5th International Workshop on Data Analysis and Reduction for Big Scientific Data (DRBSD-5), Denver, CO, USA, 17-17 November 2019, 16–22. DOI: 10.1109/DRBSD-549595.2019.00008.

Hoang D, Summa B, Bhatia H, et al. (2021) Efficient and flexible hierarchical data layouts for a unified encoding of scalar field precision and resolution. *IEEE Transactions on Visualization and Computer Graphics* 27(2): 603–613. DOI: 10.1109/TVCG.2020.3030381.

Kalamkar D, Mudigere D, Mellempudi N, et al. (2019) A study of BFLOAT16 for deep learning training. ArXiv Preprint arXiv: 1905.12322.

Lakshminarasimhan S, Shah N, Ethier S, et al. (2011) Compressing the incompressible with ISABELA: in-situ reduction of

spatio-temporal data. *Euro-Par Parallel Processing*. Berlin: Springer, 366–379. DOI: 10.1007/978-3-642-23400-2_34.

Laney D, Langer S, Weber C, et al. (2013) Assessing the effects of data compression in simulations using physically motivated metrics. IEEE/ACM International Conference for High Performance Computing, Networking, Storage and Analysis (SC), Denver, CO, USA, 17-22 November 2013, 76:1–12. DOI: 10.1145/2503210.2503283.

Langer SH, Spears B, Peterson JL, et al. (2016) A HYDRA UQ workflow for NIF ignition experiments. Second Workshop on in Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV), Salt Lake City, UT, USA, 13-13 November 2016, 1–6. DOI: 10.1109/ISAV.2016.006.

Leong C (2018) SoftPosit version 0.4.1. https://gitlab.com/cerlane/SoftPosit.

Li S, Lindstrom P and Clyne J (2023) Lossy scientific data compression with SPERR. IEEE International Parallel and Distributed Processing Symposium (IPDPS), St. Petersburg, FL, USA, 15-19 May 2023, 1007–1017. DOI: 10.1109/IPDPS54959.2023.00104.

Liang X, Di S, Tao D, et al. (2018) Error-controlled lossy compression optimized for high compression ratios of scientific datasets. IEEE International Conference on Big Data, Seattle, WA, USA, 10-13 December 2018, 438–447. DOI: 10.1109/BigData.2018.8622520.

Lim SM and Jun SW (2022) Mobilenets can be lossily compressed: neural network compression for embedded accelerators. *Electronics* 11(6): 858. DOI: 10.3390/electronics11060858.

Lindstrom P (2014) Fixed-rate compressed floating-point arrays. *IEEE Transactions on Visualization and Computer Graphics* 20(12): 2674–2683. DOI: 10.1109/TVCG.2014.2346458.

Lindstrom P (2019) Universal coding of the reals using bisection. *Conference for Next Generation Arithmetic (CoNGA))*, Singapore, March 13-14, 2019, 1–10. DOI: 10.1145/3316279.3316286.

Lindstrom P and Isenburg M (2006) Fast and efficient compression of floating-point data. *IEEE Transactions on Visualization and Computer Graphics* 12(5): 1245–1250. DOI: 10.1109/TVCG.2006.143.

Lindstrom P and Morrison G (2022) ZFP version 1.0.0. https://github.com/LLNL/zfp.

Lindstrom P, Chen P and Lee EJ (2016) Reducing disk storage of full-3D seismic waveform tomography (F3DT) through lossy online compression. *Computers & Geosciences* 93: 45–54. DOI: 10.1016/j.cageo.2016.04.009.

Lindstrom P, Lloyd S and Hittinger J (2018) Universal coding of the reals: alternatives to IEEE floating point. *Conference for Next Generation Arithmetic (CoNGA)*, Singapore, March 28, 2018, 1–14. DOI: 10.1145/3190339.3190344.

Liu X, Gonzalez-Guerrero P, Peng I, et al. (2023) Accelerator integration in a tile-based SoC: lessons learned with a hardware floating point compression engine. Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis, Denver, CO, USA, November 12-17, 2023, 1662–1669. DOI: 10.1145/3624062.3624245.

Margetis AS, Papoutsis-Kiachagias E and Giannakoglou K (2021) Lossy compression techniques supporting unsteady adjoint on 2D/3D unstructured grids. *Computer Methods in Applied Mechanics and Engineering* 387: 114152. DOI: 10.1016/j.cma.2021.114152.

Markidis S, Chien SWD, Laure E, et al. (2018) NVIDIA tensor core programmability, performance & precision. IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Vancouver, BC, Canada, May 21-25, 2018, 522–531. DOI: 10.1109/IPDPSW.2018.00091.

Martel M (2021) Blaz version 1.1. https://github.com/mmartel66/blaz.

Martel M (2022) Compressed matrix computations. IEEE/ACM International Conference on Big Data Computing, Applications and Technologies (BDCAT)Vancouver WAUSA, *December* 6-9, 2022, 68–76. DOI: 10.1109/BDCAT56447.2022.00016.

McCallen D, Tang H, Wu S, et al. (2022) Coupling of regional geophysics and local soil-structure models in the EQSIM fault-to-structure earthquake simulation framework. *International Journal of High Performance Computing Applications* 36(1): 78–92. DOI: 10.1177/10943420211019118.

Michelogiannakis G, Williams A, Williams S, et al. (2014) Collective memory transfers for multi-core chips. 28th ACM International Conference on Supercomputing, Munich, Germany, June 10-13, 2014, 343–352. DOI: 10.1145/2597652.2597654.

Miles A, jakirkham and Bussonnier M, et al. (2024) *zarr-developers/zarr-python: v3.0.0-alpha*. (June 12, 2024). DOI: 10.5281/zenodo.11592827.

Ning P and Hesselink L (1992) Vector quantization for volume rendering. *ACM Workshop on Volume Visualization (VVS)*, Boston, MA, USA, October 19-20, 1992, 69–74. DOI: 10.1145/147130.147152.

Orf L (2019) A violently tornadic supercell thunderstorm simulation spanning a quarter-trillion grid volumes: computational challenges, I/O framework, and visualizations of tornadogenesis. *Atmosphere* 10(10): 578. DOI: 10.3390/atmos10100578.

Quirk JJ and Karni S (1996) On the dynamics of a shock-bubble interaction. *Journal of Fluid Mechanics* 318: 129–163. DOI: 10.1017/S0022112096007069.

Ramesh B, Zhou Q, Shafi A, et al. (2022) Designing efficient pipelined communication schemes using compression in MPI libraries. 29th IEEE International Conference on High Performance Computing, Data, and Analytics (HiPC), Bengaluru, India, 18-21 December 2022, 95–99. DOI: 10.1109/HiPC56025.2022.00024.

Schneider J and Westermann R (2003) Compression domain volume rendering. IEEE Visualization, Seattle, WA, USA,

19-24 October 2003, 293–300. DOI: 10.1109/VISUAL.2003.1250385.

Schwendeman DW (1988) Numerical shock propagation in non-uniform media. *Journal of Fluid Mechanics* 188: 383–410. DOI: 10.1017/S0022112088000771.

Shalf J, Dosanjh S and Morrison J (2011) Exascale computing technology challenges. High Performance Computing for Computational Science (VECPAR 2010), Berkeley, CA, USA, June 22-25, 2010, 1–25. DOI: 10.1007/978-3-642-19328-6_1.

Sun G and Jun SW (2019) ZFP-V: hardware-optimized lossy floating point compression. IEEE International Conference on Field-Programmable Technology (ICFPT), Tianjin, China, 09-13 December 2019, 117–125. DOI: 10.1109/ICFPT47387.2019.00022.

Sun G, Kang S and Jun SW (2020) BurstZ: a bandwidth-efficient scientific computing accelerator platform for large-scale data 34th. ACM International Conference on Supercomputing, Barcelona, Spain, June 29-July 2, 2020, 1–12. DOI: 10.1145/3392717.3392746.

Sun G, Kang S and Jun SW (2022) BurstZ+: eliminating the communication bottleneck of scientific computing accelerators via accelerated compression. *ACM Transactions on Reconfigurable Technology and Systems* 15(2): 1–34. DOI: 10.1145/3476831.

The ASCAC Subcommittee for the Top Ten Exascale Research Challenges (2014) Top ten exascale research challenges. Technical Report 1222713. U.S. Department of Energy Advanced Scientific Computing Advisory Committee. DOI: 10.2172/1222713.

Trefethen LN and Bau D (1997) *Numerical Linear Algebra*. Philadelphia: SIAM. DOI: 10.1137/1.9781611977165.

Triantafyllides P, Reza T and Calhoun JC (2019) Analyzing the impact of lossy compressor variability on checkpointing scientific simulations. IEEE International Conference on Cluster Computing (CLUSTER), Albuquerque, NM, USA, 23-26 September 2019, 1–5. DOI: 10.1109/CLUSTER.2019.8891052.

Trojak W and Witherden FD (2021) Inline vector compression for computational physics. *Computer Physics Communications* 258: 107562. DOI: 10.1016/j.cpc.2020.107562.

Wallace GK (1992) The JPEG still picture compression standard. *IEEE Transactions on Consumer Electronics* 38(1): 18–34. DOI: 10.1109/30.125072.

Wegener A (2013) Universal numerical encoder and profiler reduces computing's memory wall with software, FPGA, and SoC implementations. IEEE Data Compression Conference, Snowbird, UT, USA, 20-22 March 2013, 528. DOI: 10.1109/DCC.2013.107.

Williams S, Waterman A and Patterson D (2009) Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM* 52(4): 65–76. DOI: 10.1145/1498765.1498785.

Yang A, Mukka H, Hesaaraki F, et al. (2015) MPC: a massively parallel compression algorithm for scientific data. IEEE International Conference on Cluster Computing, Chicago, IL, USA, 08-11 September 2015, 381–389. DOI: 10.1109/CLUSTER.2015.59.

Zhao K, Di S, Lian X, et al. (2020) SDRBench: scientific data reduction benchmark for lossy compressors. IEEE International Conference on Big Data, Atlanta, GA, USA, 10-13 December 2020, 2716–2724. DOI: 10.1109/BigData50022.2020.9378449.

Zhou Q, Chu C, Kumar NS, et al. (2021) Designing high-performance MPI libraries with on-the-fly compression for modern GPU clusters. IEEE International Parallel and Distributed Processing Symposium (IPDPS), Portland, OR, USA, 17-21 May 2021, 444–453. DOI: 10.1109/IPDPS49936.2021.00053.

Zhou Q, Anthony Q, Shafi A, et al. (2022a) Accelerating broadcast communication with GPU compression for deep learning workloads. 29th IEEE International Conference on High Performance Computing, Data, and Analytics (HiPC), Bengaluru, India, 18-21 December 2022, 22–31. DOI: 10.1109/HiPC56025.2022.00016.

Zhou Q, Kousha P, Anthony Q, et al. (2022b) Accelerating MPI all-to-all communication with online compression on modern GPU clusters. High Performance Computing. Berlin: Springer, 3–25. DOI: 10.1007/978-3-031-07312-0_1.

Peter Lindstrom is a computer scientist in the Center for Applied Scientific Computing at Lawrence Livermore National Laboratory. His research focuses on data compression, scientific visualization, and high-performance computing. Peter earned a PhD in Computer Science from Georgia Institute of Technology in 2000 and holds B.S. degrees in Computer Science, Mathematics, and Physics from Elon University. He is the chief architect of the R&D 100 award-winning ZFP compressor and led the development of ZFP as part of the DOE Exascale Computing Project.

Jeffrey Hittinger obtained the PhD degree in Aerospace Engineering and Scientific Computing from the University of Michigan in 2000. He currently is the Division Director for the Center for Applied Scientific Computing at Lawrence Livermore National Laboratory. His research interests include high-order methods for partial differential equations, adaptive mesh refinement, a posteriori error estimation, and the computational modeling of plasmas for fusion energy applications.

James Diffenderfer graduated from the University of Florida in 2020 with a PhD degree in Mathematics and a M.S. degree in Computer Science. He is a Computational Scientist in the Center for Applied Scientific Computing at Lawrence Livermore National Laboratory. His research

interests include numerical analysis, neural network robustness and compression, and scientific machine learning.

Alyson Fox is a computational mathematician at the Center for Applied Scientific Computing at Lawrence Livermore National Laboratory. She obtained her M.S. and PhD in Applied Mathematics from the University of Colorado Boulder in 2015 and 2017, respectively. Her research interests include floating-point error analysis, lossy compression algorithms, and numerical linear algebra.

Daniel Osei-Kuffuor is a computational scientist in the Center for Applied Scientific Computing (CASC) at the Lawrence Livermore National Laboratory (LLNL). Prior to joining LLNL, Daniel earned a PhD in Scientific Computing and Mathematics from the University of Minnesota in 2011. He also holds an M.Sc. in Computational Science and Engineering from the University of Greenwich, London. His research interests include linear algebra and sparse matrix computations, scalable solvers and preconditioners, numerical analysis and variable precision computing, scalable algorithms for electronic structure calculations, HPC, and performance portable scientific Software design.

Jeffrey Banks received his PhD in applied mathematics from Rensselaer Polytechnic Institute in 2006. Subsequently he completed postdoctoral appointments at Sandia National Laboratories in Albuquerque, New Mexico, and Lawrence Livermore National Laboratory in Livermore, California. In 2010 he was appointed as a staff scientist at LLNL where he remained until moving back to RPI. In 2015 he was appointed the Eliza Ricketts Foundation Career Development Chair in the Department of Mathematical Sciences at RPI, and was promoted to full professor in 2023.