

Synthetic Data Generation for Evaluating Parallel I/O Compression Performance and Scalability

Sean B. Ziegeler
DoD HPCMP PETTT / Engility
Stennis, MS, USA
sean.ziegeler@engility.com

Christopher P. Stone
DoD HPCMP PETTT / Engility
Athens, GA, USA
chris.stone.ctr@engility.com

Abstract—Compression is one of the most common forms of data reduction and is typically the least invasive. As compute capability continues to outpace I/O bandwidths, compression becomes that much more attractive. This paper explores the scalable performance of parallel compression and presents an in-depth analysis of a coherent noise algorithm to generate synthetic data that can be used to easily evaluate parallel compression. The algorithm favors simplicity, ease-of-use, and scalability over high-fidelity reconstruction of real data, so we go to lengths to show that the synthetic data generated is suitable as a proxy for evaluating compression, especially in benchmarks and mini-apps.

Keywords—I/O, compression, data systems, data reduction, high performance computing

I. INTRODUCTION

I/O is known to be one of the worst bottlenecks in HPC. The substitution of I/O for faster, more scalable computation that reduces the storage footprint has the potential to improve performance significantly [1]. At exascale, it is possible that pure computation becomes significantly cheaper than data movement, so compression may become quite common. As such, I/O benchmarks and mini-apps need to generate data that is quantifiably compressible.

Mini-apps can provide a valuable analysis of compression with I/O performance. First, they allow easy reconfiguration and addition of new I/O methods and routines. Second, they impose more precise control over the conditions under which I/O occurs, allowing for more reliable comparison of results. A collection of new mini-apps, *MiniIO*, can serve this purpose [1,2]. Numerous tools already exist to benchmark and measure I/O. A brief review of these is provided in [1], but they range from basic throughput measurement tools like IOR [3] to I/O kernels extracted from actual simulation codes. ADIOS introduced a more abstracted approach in their “skel” tool, which reproduces only I/O code based upon the utilization of ADIOS in a larger code [4]. The most sophisticated tool of which we are aware is “MACSio,” which models a variety of potential simulation data structure designs and provides a modular system to add I/O routines [5,6]. MACSio appears to have been conceived at nearly the same time as our approach.

Unfortunately, most I/O benchmarks and mini-apps of which we are aware do not thoroughly account for compression. Intel MPI Benchmarks (IMB) [7], HPIO [8], and ADIOS skel [4] fill with data arrays with a constant (usually 0), which is trivially compressible. Other benchmarks such as b_eff_io [9], netCDF-

Bench [10], EPCC BenchIO [11], and HACC I/O [12] fill arrays with a sequence; an improvement but this still does not approximate a scientific data set. IOR provides an option to fill arrays with incompressible noise or a combination of indices, MPI rank, and time stamp [3]. NPB BT-IO [13] and MADbench [14] produce pseudo-data arrays based on the mathematics and physics for which the applications serve as proxy. Finally, MACSio and MiniIO use coherent noise, which can be adjusted to be more or less “compressible.” [2,6].

II. MINIIO AND SYNTHETIC DATA

The Department of Defense (DoD) High Performance Computing Modernization Program (HPCMP) User Productivity Enhancement, Technology Transfer, and Training (PETTT) program developed a series of I/O mini-apps: MiniIO. It currently consists of four mini-apps that represent several major simulation types.

A. Mini-apps Overview

MiniIO apps are currently based on MPI. Hybrid parallelism is possible but currently not necessary, since one could run fewer ranks per node to simulate threads. The mini-apps iterate over a specified number of time steps. The four mini-apps are as follows:

- *struct* – simulates an application with a static, structured 3D grid, a 2D tiling of that grid across ranks, time-variant data, blanked data values and load balancing. Struct is used in this paper for scaling studies.
- *unstruct* – a static or time-variant 3D unstructured grid, a 3D decomposition, and time-variant data values.
- *amr* – an adaptive mesh refinement (AMR) grid, 3D decomposition, with time-variant grid and data values.
- *cartiso* – starts with a Cartesian grid, 3D decomposition, and extracts isosurface geometry for output. The data values are set by a function that allow one to control the load balancing and amount of isosurface geometry data [1]. It is used in this paper to study the spectral properties of the coherent noise algorithm.

B. Synthetic Data Generation with Coherent Noise

With compression, one must take care how the data structures are filled. A constant value results in trivially small compressed data. Filling with pseudo-random data is also unfair, as the high apparent entropy renders it nearly incompressible. Actual simulation data will be somewhere in between, with

some amount of spatial coherence amenable to compression, depending upon the type of simulation.

MACSio solves this problem by using Perlin noise [15]. Ken Perlin later improved upon his eponymous noise significantly with simplex noise [16] especially in higher dimensions, but also patented that method. Since some recent compression algorithms are capable of 4D compression (e.g., 3D space plus time), we thought it important to generate temporally coherent noise. We chose an open variant made available via the public domain OpenSimplex Noise C library [17].

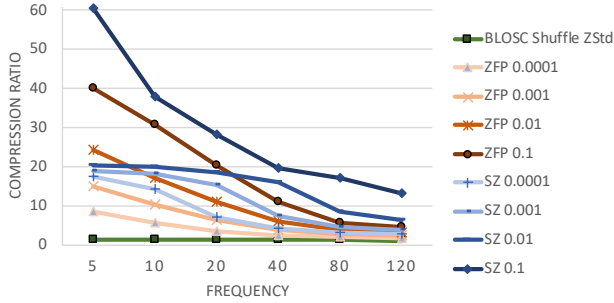


Fig. 1. Several compression results for various frequencies of coherent noise.

OpenSimplex noise (OSN) is composed of a set of basis points with pseudorandom vectors and a high-order falloff function to interpolate between the basis points. As such, generating basis points at a low frequency creates data with less apparent entropy, which would compress better. Higher frequencies should contain more random values and compress less. Using a high-order interpolator should mean that linear and quadratic predictor-based compression algorithms do not trivially compress interpolated regions. The noise is generated in 4D to simulate time step changes, which accommodates temporal compression.

To provide initial evidence that OpenSimplex noise is indeed sufficient as synthetic data, compression algorithms were applied to OpenSimplex noise at various sampling frequencies. In Fig. 1, increasing the frequency reduces the compressibility as measured by the compression ratio (C_R) (i.e., the ratio of compressed and original dataset size). The compression methods used in Fig. 1 include: (a) *Blosc* [18] Shuffle+ZStd – applies bit-shuffling then ZStd lossless compression [19,20]; (b) *ZFP* – small block transform with embedded coding and error-bounded lossy compression [21]; (c) *SZ* – a best-fit predictor-based method with very high throughput and error-bounded lossy compression [22,23]. All of the above are provided in ADIOS. ZFP and SZ are designed for floating point data. Since they offer the same type of absolute error-bound, a range of error bounds (0.0001, 0.001, 0.01, 0.1) are used for each. Blosc offers several options besides ZStd, but ZStd has been shown to provide the best lossless compression ratio in general. Another algorithm, *gzip*, an implementation of the extended-Rice lossless algorithm [24], was also attempted but it increased the data size at higher frequencies. This caused ADIOS to overflow buffers, so its results were excluded in favor of Blosc.

The implementation of OSN is completely concurrent and therefore embarrassingly parallel, requiring no communication to remain consistent across parallel decompositions. It also requires minimal memory to maintain state. This makes it a good

choice as a synthetic data generator for high performance computing (HPC) at very large scales. Because OSN is based on sums of scaled random frequencies, it can theoretically approximate the spectral behavior of real data sets that contain similar frequencies and value ranges. OSN's interpolation also facilitates easy adaptation to unstructured, AMR, and other irregular grids.

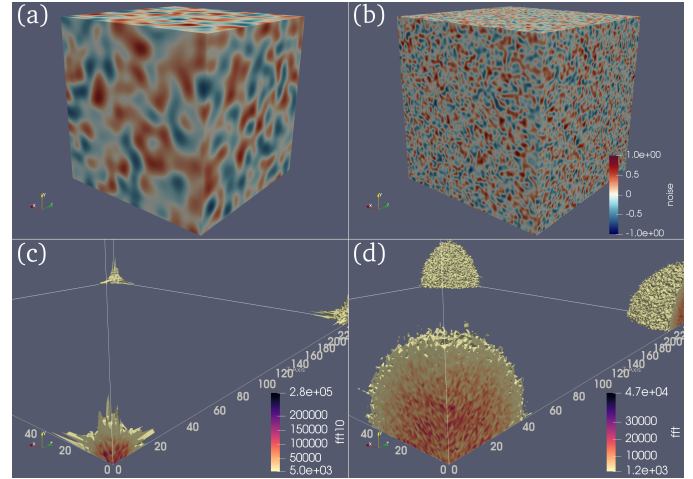


Fig. 2. OSN synthetic data, in 3D, 256^3 in size, at (a) 10 Hz, (b) 40 Hz, and the Fourier transform of the data at (c) 10 Hz and (d) 40 Hz.

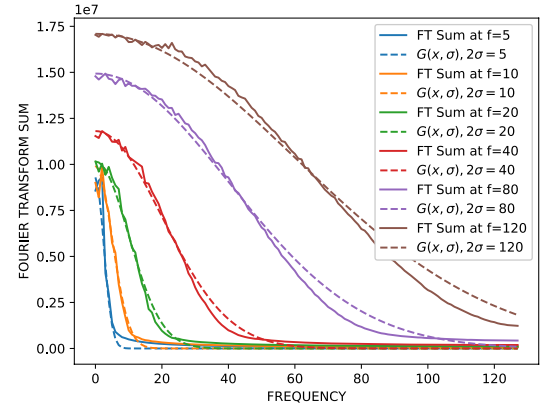


Fig. 3. Sums of 3D Fourier transforms along one axis of OSN at frequencies of 5, 10, 20, 40, 80, and 120 Hz (solid lines). The dashed lines show Gaussian curves $G(x, \sigma)$, where 2σ is equal to the same frequencies.

There are other options for coherent noise. In [25], the authors fill the frequency space with the desired spectral content then perform an inverse Fourier transform to generate data. A similar algorithm was developed by Peachy [26]. However, Fourier transforms are very expensive at large scales because of the all-to-all communication required for the transpose. In addition, the application of Fourier transforms for irregular grids is difficult. Another method, known as Weiner interpolated noise [27], attempts to improve upon known spectral issues in Perlin-based noise (which includes OSN). Perlin-based noise is not band-limited, so any high-frequency content includes low frequencies. For example, 10 and 40 Hz synthetic noise samples shown in Fig. 2c and 2d include all lower frequencies along all three axes. However, the Weiner interpolation suffers from similar band-limitation issues when sampling at higher dimensions. Wavelet noise [28] solves the issue elegantly and provides a convenient spectral-scale space to populate spectral

properties like a real data set. However, a large 4D inverse wavelet transform would be required for temporal coherency in a large synthetic data set. This is not prohibitively expensive though still more than OSN. Also, the wavelet transform is well suited to structured data, but it would be non-trivial to extend the method to irregularly-gridded data. This investment did not seem necessary to create synthetic data that serves as a proxy for evaluating compression performance at scale.

Creating a realistic data proxy begins with understanding the spectral properties of OSN. Fig. 2 shows the 3D Fourier transform (FT) of two OSN-generated data sets at 10 Hz and 40 Hz. Notice how the majority of spectral content in the FT's in Fig. 2c and 2d tapers down to and ends at 10 Hz and 40 Hz, respectively, along each axis. Summing the values along any two axes to produce a 1D curve along the remaining axis produces a Gaussian curve. Fig. 3 shows that the frequency used to produce the synthetic data is twice the standard deviation, σ , of the Gaussian function $G(x, \sigma)$.

C. Verifying Coherent Noise versus Real Data

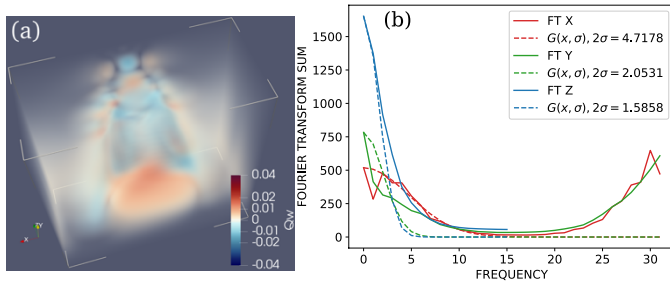


Fig. 4. Variable Q_w at timestep 12990 from the Tapered Cylinder data set, (a) volume rendered, and (b) FT sums along x (red), y (green), and z (blue) axes with Gaussian curve fits along each axis.

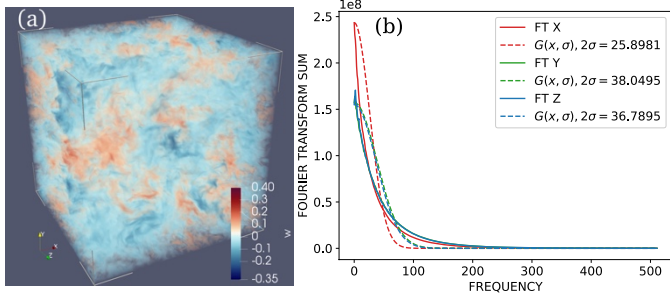


Fig. 5. Variable U_w of the Buoyancy Driven Turbulence data set, (a) volume rendered, and (b) FT sums along x, y, and z axes with Gaussian curve fits along each axis.

The first step, therefore, is to perform a 3D FT on the original data. Since OSN will produce spectral content up to the specified frequency on each axis, only one frequency may be specified for each dimension. The second step is therefore to produce a 1D projection of the original data's FT onto each axis using a summation. Finally, a Gaussian curve is fit to the 1D FT projections and the corresponding standard deviations σ_x , σ_y , and σ_z can be used to generate the appropriate coherent noise.

Fig. 4 and 5 show examples of the process of fitting Gaussian curves to the 3D FT sums along each axis of one variable from two respective data sets. Fitting a Gaussian curve to the FT sum of real data is probably the trickiest part of the process. A number of methods are possible for determining the “best” fit.

These include the following, with names that we provide for later reference in the results:

- **FitFull** – fit the Gaussian function amplitude and σ to the full FT sum curve including the 0 Hz value using least-squares – this assumes that the 0 Hz value is important in the spectral content, but need not be the exacting starting point of the fit curve,
- **FitFixA** – fix the Gaussian amplitude and fit the σ to the full FT sum curve including the 0 Hz value using least-squares – this assumes that the 0 Hz value is critical to the spectral content and the fit curve must start there,
- **Fit1Hz** – fix the Gaussian amplitude at the 1 Hz value divided by $e^{-0.5}$ as a replacement for the original 0 Hz value and fit the σ to the FT sum curve starting at 1 Hz using least-squares – this discards the 0 Hz value in favor of a replacement that is a natural extrapolation of a Gaussian at a given 1 Hz value,
- **Area2Hz** – match the integral (area) of the Gaussian to the integral of the FT sum curve – this allows the curve fit to add extra low-frequency content to compensate for high-frequency content that may be beyond the decay of the Gaussian curve (e.g., see Fig. 4b).

Four publicly available data sets were used to test the OSN fitting approach, with a wide variety of sizes and challenging spectral content:

- **Tapered Cylinder** – a NASA data set with viscous flow around a tapered cylinder, with a grid of $64 \times 64 \times 32$, including five variables: density (Q), total energy per unit volume ($Qe0$), x-momentum (Qu), y-momentum (Qv), z-momentum (Qw), all nondimensional, with 400 time steps (only 4 were included in this study) [29];
- **CAM** – netCDF initialization data for the Community Atmosphere Model (CAM) with a grid of $128 \times 26 \times 64$, including four variables: specific humidity (Q), temperature (T), zonal wind (U), and meridional wind (V) [30];
- **ECHAM** – netCDF output from the European atmospheric general circulation model (ECHAM) with a grid of $192 \times 96 \times 47$, including the relative humidity ($RelHum$) variable [30];
- **Buoyancy Driven Turbulence** – simulation output from the Los Alamos direct numerical simulation code (CFDNS) with a grid of 1024^3 , including five variables: density, pressure, and x-velocity (Uu), y-velocity (Uv), and z-velocity (Uw) [31,32].

The process of testing is, for each variable in each real data set: (1) compress the original data using a predefined set of compression methods, (2) find the OSN frequencies per axis using the aforementioned fitting process on the original data, (3) produce OSN synthetic data using those frequencies, (4) compress the OSN synthetic data using the same set of predefined compression methods, and (5) compare the compression ratios of the compressed original data to the those of the compressed OSN synthetic data.

TABLE I. TAPERED CYLINDER DATA SET RESULTS

Q – $\sigma_x=3.33$ (Fit1Hz), $\sigma_y=1.65$ (Fit1Hz), $\sigma_z=1.77$ (Fit1Hz)							
Method	Oratio	Sratio	RelErr	Method	Oratio	Sratio	RelErr
zfp/1e-4	5.77	5.12	-0.12	sz/1e-4	4.11	4.08	-0.01
zfp/1e-3	11.27	9.73	-0.14	sz/1e-3	8.56	8.06	-0.06
zfp/1e-2	20.13	16.74	-0.18	sz/1e-2	29.26	18.23	-0.55
zfp/1e-1	33.66	28.38	-0.17	sz/1e-1	170.22	142.15	-0.18
Blosc	1.95	1.56	-0.22				
Qe0 – $\sigma_x=4.01$ (Fit1Hz), $\sigma_y=1.61$ (Fit1Hz), $\sigma_z=1.63$ (Fit1Hz)							
zfp/1e-4	6.00	5.81	-0.03	sz/1e-4	4.31	4.49	+0.04
zfp/1e-3	12.16	11.68	-0.04	sz/1e-3	9.11	9.14	+0.00
zfp/1e-2	23.45	21.27	-0.09	sz/1e-2	26.05	24.36	-0.06
zfp/1e-1	44.73	37.29	-0.18	sz/1e-1	169.93	142.15	-0.18
Blosc	1.30	1.24	-0.05				
Qu – $\sigma_x=4.06$ (Fit1Hz), $\sigma_y=1.5$ (Fit1Hz), $\sigma_z=1.5$ (Fit1Hz)							
zfp/1e-4	4.24	3.83	-0.10	sz/1e-4	2.91	2.88	-0.01
zfp/1e-3	7.55	6.90	-0.09	sz/1e-3	4.65	4.62	-0.01
zfp/1e-2	13.21	11.83	-0.11	sz/1e-2	10.13	9.57	-0.06
zfp/1e-1	24.98	21.63	-0.14	sz/1e-1	32.22	31.35	-0.03
Blosc	1.38	1.19	-0.15				
Qv – $\sigma_x=4.57$ (Fit1Hz), $\sigma_y=3.63$ (FitFull), $\sigma_z=1.71$ (FitFull)							
zfp/1e-4	4.34	3.34	-0.27	sz/1e-4	3.08	2.88	-0.06
zfp/1e-3	7.77	5.94	-0.28	sz/1e-3	5.21	4.60	-0.12
zfp/1e-2	13.58	10.44	-0.27	sz/1e-2	12.59	9.60	-0.28
zfp/1e-1	26.19	20.07	-0.28	sz/1e-1	55.78	38.39	-0.41
Blosc	1.24	1.10	-0.12				
Qw – $\sigma_x=4.71$ (FitFixA), $\sigma_y=2.05$ (FitFull), $\sigma_z=2.17$ (FitFixA)							
zfp/1e-4	5.64	3.87	-0.15	sz/1e-4	4.83	3.59	-0.29
zfp/1e-3	11.14	7.43	-0.15	sz/1e-3	11.48	6.59	-0.59
zfp/1e-2	21.57	14.16	-0.20	sz/1e-2	42.87	14.83	-1.55
zfp/1e-1	44.70	29.48	-0.30	sz/1e-1	170.10	142.15	-0.18
Blosc	1.23	1.08	-0.10				

TABLE II. CAM DATA SET RESULTS

Q – $\sigma_x=0.51$ (FitFull), $\sigma_y=0.77$ (FitFull), $\sigma_z=0.26$ (FitFull)							
Method	Oratio	Sratio	RelErr	Method	Oratio	Sratio	RelErr
zfp/1e-6	7.35	5.94	-0.24	sz/1e-6	5.99	3.59	-0.67
zfp/1e-5	10.01	8.55	-0.17	sz/1e-5	10.22	6.67	-0.53
zfp/1e-4	14.65	12.64	-0.16	sz/1e-4	21.20	12.85	-0.65
zfp/1e-3	30.29	24.07	-0.26	sz/1e-3	63.35	33.94	-0.87
Blosc	1.26	1.64	+0.30				
T – $\sigma_x=0.4$ (FitFull), $\sigma_y=0.42$ (FitFull), $\sigma_z=0.38$ (FitFull)							
zfp/1e-4	3.16	3.15	-0.01	sz/1e-4	2.26	1.92	-0.18
zfp/1e-3	4.02	5.04	+0.25	sz/1e-3	4.35	2.85	-0.52
zfp/1e-2	5.03	7.00	+0.39	sz/1e-2	6.90	5.04	-0.37
zfp/1e-1	6.72	10.16	+0.51	sz/1e-1	11.78	9.49	-0.24
Blosc	1.33	1.86	+0.40				
U – $\sigma_x=2.35$ (FitFixA), $\sigma_y=0.4$ (FitFull), $\sigma_z=0.3$ (FitFull)							
zfp/1e-4	3.01	2.77	-0.09	sz/1e-4	1.04	1.10	+0.06
zfp/1e-3	3.78	4.16	+0.10	sz/1e-3	3.69	2.12	-0.74
zfp/1e-2	4.66	5.99	+0.28	sz/1e-2	6.03	3.34	-0.80
zfp/1e-1	6.09	9.10	+0.49	sz/1e-1	9.77	5.95	-0.64
Blosc	1.18	1.42	+0.21				
V – $\sigma_x=2.39$ (FitFull), $\sigma_y=0.59$ (FitFixA), $\sigma_z=1.84$ (FitFull)							
zfp/1e-4	3.05	2.26	-0.35	sz/1e-4	1.03	1.08	+0.05
zfp/1e-3	3.83	3.23	-0.19	sz/1e-3	3.50	2.12	-0.65
zfp/1e-2	4.74	4.59	-0.03	sz/1e-2	5.85	3.25	-0.80
zfp/1e-1	6.21	7.06	+0.14	sz/1e-1	9.29	5.72	-0.63
Blosc	1.15	1.30	+0.13				

TABLE III. ECHAM DATA SET RESULTS

RelHum – $\sigma_x=20.2$ (Area2Hz), $\sigma_y=6.61$ (Area2Hz), $\sigma_z=3.03$ (Area2Hz)							
Method	Oratio	Sratio	RelErr	Method	Oratio	Sratio	RelErr
zfp/1e-4	2.81	2.62	-0.07	sz/1e-4	3.53	2.64	-0.34
zfp/1e-3	4.00	3.94	-0.02	sz/1e-3	5.44	3.78	-0.44
zfp/1e-2	5.59	6.22	+0.11	sz/1e-2	9.36	6.25	-0.50
zfp/1e-1	8.72	11.63	+0.33	sz/1e-1	22.71	13.89	-0.64
Blosc	1.39	1.32	-0.05				

TABLE IV. BUOYANCY DRIVEN TURBULENCE DATA SET RESULTS

Density – $\sigma_x=29.3$ (Area2Hz), $\sigma_y=30.4$ (Area2Hz), $\sigma_z=31.3$ (Area2Hz)							
Method	Oratio	Sratio	RelErr	Method	Oratio	Sratio	RelErr
zfp/1e-6	3.54	4.07	+0.15	sz/1e-6	6.42	8.60	+0.34
zfp/1e-5	5.22	6.28	+0.20	sz/1e-5	12.66	14.83	+0.17
zfp/1e-4	8.70	10.06	+0.16	sz/1e-4	17.26	18.43	+0.07
zfp/1e-3	21.13	20.15	-0.05	sz/1e-3	39.51	22.17	-0.78
Blosc	2.28	2.04	-0.12				
Pressure – $\sigma_x=6.16$ (Fit1Hz), $\sigma_y=6.5$ (Fit1Hz), $\sigma_z=9.2$ (Fit1Hz)							
zfp/1e-6	4.63	6.84	+0.48	sz/1e-6	9.17	16.54	+0.80
zfp/1e-5	7.35	10.06	+0.37	sz/1e-5	16.09	18.52	+0.15
zfp/1e-4	13.07	15.13	+0.16	sz/1e-4	18.84	19.57	+0.04
zfp/1e-3	32.23	26.83	-0.20	sz/1e-3	38.74	21.44	-0.81
Blosc	1.34	1.42	+0.07				
Uu – $\sigma_x=35.3$ (FitFull), $\sigma_y=35.1$ (FitFull), $\sigma_z=23.1$ (FitFull)							
zfp/1e-6	2.62	2.59	-0.01	sz/1e-6	3.67	3.58	-0.03
zfp/1e-5	3.46	3.41	-0.01	sz/1e-5	6.10	5.77	-0.06
zfp/1e-4	5.01	4.94	-0.01	sz/1e-4	12.93	11.99	-0.08
zfp/1e-3	10.01	9.37	-0.07	sz/1e-3	18.08	17.18	-0.05
Blosc	1.35	1.24	-0.09				
Uv – $\sigma_x=38.0$ (FitFull), $\sigma_y=24.1$ (FitFull), $\sigma_z=33.5$ (FitFull)							
zfp/1e-6	2.63	2.55	-0.03	sz/1e-6	3.69	3.55	-0.04
zfp/1e-5	3.48	3.36	-0.04	sz/1e-5	6.11	5.60	-0.09
zfp/1e-4	5.06	4.84	-0.05	sz/1e-4	12.89	12.35	-0.04
zfp/1e-3	10.16	9.16	-0.11	sz/1e-3	18.16	18.09	-0.00
Blosc	1.31	1.25	-0.05				
Uw – $\sigma_x=25.9$ (FitFull), $\sigma_y=38.0$ (FitFull), $\sigma_z=36.8$ (FitFull)							
zfp/1e-6	2.63	2.54	-0.04	sz/1e-6	3.68	3.54	-0.04
zfp/1e-5	3.49	3.33	-0.05	sz/1e-5	6.10	5.85	-0.04
zfp/1e-4	5.08	4.78	-0.06	sz/1e-4	12.87	12.91	+0.00
zfp/1e-3	10.24	9.02	-0.13	sz/1e-3	18.60	17.26	-0.08
Blosc	1.30	1.26	-0.03				

Tables I-IV contain the results of the comparisons of compressed original to synthetic data. Under the variable header is a sub header naming the columns, including the compression method (*Method*), original data C_R (*Oratio*), synthetic data C_R (*Sratio*), and a relative error between the two ratios (*RelErr*). For the purpose of this study, we consider a factor of 2x difference between the ratios as acceptable. We use the relative error to indicate this factor, as follows:

$$RelErr = (Sratio - Oratio) / \min(Sratio, Oratio) \quad (1)$$

Briefly, a *RelErr* of +1.0 means that the synthetic data compresses to a size 100% larger than the original, i.e., twice the compressed size of the original. A *RelErr* of -1.0 means that the synthetic data compresses to half the size of the original.

The predefined compression methods were nearly the same as those used to study the initial compression response of OSN in Fig. 1, that is, Blosc shuffle+ZStd, ZFP, and SZ. For ZFP and SZ, four orders of magnitude were used for the lossiness bound. The bound values started at roughly one order of magnitude less than the original variable such that the highest order would result in at least 10 distinct compressed values (very aggressive loss). Three smaller values follow, each by an order of magnitude. The largest range was $1e^{-1}$ to $1e^{-4}$; the smallest was $1e^{-3}$ to $1e^{-6}$.

In Tables I-IV, we highlight unacceptable values of *RelErr* less than -1.0 with dark blue. *RelErr* values greater than 0.5 and less than -0.5 were highlighted with light red and light blue respectively. These are acceptable by our definition, but we thought it helpful to call out marginal results. Only one result was unacceptable, Qw in Table I; given the *Sratio*, we think that

was an anomaly. The SZ algorithm simply did not compress the synthetic data well at a lossy bound of $1e-2$. Only 15% of the results were marginally acceptable values, and all of the Gaussian fitting methods were used. For further improvement, it is possible, with existing fittings as a starting point, to use a feedback loop to tune the fittings to the compression methods.

III. PARALLEL COMPRESSION PERFORMANCE

Our previous study showed that compression, if done properly, can effectively improve bandwidth for simulation output [33]. Empirically, it found that a combination of fast compression, high C_R 's, and scalability means that as the number of cores increase, the speedup due to compression increases, especially once I/O bandwidth is saturated. So far, lossy compression has been required to attain the higher C_R 's, and bounded lossy compression allows a scientist to quantifiably limit the effects of the lossiness.

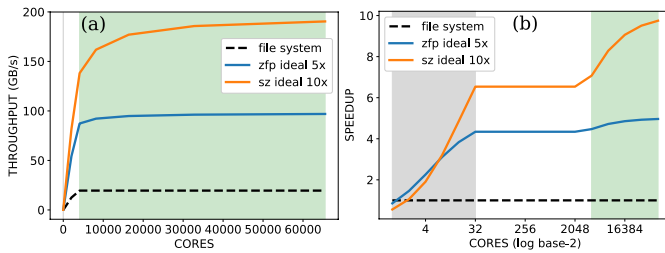


Fig. 6. Plots of an ideal compression performance model illustrating a fast algorithm with lower C_R (ZFP) and a slower algorithm with higher C_R , shown in terms of (a) throughput and (b) speedup vs. cores on a semi-logarithmic scale.

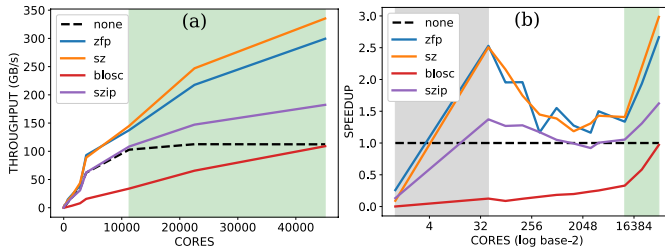


Fig. 7. Actual performance of four compression methods (ZFP, SZ, Blosc, and SZip) in terms of (a) throughput and (b) speedup over I/O without compression.

For this paper, we developed an ideal compression performance model that predicts the weak scaling speedup provided by compression versus the throughput of the file system at a given core count. This is captured in Eq. 2,

$$\text{Speedup} = 1/(1/C_R + F_{BW}/C_{TP}), \quad (2)$$

where C_R is, as before, the compression ratio, C_{TP} is the compression throughput (per core or thread), and F_{BW} is the file system bandwidth. Note that this model assumes no dependency between C_R and the partitioned dataset size (i.e., the data per core). This is reasonable for large partitions but may not hold for small datasets.

An example of the application of Eq. 2 is shown in Fig. 6 with measured values from ZFP and SZ on a typical cluster file system. Values, from an example CFD application, for ZFP were $C_R=5$, $C_{TP}=206$ MB/s; values for SZ were $C_R=10$, $C_{TP}=118$ MB/s. F_{BW} was a function starting at 200 MB/s for a single node (assuming 32 cores per node) and rising linearly per node until

a maximum saturated bandwidth of 20 GB/s. Fig. 6a shows the throughput of the file system by itself and with ZFP and SZ for a range of cores. Fig. 6b shows the speedup of ZFP and SZ relative to the file system throughput at a given core count. Notice that there are three regimes of performance, identified by background color, especially noticeable in Fig. 6b. Gray (barely visible in 6a) indicates intranode performance; white indicates internode performance before the file system is saturated, and green indicates when the file system is saturated at maximum bandwidth. The important regime is the final one, where the speedup scales asymptotically toward the respective C_R . This shows that, at scale, C_R is significantly more important than C_{TP} .

Fig. 7 shows compression performance on a real system using MiniIO with our synthetic data. The system is a Cray XC40 with 2,858 nodes of Intel Broadwell cores (44 each) and a Lustre file system. The grid was $128 \times 128 \times 1024$ points per MPI rank with the data values ranging from -1 to 1. OSN was configured to produce typical compressible data, in this case, somewhere in the middle of frequencies seen in Tables I-IV (we chose a frequency of 10 Hz per 128 grid points). ZFP and SZ were set at lossy bounds of $1e-4$ each. This yielded the following C_R 's: SZ (23.8 ± 2.73), ZFP (3.54 ± 0.474), SZip (1.80 ± 0.156), and Blosc (2.34 ± 0.098), where the $\pm$ errors are standard deviation of C_R across all core counts. Blosc (at these extreme settings) was slow enough that it negatively impacted speedup, despite a good C_R for lossless compression. The others performed mostly as expected and exhibited the three performance regimes. In the pre-saturated bandwidth phase (white background in Fig. 7), ranging from 44 to 11,264 cores, the file system speedup seems to overtake the compression speeds. While this produces an overall throughput improvement, the speedup relative to no compression drops. The expected behavior resumes beyond 11,264 cores (green background), where we predict that each compression method will approach its C_R as a speedup factor.

IV. CONCLUSION

This paper aims to demonstrate that OSN is a sufficiently good stand-in for real data given some initial spectral characterization of that data. Since OSN is highly scalable, it can be used on very large future and even exascale systems to evaluate parallel compression algorithms for I/O. We utilize a mini-app, driven by OSN, to show that, at large core counts and given a fast enough minimum compression throughput, the key metric that influences speedup is the C_R .

This material is based upon work supported by, or in part by, the Department of Defense High Performance Computing Modernization Program (HPCMP) under User Productivity, Technology Transfer, and Training (PETTT) contract # GS04T09DBC0017. Any opinions, finding and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the DoD HPCMP.

REFERENCES

- [1] S. B. Ziegeler, "An I/O mini-app dedicated to in situ visualization," Proc. Second Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization, SC'16, ACM, IEEE, Salt Lake City, UT, November 2016.
- [2] S. B. Ziegeler, "An in situ derived data I/O mini-app: design and early results," Department of Energy Computer Graphics Forum, Pacific Grove, CA, April 25-28, 2016.
- [3] R. Hedges, B. Loewe, T. McLarty, and C. Morrone, "Parallel file system testing for the lunatic fringe: the care and feeding of restless I/O power users," Proc. 22nd IEEE/13th NASA Goddard Conference on Mass Storage Systems and Technologies, Monterey, CA, April 2005.
- [4] Q. Liu, J. Logan, Y. Tian, H. Abbasi, N. Podhorszki, J. Y. Choi, S. Klasky, R. Tchoua, J. Lofstead, R. Oldfield, M. Parashar, N. Samatova, K. Schwan, A. Shoshani, M. Wolf, K. Wu, and W. Yu, "Hello ADIOS: the challenges and lessons of developing leadership class I/O frameworks," in *Concurrency Computat: Pract. Exper.*, 26, Aug. 2013, pp. 1453–1473.
- [5] M. C. Miller, "Design and implementation of MACSio," Technical Report. Lawrence Livermore National Laboratory, LLNL-TR-670388, 2015.
- [6] M. C. Miller, "Probing portable performance of parallel I/O paradigms with MACSio," COE Performance Portability Meeting, Glendale, AZ., April 2016.
- [7] Intel, "Intel MPI Benchmarks User Guide," 2017. [Online]. Available: <https://software.intel.com/en-us/imb-user-guide> [Accessed 19-Aug-2018].
- [8] A. Ching, A. Choudhary, W. Liao, L. Ward, and N. Pundit, "Evaluating I/O characteristics and methods for storing structured scientific data," in *International Parallel and Distributed Processing Symposium*, IEEE/ACM IPDPS, Rhodes Island, Greece, 2006.
- [9] R. Rabenseifner, A. E. Koniges, J. Prost, and R. Hedges, "The parallel effective I/O bandwidth benchmark: b_eff_io," *Parallel I/O for Cluster Computing*, 4, 107-132, Kogan Page Ltd., 2004, ISBN 1-903996-50-3.
- [10] Universität Hamburg, "netCDF-bench," 2018. [Online]. Available: https://wr.informatik.uni-hamburg.de/research/projects/bullio/benchmark_tool/start [Accessed 19-Aug-2018].
- [11] EPCC at the University of Edinburgh, "BenchIO," 2018. [Online]. Available: <https://github.com/EPCCed/benchio> [Accessed: 19-Aug-2018].
- [12] CORAL Collaboration, "HACC I/O," 2014. [Online]. Available: <https://asc.llnl.gov/CORAL-benchmarks/#hacc> [Accessed: 19-Aug-2018].
- [13] P. Wong and R. F. Van der Wijngaart, "NAS Parallel Benchmarks I/O Version 2.4," NAS Technical Report NAS-03-002, 2003.
- [14] J. Borrill, J. Carter, L. Oliker, D. Skinner, and R. Biswas, "Integrated performance monitoring of a cosmology application on leading HEC platforms," Proc. International Conference on Parallel Processing, ICPP'05, Oslo, Norway, June 2005.
- [15] K. Perlin, "An image synthesizer," Proc. ACM SIGGRAPH '85, San Francisco, CA, July 1985.
- [16] K. Perlin, "Improving noise," Proc. ACM SIGGRAPH '02, San Antonio, TX, July 2002.
- [17] S. M. Cameron, "Open simplex noise in C," 2015. [Online]. Available: <https://github.com/smcameron/open-simplex-noise-in-c> [Accessed: 19-Aug-2018].
- [18] F. Alted, "Why modern CPUs are starving and what can be done about it," in *Computing in Science & Engineering*, 12, 2, 68-71, IEEE, 2010.
- [19] L. Dignan, "Facebook open sources Zstandard data compression algorithm, aims to replace technology behind Zip," *Between the Lines*, ZDNet. 2016. [Online]. Available: <https://www.zdnet.com/article/facebook-open-sources-zstandard-data-compression-algorithm-aims-to-replace-technology-behind-zip/> [Accessed 19-Aug-2018].
- [20] Facebook, "Zstandard," 2018. [Online]. Available: <http://facebook.github.io/zstd/> [Accessed: 19-Aug-2018].
- [21] P. Lindstrom, "Fixed-rate compressed floating-point arrays," *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2674-2683, December 2014, doi:10.1109/TVCG.2014.2346458.
- [22] S. Di, and F. Cappello, "Fast Error-bounded Lossy HPC Data Compression with SZ" *IEEE/ACM IPDPS*, Chicago, Illinois, 2016.
- [23] D. Tao, S. Di, F. Cappello, "A Novel Algorithm for Significantly Improving Lossy Compression of Scientific Data Sets," in *International Parallel and Distributed Processing Symposium*, IEEE/ACM IPDPS, Orlando, Florida, 2017.
- [24] P. Yeh, W. Xia-Serafino, L. Miles, B. Kobler, D. Menasce, "Implementation of CCSDS lossless data compression in HDF," *Earth Science Technology Conference-2002*, Pasadena, California, June 2002.
- [25] X. Xiao and D. Voelz, "Wave optics simulation approach for partial spatially coherent beams," *Optics Express*, 14, 16, 6986-6992, 2006.
- [26] D. Peachy, "Solid texturing of complex surfaces," Proc. ACM SIGGRAPH '85, San Francisco, July 1985). In *Computer Graphics* 19, 3, July 1985, 279-286.
- [27] J.P. Lewis, "Algorithms for solid noise synthesis," Proc. ACM SIGGRAPH '89, New York, New York, August 1989.
- [28] R.L. Cook and T. DeRose, "Wavelet noise", Proc. ACM SIGGRAPH '05, Los Angeles, California, August 2005.
- [29] D. Jespersen and C. Levit, "Numerical simulation of flow past tapered cylinder," AIAA Paper 91-0751, 29th Aerospace Sciences Meeting, Reno, Nevada, January, 1991.
- [30] University Center for Atmospheric Research, Community Programs, "Example netCDF files," 2015. [Online]. Available: <https://www.unidata.ucar.edu/software/netcdf/examples/files.html>. [Accessed: 19-Aug-2018].
- [31] Y. Li, E. Perlman, M. Wan, Y. Yang, R. Burns, C. Meneveau, R. Burns, S. Chen, A. Szalay and G. Eyink, "A public turbulence database cluster and applications to study Lagrangian evolution of velocity increments in turbulence," *Journal of Turbulence* 9, No. 31, 2008.
- [32] E. Perlman, R. Burns, Y. Li, and C. Meneveau. "Data exploration of turbulence simulations using a database cluster," *Supercomputing SC'07*, ACM, IEEE, 2007.
- [33] S. B. Ziegeler, S. Breitenfeld, J. Renteria, and J. Henderson, "I/O mini-apps, compression, and I/O libraries for physics-based simulations," Proc. 2nd Joint International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems, SC'17, ACM, IEEE, Denver, CO, November 2017.