Performance Impacts of In Situ Wavelet Compression on Scientific Simulations

Shaomeng Li
National Center for Atmospheric Research
University of Oregon

Matthew Larsen
Lawrence Livermore National Laboratory

John Clyne
National Center for Atmospheric Research

Hank Childs
University of Oregon

ABSTRACT

In situ compression is a compromise between traditional post hoc and emerging in situ visualization and analysis. While the merits and limitations of various compressor options have been well studied, their performance impacts on scientific simulations are less clear, especially on large scale supercomputer systems. This study fills in this gap by performing in situ compression experiments on a leading supercomputer system. More specifically, we measured the computational and I/O impacts of a lossy wavelet compressor and analyzed the results with respect to various in situ processing concerns. We believe this study provides a better understanding of in situ compression as well as new evidence supporting its viability, in particular for wavelets.

ACM Reference Format:
Shaomeng Li, Matthew Larsen, John Clyne, and Hank Childs. 2017. Performance Impacts of In Situ Wavelet Compression on Scientific Simulations. In *ISAV'17: In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualizationa.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3144769.3144773

1 INTRODUCTION

With each new generation of supercomputers, their ability to generate data outpaces their ability to save data, resulting in an increasing time to write out the content of memory to disk. In response, scientists may 1) continue with the post hoc workflow and save data less often; 2) opt for in situ visualization and analysis so I/O is completely skipped; or 3) adopt in situ compression to relieve I/O while still enabling post hoc visualization and analysis. The third option is a promising compromise because it both prevents potential "missing science" from saving data too sparsely, and allows data explorations without re-running the simulation.

The performance impacts of in situ compression are not fully understood, especially on the large-scale supercomputers of today. This is because while the computational costs are relatively predictable, the I/O performance is subject to many factors on supercomputers, and the parallel filesystems themselves tend to have less intuitive characteristics. We take a first step to understand these impacts in this study. Specifically, we test an in situ lossy wavelet compressor together with a proxy simulation on Cheyenne [6], the flagship supercomputer of National Center for Atmospheric Research. We run a weak scaling test with up to 1,000 compute nodes and measured the performance impacts in both computation and I/O. Based on data collected from this experiment, our major contributions in this paper are 1) analysis of a weak scaling experiment on a large-scale supercomputer, 2) derivation of an empirical equation regarding I/O tradeoffs for in situ compression, and 3) discussion on various in situ wavelet compression considerations.

2 BACKGROUND

2.1 In Situ Processing and Analysis

In situ processing, also referred to as "co-processing," has been traditionally used for computational steering that allows users to alter a simulations as it executes for reasons such as preventing crashes and changing numerical solvers [20]. More recently, the growing I/O gap has reinvigorated research into in situ techniques, so analysis can take place while the data is still in memory or the amount of data is reduced before it is written to disk. Some representatives include Libsim [25] and Catalyst [8] for in situ visualization, Cinema [1] and proxy images [24] for in situ generation of exploration-enabled imagery, and an I/O mini-app dedicated to in situ visualization [27]. Finally, we remind our readers a comprehensive survey of state-of-the-art in situ systems by Bauer et al [3].

2.2 Compression of Scientific Data

Compression has been well explored to reduce the size of scientific data. Lossless techniques (e.g., Fpzip [18] and FPC [4]) retain the full integrity of data, but hardly achieve impressive reduction factors. As a result, more techniques opt for a lossy compression, with examples being ZFP [17], SZ [7, 23], ISABELA [11], and wavelet based compressors [10, 21, 22]. Lossy compression techniques can achieve aggressive reduction factors while still allow meaningful analysis to be carried out, as reported in analyses of turbulent flow data [15], climate data [2, 26], and physics-based proxy simulations [12].

For in situ compression, the compressor's ability to utilize parallelism becomes critical. ISABELA divides data into windows and compresses each window independently [11]. VAPOR divides a volume into domains (typically of size 64^3) and applies wavelet compression on each domain independently [21]. Li et al. [16] reported a wavelet compression implementation using data parallel primitives, achieving portable performance among multiple architectures. This work differs in that we focused on the compressor's performance impacts to simulations in an in situ setting.

3 EXPERIMENT OVERVIEW

3.1 Software Used

Our experiments ran within the ALPINE Ascent in situ infrastructure [14] (a production version of Strawman [13]), which enables fast prototyping of in situ algorithms. The infrastructure consists of two main parts: an interface to simulations and a hybrid-parallel library that provides a distributed-memory layer on top of shared-memory parallel algorithms in VTK-m [19]. Ascent contains three physics proxy-applications to start with.

Our study used the Lulesh [9] proxy-application, a 3D Lagrangian shock hydrodynamics code. Lulesh implements the Sedov test problem, which deposits initial energy at one corner of a cube, and propagates a shock wave from the origin outward to the rest of the cube. In terms of computation, Lulesh uses thread-level parallelism (via OpenMP) for calculation within a domain, and process-level parallelism (via MPI) for multiple domains.

The lossy wavelet compressor we used in this study is an implementation introduced in [16] that we integrated into ALPINE's hybrid-parallel library. This implementation fits well in our in situ experiment settings because it has comparable performance with the best multi-thread CPU wavelet compressors, and is versatile enough to enable performant compression on GPUs. It implements filter-bank based wavelet transforms, and we used the CDF 9/7 [5] wavelet kernel in this study. For lossy compression, it simply keeps wavelet coefficients containing the most information (i.e., largest magnitudes) and discards the rest. For example, a 64:1 compression means keeping $1/64th$ of all coefficients. In terms to computation, this implementation achieves thread-level parallelism using data parallel primitives on single compute nodes, but does not have inter-node communication. After finishing compressing, each node independently writes its data in the compressed form to disk via an `fwrite()` function call of the C programming language.

3.2 Simulation with In Situ Compression

Our simulation and compression code runs in an in situ setting: at the end of each simulation cycle, simulation variables are passed to the compression code to perform compression, and then written to disk in the compressed form. Multiple variables are processed one at a time, during which the simulation is suspended. As a result, each complete cycle consists of one simulation step and multiple compression and writing steps (for multiple variables). Specifically, our experiment has seven variables: pressure, energy, relative volume, artificial viscosity, and the x, y, z components of velocity. One might think compressing multiple variables together could better exploit the available parallelism. However, our in situ wavelet compressor is already a parallel implementation, so simultaneous compression would not make a big difference in computational time. (It would make more difference in memory consumption though, see Subsection 5.2). Finally, while data being saved so frequently does not reflect real world usage, it is sufficient for our study.

Our in situ compression is "tightly coupled." This means, each compute node not only performs simulation on a domain, but also compresses the data of the same domain without data movement through the network.

Disk I/O is handled individually by each compute node as well. After processing a variable of a domain, the compute node writes its content to disk as an individual file no matter whether there is compression or not, until all variables are processed.

We note that we use MPI barriers in the code to keep all MPI ranks in sync, meaning that they are always in the same stage of performing simulation, compression, or writing. We feel this represents real-world usage in terms of bursty I/O coming from all nodes at the same time.

Finally, our compression code treats compression ratio 1:1 as a special case — it does not perform any compression on the data, and directly passes the data to the file writer. In our tests, this configuration acts as the baseline case to compare and measure in situ compression impacts.

3.3 Experiment Runs

We ran our experiments on Cheyenne, the flagship supercomputer of National Center for Atmospheric Research. Each compute node is equipped with two 18-core Xeon CPUs at 2.3 GHz (36 cores in total) and 64 GB memory. Both simulation and compression software were compiled with GCC-6.3.0, and they used 64-bit floating point values to carry out their computation.

With the Lulesh simulation, we fixed the size of each MPI rank's domain size to be 320^3 (320^3 cells or 321^3 vertices). The entire simulation then scales up by using more domains/MPI ranks, making it a weak scaling problem. Lulesh supports the number of MPI ranks being cubics of natural numbers, i.e., 1^3, 2^3, 3^3, etc.

We vary two parameters for experiment runs:

- number of MPI ranks: 1, 8, 27, 64, 125, 216, 343, 512, 729, and 1,000;
- compression ratio: 1:1, 16:1, 64:1, and 128:1;

The number of MPI ranks is essentially the number of compute nodes we use, since we assign one MPI rank to a node. The total number of tests we performed is then $10 \times 4 = 40$.

4 RESULTS

4.1 Execution Time Impacts

We present the execution times in Figure 1. Each column is one experiment run. These columns are grouped into groups of four; each group has experiment runs with the same number of compute nodes. The four columns within each group differ in their compression ratios; they are 1:1, 16:1, 64:1, and 128:1 from left to right. Note the 1:1 column has no compression time.

Each column provides an execution time per cycle averaged from five cycles. Though each cycle has one simulation step, it has seven compression and writing steps for seven variables (see Subsection 3.2). The sum of the seven compression and writing times are reported here. The runs were performed during a lull on the machine in the middle of the night, in an effort to minimize I/O contention with other jobs running on the supercomputer.

Experiment results show that simulation time stays relatively steady at around 9 seconds per cycle, no matter how many nodes are in use and whether or not in situ compression is involved. Wavelet compression adds computational overhead to every cycle. This overhead is mostly consistent as well (around 5 seconds per cycle), because the wavelet transform and coefficient thresholding steps are both independent of the final compression ratio.

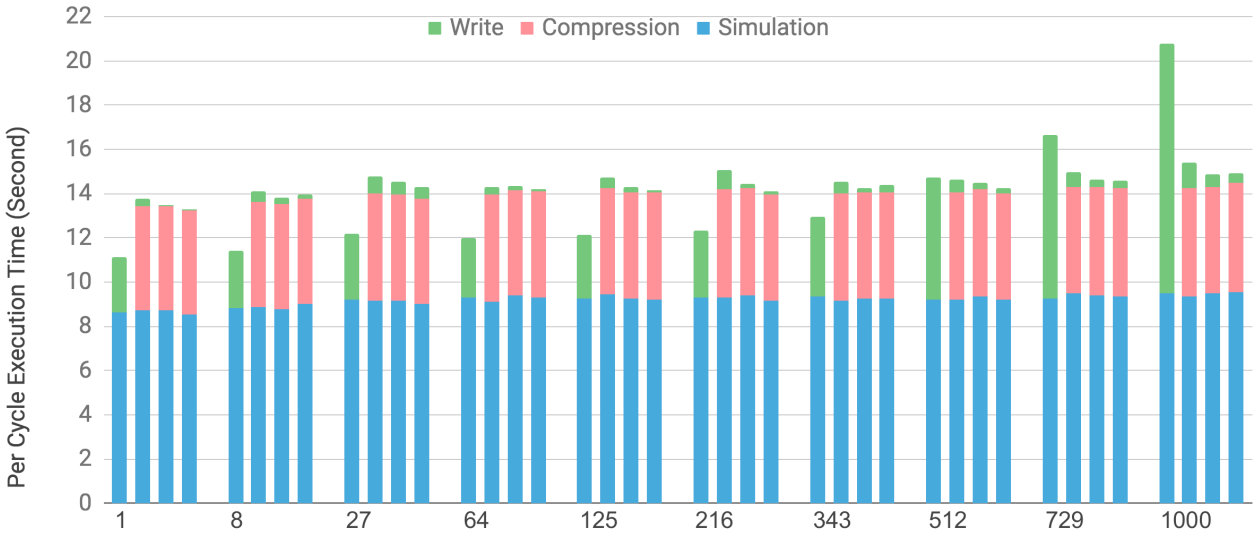

Figure 1: Per cycle execution time breakdown of 40 experiment runs. Each group of four uses the same number of compute nodes (X labels), but different compression ratios: 1:1, 16:1, 64:1, and 128:1 (from left to right within each group).

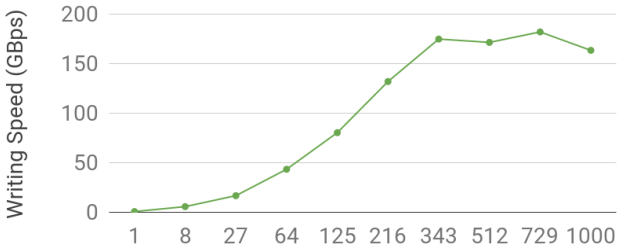

Figure 2: Achieved parallel filesystem writing speed with different number of compute nodes (X-axis).

4.2 I/O Performance Analysis

The I/O time shows more interesting results than computational time considering the huge difference of the amount of data being written to the filesystem. With our experiment settings, each node generates a fixed amount of raw data per cycle. The total amount of data is then proportional to the number of nodes. However, looking at the writing time of raw data (left-most columns of each group of four), they are by no means proportional to the total amount of data being written.

We consider this interesting I/O behavior due to the very large aggregate bandwidth of parallel filesystems. This bandwidth is large enough that a small set of compute nodes are not using up all of it — the I/O is constrained by the network between individual nodes and the parallel filesystem. After the number of concurrently-writing nodes grows past a certain point, the I/O bottleneck starts to shift to the parallel filesystem, when its bandwidth gets saturated. Looking at our results, the raw data writing time is almost constant up to 216 nodes, and then starts to grow with the number of nodes.

We compared our results with the specs of the test system (Cheyenne of NCAR), which is equipped with a 200 GBps filesystem. The amount of raw each compute node generates each cycle is roughly 1.84 GB. The achieved aggregate I/O is then calculated using the total amount of data and writing time, which is presented in Figure 2. It tops out at approximately 170 GBps starting from 343 nodes. We achieved this plausible I/O rate because we run our experiments when the supercomputer is relatively idle, and in-production simulations are likely to face more contentious conditions.

The I/O time with compression is much less consistent, meaning that writing time for the same-ratio compressed data vary considerably from one run to another. Compared to raw data writing time, it is almost always taking a larger percentage of time than its data size fraction. We consider this to be due to the relatively large latency of parallel filesystems, which introduces a lot of variance to writes with small amount of data. The only certainty for such scenarios is that writing compressed data takes much less time than raw data.

5 VIABILITY OF IN SITU COMPRESSION

5.1 Overall I/O Viability

With in situ compression, computational overhead is incurred while the actual I/O is most likely reduced. We thus consider the overall I/O, which includes both computational and the actual I/O cost. In our experiment, computational overhead counts for the majority of overall I/O, which may or may not improve over writing raw data.

In the case where the number of compute nodes is small, the achievable aggregate I/O is able to grow as the number of compute node grows. The computation of the compression then needs to be performed fast enough to improve the overall I/O, at least faster than writing raw data to disk. This can be considered as a performance lower bound. The in situ wavelet compression implementation we tested fails in this criterion, and it caused overall I/O to grow with less than 512 nodes.

In the case with a large number of compute nodes, their concurrent I/O requests may top out past the aggregate I/O of the parallel filesystem. In situ compression is then possible to improve overall I/O here. Given a filesystem with aggregate I/O bandwidth V_{aggr},

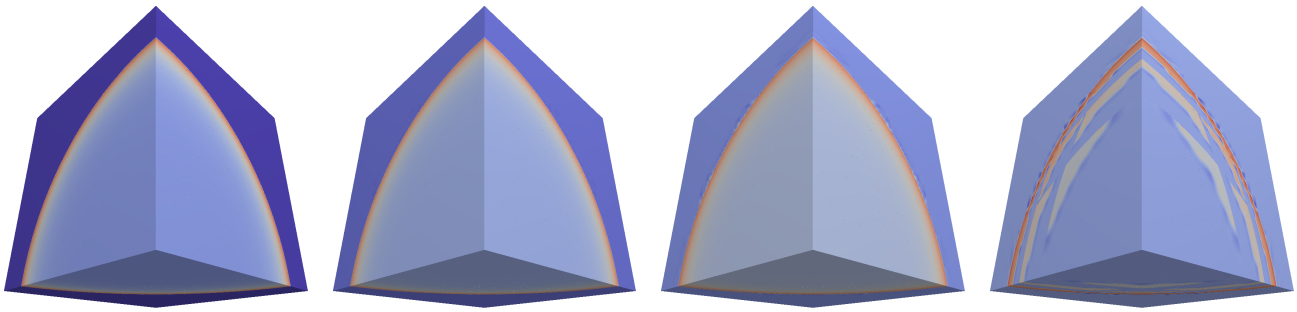

Figure 3: Renderings of the pressure field from Lulesh: a shock wave propagates in a cube. From left to right, they are from the raw data, and compressed data with ratios 16:1, 64:1, 128:1. The most artifacts emerge on the shock wave front.

N compute nodes each generating D_{domain} amount of data, the I/O time to write the raw data is $T_{raw} = (N \cdot D_{domain})/V_{aggr}$. Assume an in situ compressor that takes T_{comp} time in calculation and achieves a compression ratio of R, the overall I/O for the same N compute nodes is then $T_{in-situ} = T_{comp} + (N \cdot D_{domain})/(V_{aggr} \cdot R)$. The overall I/O $T_{in-situ}$ improves over T_{raw} when the compression time satisfies the following equation:

$$T_{comp} < \frac{N \cdot D_{domain} \cdot (R-1)}{V_{aggr} \cdot R} .$$

With lossy compression, compression ratios are usually big enough that $(R-1)/R$ can be approximated to be 1. The equation can then be re-written as:

$$N > \frac{T_{comp} \cdot V_{aggr}}{D_{domain}} .$$

This equation means that with enough compute nodes, in situ compression can always improve the overall I/O. First, without much surprise, this threshold is proportional to the computational overhead T_{comp}, so faster compression lowers this threshold. Second, this threshold is also proportional to the rate between aggregate filesystem bandwidth and the domain size (V_{aggr}/D_{domain}). Given that the domain size may be constrained by the memory capacity of compute nodes, and the memory capacity growth keeps outpacing the bandwidth growth, this trend further lowers this threshold. In our experiment, 512 was estimated to be the threshold so the overall I/O was improved with 729 and 1,000 nodes; the I/O time of 1,000 nodes was almost cut in half. We anticipate simulation runs with larger numbers of nodes would benefit even more.

Besides the factors we have analyzed here, the overall I/O of real-world simulation runs is also subject to other variations, including hardware specifications, parallel filesystem characteristics, and even other users concurrently using the system. An accurate analysis on the overall I/O is even more challenging with these variations. We believe the trend is a more important takeaway, which is that larger simulations are more likely to benefit from a reduced overall I/O, and the amount of benefit is going to grow as the relative I/O bandwidth keeps decreasing. Moreover, any potential improvement of the overall I/O is already on top of storage benefit from a reduced data size.

5.2 Memory Viability

Memory overhead is another consideration of in situ compression, since many simulations are already bound by available system memory. The wavelet compressor we used does not work in a streaming fashion, instead, it requires an extra buffer space for each variable it processes. That means, for a variable of size S, it introduces a memory overhead of $2S$. A simulation usually consists of multiple variables. In the case where the in situ compressor fully makes use of the available parallelism, this memory overhead is not multiplied since these variables can be processed one by one. The $2S$ memory overhead thus becomes independent of the total number of variables, making in situ compression more viable. In the case where the compressor is not fully parallelized, multiple variables need to be processed simultaneously to exploit the available parallelism. The $2S$ memory overhead is then multiplied, making in situ compression less viable. The wavelet compressor in our experiment falls into the first case, so it introduced memory overhead that is twice the largest variable at 321^3 resolution, or 529 MB in total, which was acceptable for our test supercomputer system.

Table 1: Error measurements from compression for each compression ratio (first column from left). Both root-mean-square error (RMSE) and $L-\infty$ norm values (second column) are normalized by the range of data. The rest columns are evaluations of five data fields: energy (e), relative volume (v), pressure (p), artificial viscosity (q), and x-velocity (xd).

		e	v	p	q	xd
16	RMSE	$6.7e-8$	$4.1e-8$	$3.5e-4$	$2.2e-4$	$1.1e-3$
	$L-\infty$	$9.2e-7$	$4.7e-7$	$5.3e-3$	$4.4e-3$	$1.5e-2$
64	RMSE	$7.7e-7$	$4.8e-7$	$4.9e-3$	$5.8e-3$	$1.0e-2$
	$L-\infty$	$1.8e-5$	$1.6e-5$	$1.0e-1$	$1.2e-1$	$2.0e-1$
128	RMSE	$1.7e-6$	$1.1e-6$	$1.0e-2$	$1.2e-2$	$1.8e-2$
	$L-\infty$	$3.8e-5$	$2.3e-5$	$2.6e-1$	$2.9e-1$	$3.4e-1$

5.3 Data Integrity Viability

Data integrity is a universal concern of all lossy compression techniques, but the acceptance of information loss greatly varies. This acceptance is usually determined by factors such as the nature of the intended analysis, the accuracy requirement of a particular application, as well as resource limitations such as I/O and storage. Given that this paper focuses on performance impacts of in situ compression, we present a visualization and simple statistics of the resulting compression but choose not to go into detailed analysis.

Figure 3 presents ray tracing results of the pressure field of this shock wave at time step 4,300, in a 128^3 cube. While the 16:1 result looks almost indistinguishable from the baseline, the 64:1 result has clear "ripples" along the shock wave front, since that is where most of the energy resides. The 128:1 result is even more deteriorated with not only rough front, but also artifacts in the volume. Table 1 presents error measurements for multiple data fields at the same time step. The normalized root-mean-square error provides an average deviation, and the normalized $L-\infty$ norm provides the maximum point-wise difference. We used a 128^3 cube for this visualization instead of sizes in our performance tests (320^3 or even larger) because the per-voxel differences are still prominent in smaller volumes; renderings look more similar to each other in a 320^3 cube. Again, data integrity is highly analysis-dependent and a complex topic by itself. Interested readers may further read into recent publications such as [12, 26] and [15].

5.4 Storage Viability

In situ compression reduces the storage cost regardless of its performance: either the same data takes less storage, or the same storage is able to hold more data. Given the data integrity requirements are satisfied, both scenarios are welcome for scientists. Thus, the storage viability is really a benefit rather than a concern.

6 CONCLUSION

We studied performance impacts of in situ wavelet compression on a scientific simulation. Based on results and analysis of a weak scaling experiment on a large-scale supercomputer, we gained better understanding of the I/O impacts of in situ compression, and derived an empirical equation to quantify when we can expect in situ compression to improve the overall I/O. Finally, we argued that in situ compression is a viable alternative to post hoc and in situ analysis by discussing various aspects of its viability.

7 ACKNOWLEDGEMENT

We would like to acknowledge high-performance computing support from Cheyenne (doi:10.5065/D6RX99HX) provided by NCAR's Computational and Information Systems Laboratory, sponsored by the National Science Foundation.

Part of this research was supported by the Exascale Computing Project (17-SC-20-SC), and part of work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344 (LLNL-CONF-737884).

REFERENCES

[1] James Ahrens, Sébastien Jourdain, Patrick O'Leary, John Patchett, David H Rogers, and Mark Petersen. 2014. An image-based approach to extreme scale in situ visualization and analysis. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Press, 424–434.

[2] Allison H Baker, Haiying Xu, John M Dennis, Michael N Levy, Doug Nychka, Sheri A Mickelson, Jim Edwards, Mariana Vertenstein, and Al Wegener. 2014. A methodology for evaluating the impact of data compression on climate simulation data. In *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing*. ACM, 203–214.

[3] A. C. Bauer, H. Abbasi, J. Ahrens, H. Childs, B. Geveci, S. Klasky, K. Moreland, P. O'Leary, V. Vishwanath, B. Whitlock, and E. W. Bethel. 2016. In Situ Methods, Infrastructures, and Applications on High Performance Computing Platforms. In *Proceedings of the Eurographics: State of the Art Reports (EuroVis '16)*. Eurographics Association, Goslar, Germany, 577–597.

[4] M. Burtscher and P. Ratanaworabhan. 2009. FPC: A high-speed compressor for double-precision floating-point data. *IEEE Trans. Comput.* 58, 1 (2009), 18–31.

[5] Albert Cohen, Ingrid Daubechies, and J-C Feauveau. 1992. Biorthogonal bases of compactly supported wavelets. *Communications on pure and applied mathematics* 45, 5 (1992), 485–560.

[6] Computational and Information Systems Laboratory. 2017. Cheyenne: SGI ICE XA System. *Boulder, CO: National Center for Atmospheric Research.*

[7] Sheng Di and Frank Cappello. 2016. Fast Error-bounded Lossy HPC Data Compression with SZ. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, IEEE, Chicago, IL.

[8] Nathan Fabian, Kenneth Moreland, David Thompson, Andrew C Bauer, Pat Marion, Berk Gevecik, Michel Rasquin, and Kenneth E Jansen. 2011. The paraview coprocessing library: A scalable, general purpose in situ visualization library. In *IEEE Symposium on Large Data Analysis and Visualization (LDAV)*. IEEE, 89–96.

[9] Ian Karlin, Jeff Keasler, and Rob Neely. 2013. *LULESH 2.0 Updates and Changes*. Technical Report LLNL-TR-641973. 1–9 pages.

[10] Beong-Jo Kim and William A Pearlman. 1997. An embedded wavelet video coder using three-dimensional set partitioning in hierarchical trees (SPIHT). In *Proceedings of Data Compression Conference (DCC'97)*. IEEE, 251–260.

[11] Sriram Lakshminarasimhan, Neil Shah, Stephane Ethier, Scott Klasky, Rob Latham, Rob Ross, and Nagiza F Samatova. 2011. Compressing the incompressible with ISABELA: In-situ reduction of spatio-temporal data. In *Euro-Par 2011 Parallel Processing*. Springer, 366–379.

[12] Daniel Laney, Steven Langer, Christopher Weber, Peter Lindstrom, and Al Wegener. 2013. Assessing the effects of data compression in simulations using physically motivated metrics. In *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 76.

[13] Matthew Larsen et al. 2015. Strawman: A Batch In Situ Visualization and Analysis Infrastructure for Multi-Physics Simulation Codes. In *Proceedings of the Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV2015)*. ACM, New York, NY, USA, 30–35.

[14] Matthew Larsen, James Aherns, Utkarsh Ayachit, Eric Brugger, Hank Childs, Berk Geveci, and Cyrus Harrison. 2017. The ALPINE In Situ Infrastructure: Ascending from the Ashes of Strawman. In *Proceedings of the In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization Workshop (ISAV2017)*. ACM, New York, NY, USA.

[15] Shaomeng Li, Kenny Gruchalla, Kristin Potter, John Clyne, and Hank Childs. 2015. Evaluating the Efficacy of Wavelet Configurations on Turbulent-Flow Data. In *IEEE Symposium on Large Data Analysis and Visualization (LDAV)*. 81–89.

[16] Shaomeng Li, Nicole Marsaglia, Vincent Chen, Christopher Sewell, John Clyne, and Hank Childs. 2017. Achieving Portable Performance For Wavelet Compression Using Data Parallel Primitives. In *Proceedings of EuroGraphics Symposium on Parallel Graphics and Visualization (EGPGV)*. The Eurographics Association.

[17] Peter Lindstrom. 2014. Fixed-rate compressed floating-point arrays. *IEEE Transactions on Visualization and Computer Graphics* 20, 12 (2014), 2674–2683.

[18] Peter Lindstrom and Martin Isenburg. 2006. Fast and efficient compression of floating-point data. *IEEE Transactions on Visualization and Computer Graphics* 12, 5 (2006), 1245–1250.

[19] Kenneth Moreland, Christopher Sewell, William Usher, Li-ta Lo, Jeremy Meredith, David Pugmire, James Kress, Hendrik Schroots, Kwan-Liu Ma, Hank Childs, et al. 2016. Vtk-m: Accelerating the visualization toolkit for massively threaded architectures. *IEEE computer graphics and applications* 36, 3 (2016), 48–58.

[20] Jurriaan D. Mulder, Jarke J. van Wijk, and Robert van Liere. 1999. A survey of computational steering environments. *Future Generation Computer Systems* 15, 1 (1999), 119 – 129.

[21] A. Norton and J. Clyne. 2012. The VAPOR Visualization Application. In *High Performance Visualization*, E.W. Bethel, H. Childs, and C. Hanson (Eds.). 415–428.

[22] Xiaoli Tang, William A Pearlman, and James W Modestino. 2003. Hyperspectral image compression using three-dimensional wavelet coding. In *Electronic Imaging 2003*. International Society for Optics and Photonics, 1037–1047.

[23] Dingwen Tao, Sheng Di, Zizhong Chen, and Franck Cappello. 2017. Significantly Improving Lossy Compression for Scientific Data Sets Based on Multidimensional Prediction and Error-Controlled Quantization. *Proceedings of the 31th IEEE International Parallel and Distributed Processing Symposium* (2017).

[24] Anna Tikhonova, Carlos D Correa, and Kwan-Liu Ma. 2010. Visualization by proxy: A novel framework for deferred interaction with volume data. *IEEE Transactions on Visualization and Computer Graphics* 16, 6 (2010), 1551–1559.

[25] Brad Whitlock, Jean M Favre, and Jeremy S Meredith. 2011. Parallel in situ coupling of simulation with a fully featured visualization system. In *Proceedings of the 11th Eurographics conference on Parallel Graphics and Visualization*. Eurographics Association, 101–109.

[26] J. Woodring, S. Mniszewski, C. Brislawn, D. DeMarle, and J. Ahrens. 2011. Revisiting wavelet compression for large-scale climate data using JPEG 2000 and ensuring data precision. In *IEEE Symposium on Large Data Analysis and Visualization (LDAV)*. 31–38.

[27] Sean B Ziegeler. 2016. An I/O Mini-App dedicated to in situ visualization. In *Proceedings of the In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization Workshop (ISAV2016)*. IEEE, 29–34.