Evaluating Lossy Compressors for Inline Compression

Donald Elmore
delmore@g.clemson.edu
Clemson University
Clemson, South Carolina

Jon C. Calhoun (Advisor)
jonccal@clemson.edu
Clemson University
Clemson, South Carolina

ABSTRACT

Modern HPC applications require massive amounts of data. The amount of data required is growing faster than memory capabilities are. An example of this is pySDC, a framework for solving collocation problems iteratively using parallel-in-time methods. These methods require storing and exchange of 3D volume data for each parallel point in time. If a simulation consists of M parallel-in-time stages, where the full spatial problem has to be stored for the next iteration, the memory demand for a single state variable is $M * Nx * Ny * Nz$ per time-step. For a simulation with many state variables or stages, the memory requirement is considerable. Lossy compression is a way to counteract this [2]. This poster evaluates several state-of-the-art lossy compressor algorithms such as SZ and ZFP for their applicability to inline compression for pySDC. We evaluate the compressors based on the following metrics: compression ratio, compression bandwidth, decompression bandwidth, and overall error introduced. Results show that using SZ with an error bound of $1e{-}5$, we are able to reduce the memory footprint for the data vectors for the solution u by a factor of 311.99 while maintaining an acceptable level of loss.

CCS CONCEPTS

• **Information systems → Compression strategies**; • **Theory of computation → Data compression**.

1 INTRODUCTION

The data required for large HPC applications creates a performance bottleneck for many applications where the data required will not all fit in RAM. Lossy data compression is an effective tool for reducing memory and storage requirements [2]. During lossy compression, error is introduced to the data. For HPC applications some error is often acceptable, as long as the error is bounded to a known tolerance [1]. If the application allows for some bounded error a greater compression ratio can be achieved, reducing the memory requirement [4]. This idea of trading accuracy for reduction in memory footprint enables running larger, more realistic simulations.

Applications that have large memory overhead, such as pySDC, require reduction in memory requirement. We propose inline lossy compression to counteract this. For applications such as pySDC, where the limiting factor is memory, lossy compression is beneficial.

This poster makes the following contributions:

- Provides an evaluation of state-of-the-art lossy compression algorithms for use in pySDC;
- Shows lossy data compression is an effective tool for reducing memory requirements for pySDC; and
- Highlights current compression/decompression bandwidth is not fast enough for inline lossy compression on HPC applications

2 BACKGROUND

pySDC is a Python implementation of the spectral deferred correction (SDC) approach. pySDC is an acceptable candidate for inline compression because a simulation requires the full spatial problem to be stored for multiple parallel points in time. Due to the need for each of M parallel-in-time stages to have $Nx * Ny * Nz$ data for each time-step the data required for a large simulation is massive [6]. pySDC iteratively converges to a solution. Thus, it allows some error to be acceptable. Introducing error allows the memory requirement to be greatly reduced with lossy compression. pySDC solves $Au = f$ with all stages within a time-step being stored for the next iteration. We compress u at all parallel-in-time stages in order to reduce per node memory requirements, to avoid running out of memory.

3 METHODOLOGIES

pySDC loops over individual parallel points in time, computing within sub-time-steps each loop in order to update u. In order to evaluate inline lossy compression in pySDC, we compress any time a variable is written or updated and decompress any time a variable needs to be read. It is important to note that when data is decompressed for reading it still exists within a compressed state so that after the data has been read it does not have to be recompressed; only the data being actively used is in a decompressed state. Within the pySDC problem used for testing, there exist more reads than writes. Lossy compressor decompression is quicker than compression, so this occurrence can exploited when applying to pySDC because there are more decompresses than compresses needed.

4 EXPERIMENTAL RESULTS

We test a pySDC 1D heat diffusion problem with 64^3 degrees of freedom. pySDC is set with an error tolerance of $1e{-}10$ and max iteration count per time-step of 20. We evolve this problem for 50 time steps and monitor error due to lossy compression against the analytic solution.

For lossy compression we explore the following compressors:

- SZ Version 2.1.5
- ZFP Version 0.5.5
- Naïve truncation from 64 bit precision to 32 bit precision

All tests run on the Clemson University Palmetto Cluster Phase 18b using Intel Xeon 6148G CPU running pySDC version 3.

4.1 Compressor Performance

Figure 1 shows the average compression ratio across all time-steps from each compression algorithm. We see SZ $1e{-}5$ provides the best compression ratio, with truncation performing the worst. ZFP is lacking in our test problem due to poorly compressing 1D data [5].

We reinterpret the data as a 3D array in attempt improve this and compress. Our analysis shows SZ is the most effective compressor for reducing the per node memory size u for our problem. This reduction in memory requirement allows a more detailed problem to be computed, or less resources to be used.

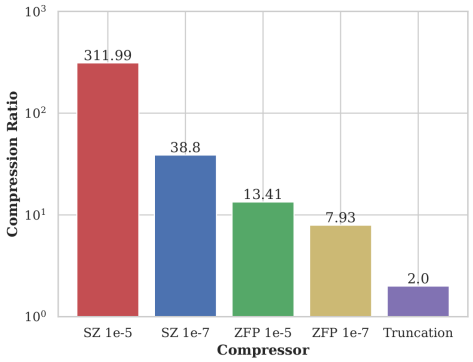

Figure 1: Avg. compression ratio over all time-steps

Figure 2 shows the average compression bandwidth for all stages across all time-steps. Results show that truncation is the fastest by a factor of 10 in terms of bandwidth. Comparing truncation with SZ and ZFP, truncation outperforms the lossy compressors by a factor of 15. This is due to SZ and ZFP being slowed down by compressor logic. Truncation is only limited by the memory bandwidth, approximately 5400 MB/s (STREAM bandwidth), where SZ and ZFP are limited by the logic of the compressor.

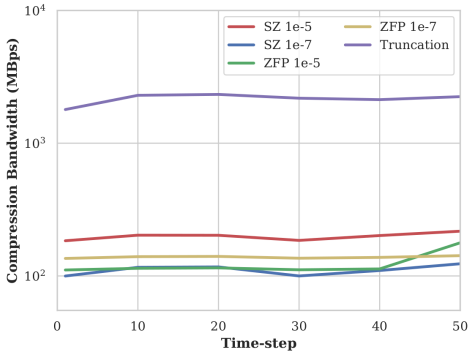

Figure 2: Avg. compression bandwidth over all time-steps

Figure 3 shows the maximal absolute error $e = |u^* - \tilde{u}|_\infty$ due to lossy compression across all time-steps. As more error is allowed, the lossy solution gets further from the analytic solution. The benefit of the lossy compression algorithms here is that they have precise error control for the user's acceptable amount of error, whereas truncation has poor error control.

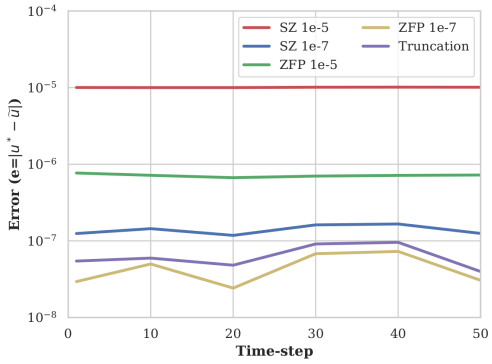

Figure 3: Error from lossy compression over all time-steps

5 CONCLUSION

Results show that truncation is faster than two state-of-the-art lossy compressors by a factor of 10. In order to achieve lossy compression with minimal time overhead, lossy compressor speeds will need to become faster by a factor of 10. This exploits a large gap between memory bandwidth and lossy compressor speeds. Current lossy compressors are not yet able to keep up with what is needed for efficient inline lossy compression. Although truncation achieves the best compression bandwidth, its compression ratio is lacking compared to a current lossy compression algorithm [3].

Currently, there is a large trade-off between lossy compressors and naïve truncation. There is either a large reduction in data size while greatly increasing application run-time, or a minimal reduction in data size and minimal increase in application run-time. With current lossy compression algorithms, inline compression is not yet achievable without greatly increasing the application run-time.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. SHF-1910197. Clemson University is acknowledged for generous allotment of compute time on Palmetto cluster.

REFERENCES

[1] Jon Calhoun, Franck Cappello, Luke N Olson, Marc Snir, and William D Gropp. 2019. Exploring the feasibility of lossy compression for PDE simulations. *The International Journal of High Performance Computing Applications* 33, 2 (2019), 397–410. https://doi.org/10.1177/1094342018762036 arXiv:https://doi.org/10.1177/1094342018762036

[2] Sheng Di and Franck Cappello. 2016. Fast error-bounded lossy HPC data compression with SZ. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 730–739.

[3] Sihuan Li, Sheng Di, Xin Liang, Zizhong Chen, and Franck Cappello. 2018. Optimizing Lossy Compression with Adjacent Snapshots for N-body Simulation Data. In *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 428–437.

[4] Xin Liang, Sheng Di, Dingwen Tao, Sihuan Li, Shaomeng Li, Hanqi Guo, Zizhong Chen, and Franck Cappello. 2018. Error-controlled lossy compression optimized for high compression ratios of scientific datasets. In *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 438–447.

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

[6] Robert Speck. 2018. pySDC - Prototyping spectral deferred corrections. *CoRR* abs/1808.02731 (2018). arXiv:1808.02731 http://arxiv.org/abs/1808.02731