CLEMSON UNIVERSITY

Evaluating Lossy Compressors for Inline Compression

Donald Elmore and Jon Calhoun (advisor)

Holcombe Department of Electrical and Computer Engineering - Clemson University

SC19

Why do we need lossy compression?

Modern HPC applications are generating massive amounts of data with large simulations. Lossy compression is a tool to greatly reduce the memory requirement of these large applications [1].

The data needed for some HPC applications, such as pySDC, will not all fit in RAM.

Introducing error through lossy compression, allows a greater compression ratio than lossless compression.

Lossy Compression

$$\underbrace{\tilde{u}}_{\text{Variable with error after decompression}} = \underbrace{u}_{\text{Original variable}} + \underbrace{\varepsilon}_{\text{User controllable compression error}}$$

What is pySDC?

pySDC is a framework for solving collocation problems iteratively using parallel in time methods, requiring 3D volume data for each parallel point in time [2].

If a simulation has M stages, the memory demand for a single state variable is M * Nx * Ny * Nz, for multiple time-steps the memory requirement is large.

Contributions:

- Evaluate lossy compressors for use in pySDC
- Show lossy compression is effective for reducing memory overhead in pySDC
- Highlight current lossy compressors are not fast enough for inline compression on HPC applications

pySDC Pseudo Code

```
// u, fi, fe can be 4D arrays of size M x N x N x N
for t = 0:T // main time-stepping loop
  for m = 0:M // loop over all sub-time-steps at time t
    for i = 0:m
      compute using (read-only) at sub-time i
  for m = 0:M // loop over all sub time-steps at time t
    for i = 0:m
      compute using (read-only) at sub-time i
    update u at time m
    update fi and fe at time m using u at time m
```

Experimental Setup

Compressors

- SZ [1] – version 2.1.5
- ZFP [3] – version 0.5.5
- Truncation (64-bit to 32-bit precision)

Application (pySDC v3 [2])

- 1D Heat Diffusion Problem, 64^3 degrees of freedom
- Tolerance = 1e-10
- 50 time-steps
- Iterations per time-step = 20

Test Metrics

- Compression ratio
- Compression bandwidth
- Decompression bandwidth
- Amount of error introduced

Testing Methodology

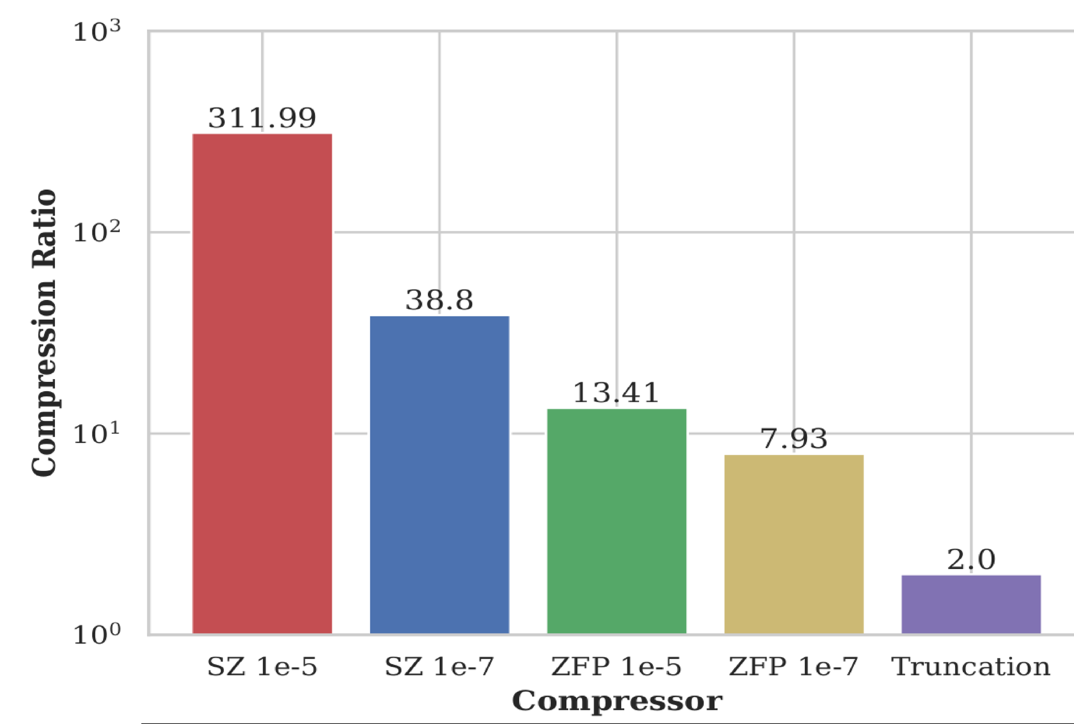
To evaluate the applicability of inline lossy compression to pySDC, we compress any time that a variable is saved/updated and decompress any time that a variable needs to be read.

Experimental Pseudo Code

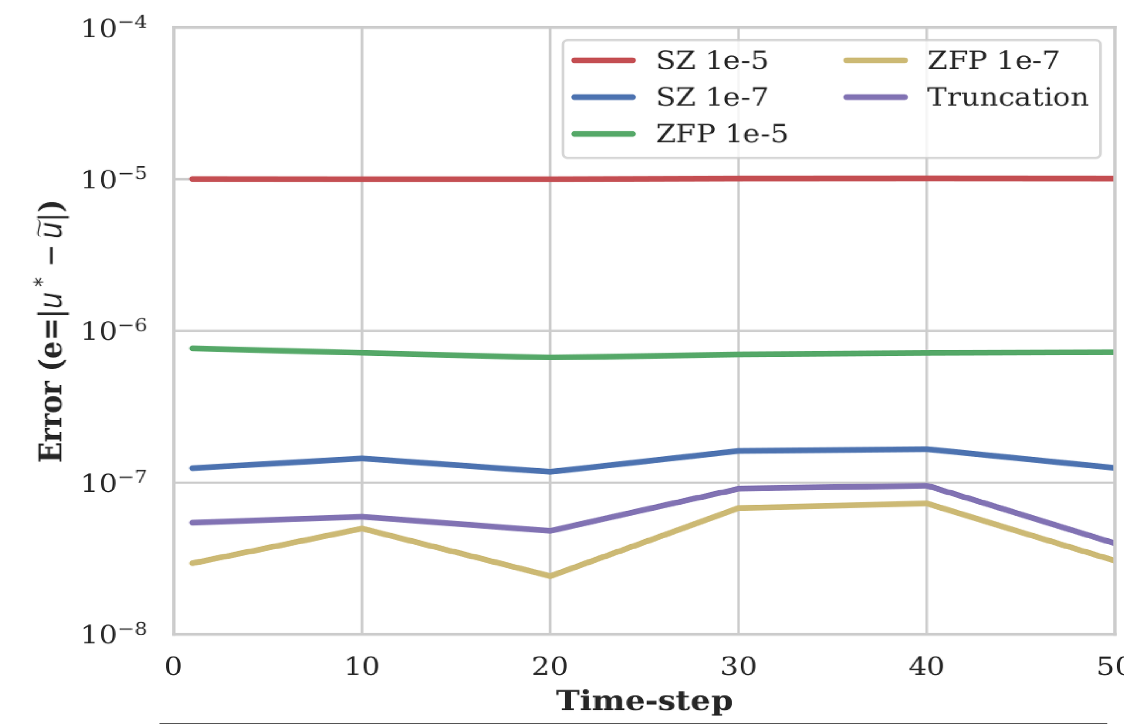
```
// u, fi, fe can be 4D arrays of size M x N x N x N
for t = 0:T // main time-stepping loop
  for m = 0:M // loop over all sub-time-steps at time t
    for i = 0:m
      compute using read (var=u) at sub-time i [decompress]
  for m = 0:M // loop over all sub time-steps at time t
    for i = 0:m
      compute using read (var=fi/fe) at sub-time i [decompress]
    update u at time m (u=new) [compress]
    update fi/fe  at time m (fi/fe=new) using u [compress]
```

Experimental Results

How do compression ratio, bandwidth, and error vary with each of the compressors tested?



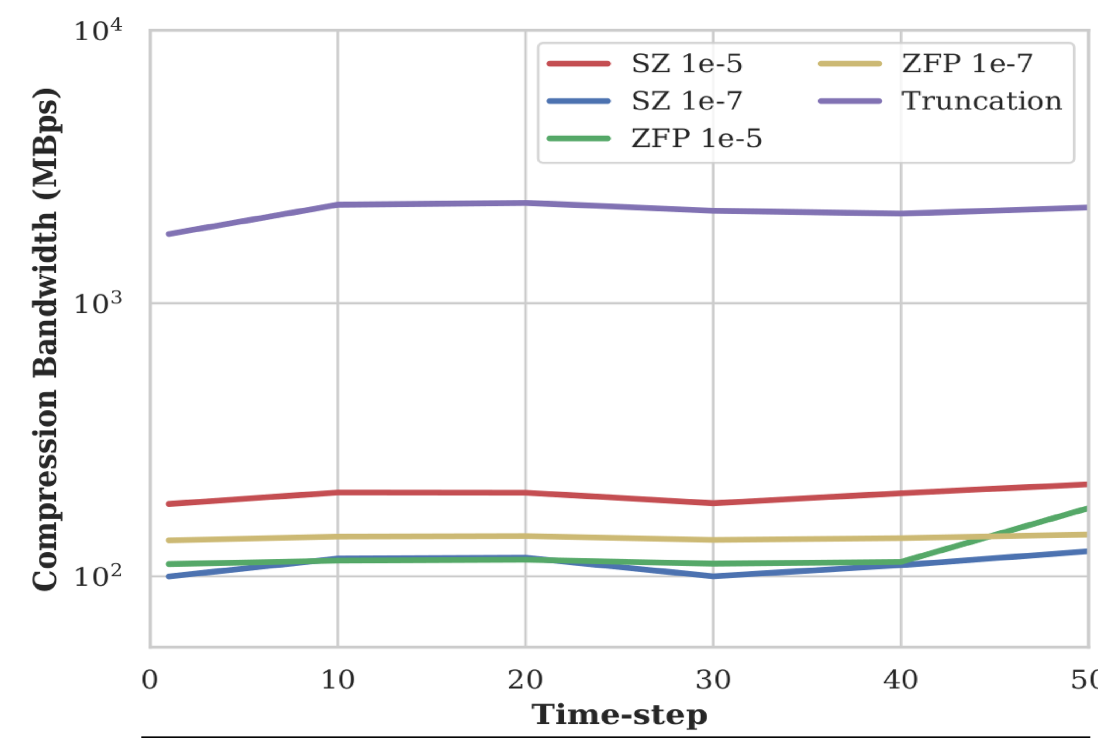
Average compression ratio across all time-steps from all compressors.



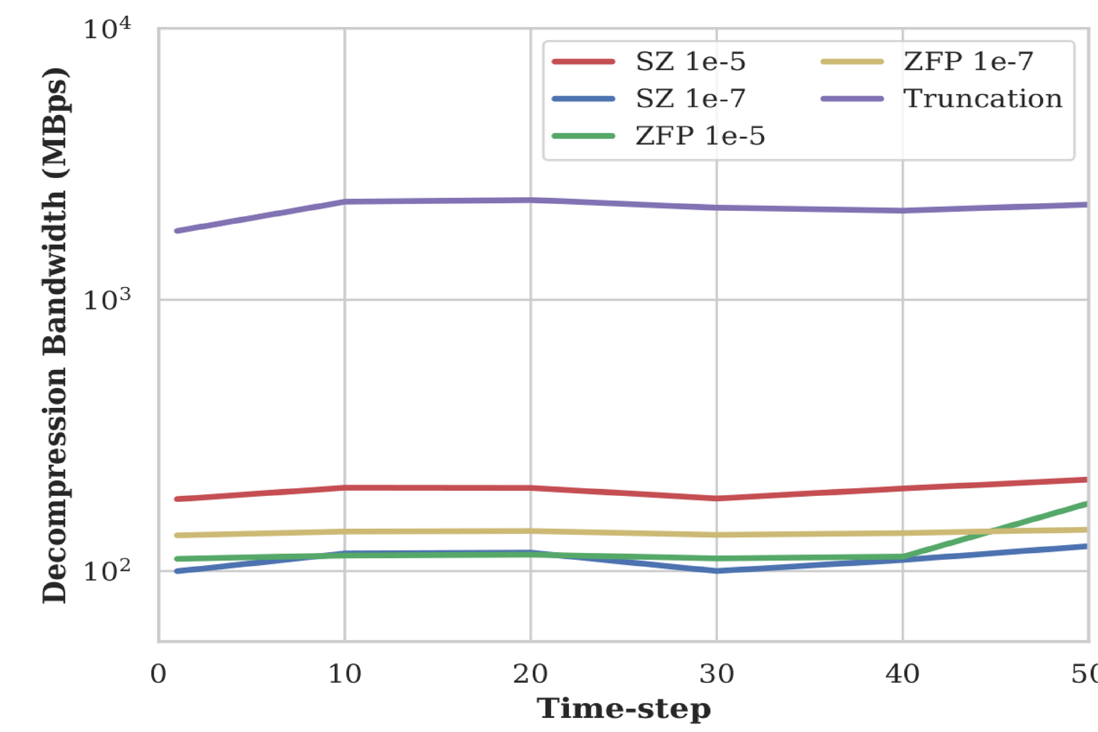
Error due to lossy compression across all time-steps from all compressors.

SZ 1e-5 provides the best compression ratio, with truncation performing the worst.

As the error bound loosens the solution becomes further from the analytic solution.



Average compression bandwidth in Mbps across all time-steps



Average decompression bandwidth in Mbps across all time-steps

Truncation has the fastest compression/decompression bandwidth by a factor of ~10. Truncation is only limited by memory bandwidth, whereas the compressor logic is slow in comparison.

Discussion from Results

The results show that lossy compression algorithms need to be lighter weight in order to be used for efficient inline lossy compression. The trade-off between lossy compressors and naive truncation is either a large reduction in data size and increase application run-time or a minimal reduction in data size and minimal increase in application run-time.

Future Work

- Abstract compression/decompression interface into a library that allows for simple inline use, as well hot-swappable compressors.
- Explore SZ temporal compression for pySDC [4].

SCAN ME

Source code (and other projects) available at:
https://github.com/donniee14

https://www.linkedin.com/in/donald-elmore/

[1]: 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
[2]: Robert Speck. pySdc - Prototyping spectral deferred corrections. To Appear in the ACM Transactions on Mathematical Software, 2019.
[3]: Peter Lindstrom. 2014. Fixed-rate compressed floating-point arrays. IEEE Transactions on Visualization and Computer Graphics 20, 12 (2014), 2674–2683
[4]: 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.

This material is based upon work supported by the National Science Foundation under Grant No. SHF-1910197.