Exploring the Feasibility of Lossy Compression for PDE Simulations

Journal Title
XX(X):1–13
©The Author(s) 2017
Reprints and permission:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Jon Calhoun[1], Franck Cappello[2], Luke N. Olson[3], Marc Snir[3], and William D. Gropp[3]

Abstract

Checkpoint restart plays an important role in high-performance computing (HPC) applications, allowing simulation runtime to extend beyond a single job allocation and facilitating recovery from hardware failure. Yet, as machines grow in size and in complexity, traditional approaches to checkpoint restart are becoming prohibitive. Current methods store a subset of the application's state and exploit the memory hierarchy in the machine. However, as the energy cost of data movement continues to dominate, further reductions in checkpoint size are needed. Lossy compression, which can significantly reduce checkpoint sizes, offers a potential to reduce computational cost in checkpoint restart. This paper investigates the use of numerical properties of PDE simulations, such as bounds on the truncation error, to evaluate the feasibility of using lossy compression in checkpointing PDE simulations. Restart from a checkpoint with lossy compression is considered for a fail-stop error in two time-dependent HPC application codes: PlasComCM and Nek5000. Results show that error in application variables due to a restart from a lossy compressed checkpoint can be masked by the numerical error in the discretization, leading to increased efficiency in checkpoint restart without influencing overall accuracy in the simulation.

Keywords

Lossy Compression, Checkpoint-restart, Exascale, Error Propagation

1 Introduction

High-performance computing (HPC) applications rely on checkpoint restart to extend execution time beyond the requested allocation time and to tolerate failures in software and in hardware during the simulation run. While the performance and memory of HPC systems continue to increase in size, one potential bottleneck in continued use of checkpoint restart is that file system bandwidth has exhibited only slow growth in current machines. For example, the current 10 petaflop machines Blue Waters at the University of Illinois and Titan at Oak Ridge National Laboratory, and the upcoming 100 petaflop machine Sierra[1] at Lawrence Livermore National Laboratory, all incorporate 1 TB/s file systems. The new machine Summit[2] at Oak Ridge National Laboratory increases the file system bandwidth to 2.5 TB/s. However, since the memory capacity in the 100 petaflop systems are approximately 5–9 times larger than in current 10 petaflop systems, scalable applications are expected to see an increase in checkpoint restart times with a similar factor. However, as HPC systems continue to increase in size and complexity, new design constraints Bergman et al. (2008); Shalf et al. (2010) such as the high cost of data movement and comparatively free cost of computation allow for new optimizations.

Exascale systems are expected to put further pressure on the checkpoint system as the mean time between failure (MTBF) is expected to decrease, thus demanding more frequent checkpointing Cappello et al. (2009). This aspect has motivated so-called multi-level checkpoint restart Moody et al. (2010); Bautista-Gomez et al. (2011) schemes, where the memory hierarchy is leveraged to reduce the time to checkpoint and to recover when a failure occurs. The addition of burst buffers Liu et al. (2012) to HPC systems offers another approach to reduce time to checkpoint. Yet, multi-level checkpointing and burst buffers do not target a reduction in checkpoint size, which is another major avenue for checkpoint time reduction. Compression of the state variables can be used to reduce the checkpoint size. The relative cost of compression decreases as computation becomes cheaper relative to the cost of data movement. For example, for processors on HPC systems, a single IEEE-754 fused multiply-add consumes three orders of magnitude less power than a DRAM memory operation Sardashti (2015). In addition, the cost of data movement increases as the memory hierarchy is traversed from registers to file systems.

Standard compression techniques are often ill suited for floating-point data Son et al. (2014) which has prompted the development of floating-point specific compressors. These are often lossless algorithms Lindstrom and Isenburg (2006); Burtscher and Ratanaworabhan (2007) and, in most cases, achieve around a 50% reduction in checkpoint size. By switching to lossy compression schemes Di and Cappello

[1]Clemson University
[2]Argonne National Laboratory
[3]University of Illinois at Urbana-Champaign

Corresponding author:
Jon Calhoun, Holcombe Department of Electrical and Computer Engineering - Clemson University
433 Calhoun Dr
104 Riggs Hall
Clemson, SC 29634.
Email: calhou3@clemson.edu

(2016); Lakshminarasimhan et al. (2011); Lindstrom (2014), higher compression ratios can be achieved.

Previous works Ni et al. (2014); Sasaki et al. (2015); Laney et al. (2013) have shown success in restarting from a lossy checkpoint, but have not focused on the relationship between the compression error and the truncation error of the numerical approximations used in the simulation. Establishing this relation is fundamental to guaranteeing correctness for users of numerical simulations.

Various break-even points for system and application level checkpointing have been analyzed for lossless compressors on HPC machines Ibtesham et al. (2012). For lossy compression, the performance benefits are realized if the overall checkpoint time is reduced. Thus, the time to compress plays a key role in analysis. Lossy compression often exhibits improved compression and decompression times in comparison to lossless schemes Lakshminarasimhan et al. (2011); Di and Cappello (2016).

This paper investigates the feasibility of using lossy compression for checkpointing PDE simulations, by interpreting the error introduced through lossy compression as numerical error and by relating it to spatial discretization error and approximation properties in the numerical methods used in simulation. The paper highlights two important PDE model problems (advection and diffusion) and two production level HPC applications (PlasComCM[3] and Nek5000[4]) and shows that the error introduced by lossy compression at restart does not increase beyond the discretization error, even in the scenario of multiple restarts in the same execution. This underscores the idea that the compression error exhibits little influence on the quality of the final result of the simulation.

Specifically, the contributions are

- expression of lossy compression error as numerical error;
- the relationship between error due to lossy compression with physical properties in the problem;
- a discussion of boundary conditions and the attenuation of compression error;
- the feasibility of lossy compression for checkpoint-restart on kernels and real applications; and
- a performance model that distinguishes the trade-off between efficiency and accuracy in lossy compression.

The rest of this paper is outlined as follows. Section 2 discusses related work in the area of compression and HPC checkpoint restart. Section 3 outlines the relationship between error introduced through lossy compression and the approximation properties of the discretization. Section 4 uses approximation bounds to investigate propagation and reduction of error through several model PDE problems. Section 5 gives an overview of two production level applications, PlasComCM and Nek5000, and presents results from using a compression tolerance that is consistent with the numerical properties of the simulation. Section 6 uses a performance model to explore efficiency of lossy compression.

2 Related Work

The applicability of lossy compression for data transfer to and from main memory has been studied for several applications, including LULESH, pF3D, and Miranda Laney et al. (2013), where simulation correctness is measured by post-simulation analysis of the physics. In this paper, we utilize the application's spatial discretization properties to construct a bound for the amount of error allowed in the simulation. The benefit is that the accuracy of the associated numerical method is known *a priori* and compression error can be dominated by the discretization truncation error. Another approach uses a two-stage scheme to compress: first lossy, then lossless Ni et al. (2014). The checkpoint size is observed to be around 15% of the original size and restarts are shown to be successful. Another observation is that errors introduced through a particular variable may result in a large propagation of error, leading to the use of lossy compression on only a subset of the checkpointed variables. Wavelet based lossy compression has been used in the climate application NICAM Sasaki et al. (2015), where it is shown that the relative error remains less than 1.5% 1500 time-steps after restart from a lossy compressed checkpoint. One disadvantages is that the error continues to grow linearly with time. An evaluation of lossy compression on the Community Earth System Model (CESM) data sets Baker et al. (2014), uses metrics such as max-norm, normalized root mean square error, and Pearson correlation coefficient to characterize how the data set chances due to compression error. However, this work does not consider restarting a simulation from a lossy compressed checkpoint. This paper complements Baker et al. (2014) by providing and evaluating a methodology of selecting lossy compression error bounds that allow simulations to be successfully restarted from lossy compressed checkpoints.

A critical characteristic of lossy compression is the *level* of lossy compression — i.e., compression factor (factor by which the data set size is reduced). Compression factors are dependant on the magnitude of error that the compressor introduces. Higher compression factors leads to larger errors, but the magnitude of acceptable compression error is problem dependent. A natural approach to determine the acceptability of a compression error is to compare the lossy state with that of an uncompressed checkpoint and verify it passes simulation validation metrics. This trial and error method can be effective, however a more direct route is compare the compression error to the approximation properties in the application. This paper explores the applicability of lossy checkpointing from the numerical perspective and provides guidance on the setting of the lossy checkpointing compression error tolerance.

Control of compression error is an important trait of a compressor. Indeed, only a strict control of the compression error allows for the association between lossy compression and the properties of the numerical methods. Lossy compressors for floating-point data sets, such as ISABELA Lakshminarasimhan et al. (2011), NUMARCK Chen et al. (2014), ZFP Lindstrom (2014), FPZIP Lindstrom and Isenburg (2006), SZ Di and Cappello (2016); Tao et al. (2017), allow for varying degrees of control. In this work, SZ is used since the compressor adheres to preset relative and absolute bounds on the tolerance on a per element basis and provides the highest compression factors. In particular, SZ-1.3 is used, although other compressors fit within the scope of this work.

3 Linking Compression Error to Numerical Error

3.1 Lossy Compressor SZ

To use SZ, the user sets the error bound by selecting the tolerance, ϵ. It is chosen so that the difference between the original values and decompressed values is bounded relatively and/or absolutely by ϵ.

SZ compresses data by predicting value $i+1$ from values at i, $i-1$, and $i-2$ using curve fitting (constant, linear, or quadratic). If the value predicted is within the prescribed tolerance, then the curve fitting function predicting $i+1$ is encoded as a codeword indicating the curve fitting method used. If the value at $i+1$ is inaccurate using a curve fitting, then an escaped codeword is encoded. The value of $i+1$ is then added to a list of other hard-to-compress data values. This list is compressed by inspecting the binary representations of the data for commonality. For multi-dimensional data, SZ explores prediction in all spatial dimensions when curvefitting. SZ has compression routines for single and double precision arrays. This paper uses the double precision compression routines during all experiments.

3.2 Method

Selecting the correct level of lossy compression is often based on trial and error. The quality of a decompressed checkpoint is ultimately application dependent and assessing the quality requires deep knowledge of the underlying physics. To this end, *a priori* bounds on the numerical discretization scheme are useful in quantifying error in specific variables of the simulation. In addition, numerical stability of the numerical method plays an important role.

This paper exploits the fact that many HPC applications approximate the solution to PDEs or ODEs. The level of accuracy given by the truncation error gives us an upper bound on compression error tolerances. A compression tolerance less than the truncation error produces results that are within a factor of the discretization error, but not bit-reproducible to the uncompressed solution. A compression tolerance greater than truncation error will add more error into the state variables, potentially impacting simulation results.

To understand truncation error consider an approximation of $u(x+h)$ by Taylor Series expansion, a method used in deriving and analyzing numerical methods to solve PDEs and ODEs:

$$u(x+h) = u(x) + u'(x)h + \frac{u''(x)h^2}{2} + \mathcal{O}(h^3) \qquad (1)$$

This Taylor Series truncates $u(x+h)$ to second order accuracy. Therefore, any error that is less than $\mathcal{O}(h^2)$ results in a numerically equivalent approximation to $u(x+h)$ according to the accuracy specified when truncating the Taylor Series.

The approach used in this paper is to select a level of lossy compression that results in an error that is less than the spatial discretization error of the problem. A similar approach to ours is presented in Fischer et al. (2017), and uses estimates on simulation accuracy to reduce communication volume in parallel ODE solvers by compressing floating-point values via truncation. In addition, Fischer et al. (2017) proves the stability and convergence of their numerical scheme with the introduced controlled perturbations during communication. In this paper, we computationally verify our methodology for selecting lossy compression error tolerances. Future work will evaluate our methodology theoretically. Given a spatial mesh size, h_x in 1D, the order of accuracy of the method is specified as $\mathcal{O}(h_x^p)$, where p is the order of discretization accuracy. A simulation that uses a second-order discretization with $h_x = 0.1$, suggests an error of approximately $e \approx ch_x^2 = c \cdot 0.01$, for some constant c.[5] This implies that a numerical approximation, u^h, and a perturbed numerical approximation $\widetilde{u}^h = u^h + \epsilon$ are close if $\epsilon < \mathcal{O}(h_x^2)$. This motivates the approach taken in this paper, where the compression tolerance ϵ is selected such that it is less than the simulation's truncation error e^h. [6] Adding compression error less than e^h creates perturbed a numercial approximation $\widetilde{u}^h$ equlivant to u^h based on the level of approximation in u^h. For problems that use adaptive mesh refinement, the compression tolerance is selected dynamically just before the application is checkpointed based on the current finest grid resolution, and is the study of future work.

Figure 1, details the selection of lossy compression error tolerances. The numerical accuracy and spatial discretization size are used to calculate a bound on the error and to produce a compatible compression error tolerance. Optionally, the error tolerance can be further refined by leveraging application-specific knowledge about the problem such as convergence properties, physical domain, and boundary conditions. After restart from a lossy compressed checkpoint, results will not be bit-reproducible to those from the standard approach, but are within the simulation's numerical accuracy.

Our method can also be used to guide the selection of tolerances to allow more error in return for higher compression factors. For example, if ϵ is the compression tolerance for a fourth-order accurate method, it is straightforward to identify a compression tolerance, γ for a second-order method. This new tolerance, γ allows for higher compression factors at the expense of larger errors in the data. Data compressed with the new tolerance γ can be interpreted as a low-order approximation to the original data. In fact, similar approaches are used in HPC applications such as Nek5000 to reduce checkpoint size. On restart, Nek5000 bootstraps itself with a low-order method which may only require information from the previous time-step. Once the program has iterated enough through time with the low-order method that a high-order method is valid, the application switches to the high-order method for the remainder of the simulation.

Algorithm 1, outlines a generic time-stepping code that utilizes lossy compression during checkpoint restart. The user computes an *a priori* bound on the simulation's truncation error e^h by using the smallest spatial mesh size and the numerical method's order of accuracy as outlined above. Next the user selects a compression error tolerance, ϵ, less than the truncation error, e^h. The simulation starts and runs as normal until a checkpoint is taken. The state variables u, p, and ρ represent the physical properties velocity,

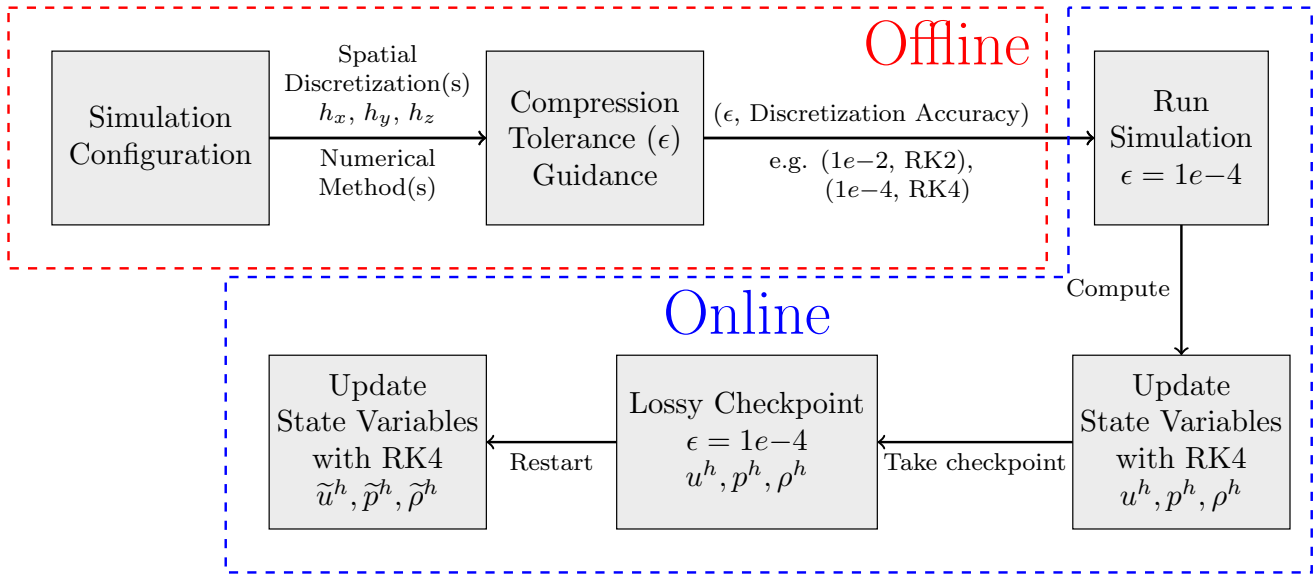


Figure 1. Overview of method to select the lossy compression error tolerance using a Runge-Kutta (RK) numerical method.

pressure, and density, respectively. These physical properties are fundamental to computational fluid dynamics codes such as PlasComCM and Nek5000. As the simulation progresses the state variables u, p, and ρ are lossy compressed before the checkpoint is written. After the checkpoint is established, computation proceeds as normal until the final time-step. If the simulation requires a restart before reaching the final time-step, then the lossy checkpoint is read and computation proceeds using the decompressed versions of the state variables.

Algorithm 1: Outline of generic time-stepping code that using lossy compressed checkpoints.

Input: e^h: User estimate of truncation error.
ϵ: User selected compression error tolerance less than e^h.
checkpoint_interval: Number of iterations between checkpoints.

```
Function main()
    if restarting == FALSE then
        u ← initVelocity()
        p ← initPressure()
        ρ ← initDensity()
    else
        u ← readDecompressVelocity()
        p ← readDecompressPressure()
        ρ ← readDecompressDensity()
    for t ← 0 to T do
        u ← updateVelocity()
        p ← updatePressure()
        ρ ← updateDensity()
        if t mod checkpoint_interval == 0 then
            lossyCompressCheckpoint(u, p, ρ, ϵ)
```

4 Understanding Compression Error Behavior on 1D Model Problems

To motivate the approach outlined in Section 3, two model PDEs are presented: 1D heat and 1D advection equations. These two model PDEs are selected to highlight the impact of physical properties on the selection of lossy compression error tolerances. The behavior of these model problems are reflected in more complex HPC applications, as observed in Section 5.

4.1 1D Heat

The 1D heat equation finds the temperature solution $u(x,t)$ for all positions x and time t along a rod with fixed length L is given by

$$u_t = u_{xx} + q(x), \quad 0 < x < 1, \tag{2}$$
$$u(0,t) = 20, u(L,t) = 50, \quad t > 0 \tag{3}$$
$$u(x,0) = f(x), \quad 0 < x < L, \tag{4}$$

where $q(x)$ is a time independent external heat source forcing term along the length of the rod. For this example, Backward Euler with second-order finite differences is used to discretize (2) yielding

$$\frac{u_j^{n+1} - u_j^n}{h_t} - \frac{u_{j+1}^{n+1} - 2u_j^{n+1} + u_{j-1}^{n+1}}{h_x^2} = q(x). \tag{5}$$

This results in a tri-diagonal linear system that is solved using a direct method at each time-step to simulate the PDE. The numerical accuracy of (5) is $\mathcal{O}(h_t; h_x^2)$, first-order accurate in time and second-order accurate in space. Thus, errors in the solution that are much smaller than $\mathcal{O}(h_t; h_x^2)$ will not impact the overall accuracy of the numerical approximation. Since restarting from a lossy compressed checkpoint adds an error into the simulation, this lossy compression error tolerance is selected so that the error is below the truncation error of the numerical scheme. For this problem, $h_x = 0.02$ implies an accuracy of $e^h = 0.02^2 =$

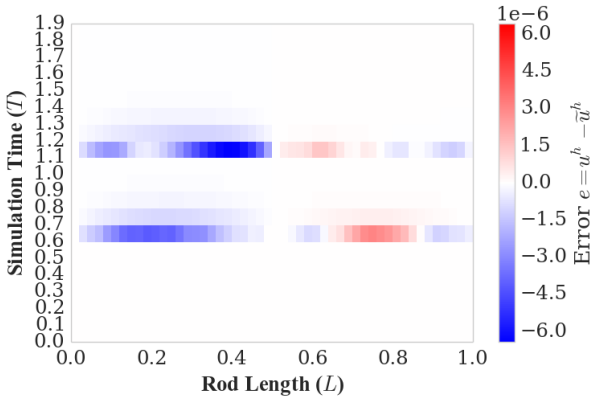


Figure 2. Error at each grid-point in a 1D heat equation between numerical solution, u^h, and a numerical solution restarted from lossy checkpoints, $\widetilde{u}^h$, at time $t = 0.6, 1.1$.

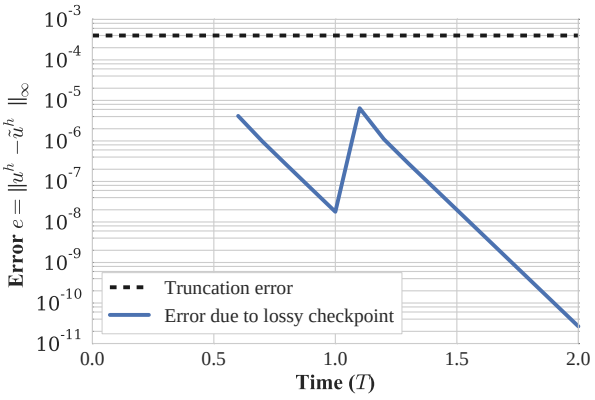


Figure 3. Maximum absolute error in the compressed numerical solution, $\widetilde{u}^h$ due to restarting from a lossy compressed checkpoint for the 1D heat equation (2).

$4e{-}4$ and a compression tolerance of $\epsilon = 1e{-}4$. Figure 2 shows error in the solution of (5) with and external heat source:

$$q(x) = \begin{cases} 100 & : x = L/2 \\ 0 & : x \neq L/2 \end{cases}$$

Figure 2 shows the error $e = u^h - \widetilde{u}^h$ in restarting the simulation at time $t = 0.6$ and $t = 1.1$ from lossy compressed checkpoints at the preceding time steps of $t = 0.5$ and $t = 1.0$, respectively. Each tick on the y-axis section represents a single time-step in the simulation. The state of the second restart is affected by two lossy compressed checkpoints. A blank (white) region in the figure denotes no error in the compressed numerical solution, $\widetilde{u}^h$, compared to an uncompressed numerical solution u^h. After restart, Figure 2 shows that error is distributed throughout the entire domain and is attenuated after a few time-steps. This is expected as this PDE models heat conduction, resulting in smooth solutions and propagation of error through the domain as time advances. In addition, with the selection of a constant temperature boundary conditions the solution will converge to a steady-state solution. This property further accelerates the aforementioned removal of error in the heat equation.

Figure 3, underscores the limited impact of lossy compression at a tolerance of $\epsilon = 1e{-}4$, which does not contribute to error above the truncation error of $e^h = 4e{-}4$. Thus, the numerical solution that depends on two restarts from lossy compressed checkpoints, $\widetilde{u}^h$, is equivalent to the original numerical solution, u^h. Although the solution is compressed to a tolerance $\epsilon = 1e{-}4$, the resulting error in $\widetilde{u}^h$ is small due to the smoothness of the solution between any two consecutive points in the domain being easily predicted by the lossy compressor.

4.2 1D Advection

A notable aspect of (2) is the attenuation of error as time evolves. The advection equation (6), does not have such a property. Instead, solutions (and error) propagate following a characteristic in the direction of flow. Any corruption in the solution remains in the wave until it reaches the boundary of the domain. In contrast, application of periodic boundaries does not allow error to escape the domain. As a result, error can accumulate with each restart from a lossy compressed checkpoint. To illustrate this point, consider a 1D advection equation with periodic boundaries:

$$\begin{aligned} u_t &= -u_x, 0 < x < 2\pi \\ u(0,t) &= u(L,t), \quad t > 0 \\ u(x,0) &= f(x), \quad 0 < x < L \end{aligned} \tag{6}$$

The equations in (6) are solved using Lax-Wendroff, which has a spatial truncation error of $\mathcal{O}(h_x^2)$, and with an initial condition of $u(x,0) = f(x) = \sin(3x)$. In this example, the lossy compression error tolerance is set at $\epsilon = 1e{-}5$ since the spatial discretization of $h_x = 0.006$ suggests accuracy up to $0.006^2 = 3.6e{-}5 = e^h$. The restarts are employed at times $t = 1.25$, 2.5, and 3.75. In Figure 4, the error history shows that with each successive restart, the error persists (error lines do not reduce in magnitude between checkpoints), increases (error lines become darker with the number of checkpoints), and is propagated though the domain in the direction of advection (error at each point shifts to the right as time increases). If periodic boundary conditions are not used, then error still exists between the two solutions due to the restart, but only until the component of the error leaves the domain.

Periodic boundary conditions allow for the accumulation of error shown in Figure 5. Figure 5, shows that the maximum error in the compressed numerical solution $\widetilde{u}^h$ at each time-step is always less than truncation error for this simulation, even when restarting multiple times. The compression error tolerance is chosen to guarantee this by staying below the truncation error of the method. This is done heuristically in a small training run by observing the max-norm over the first few time-steps after restarting and selecting a tolerance, ϵ, that yields a max-norm roughly an order of magnitude less than the truncation error, e^h, to allow for unforeseen accumulation. This can be repeated in quick secession to gauge the impact of multiple restarts.

5 Lossy Compressing Production Applications

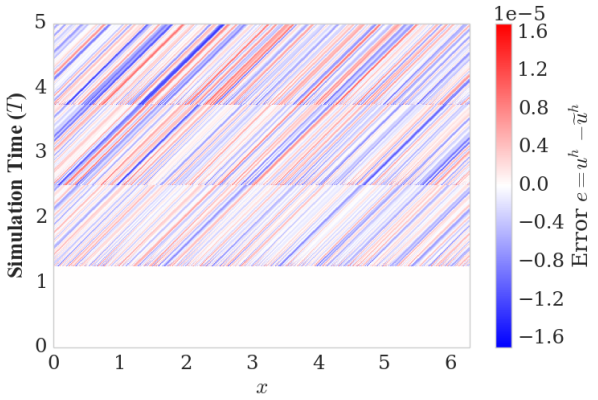


Figure 4. Error at each grid-point in a 1D advection equation between a normal numerical solution, u^h, and a numerical solution restarted from lossy checkpoints, $\widetilde{u}^h$, at times $t = 1.25$, 2.5, and 3.75.

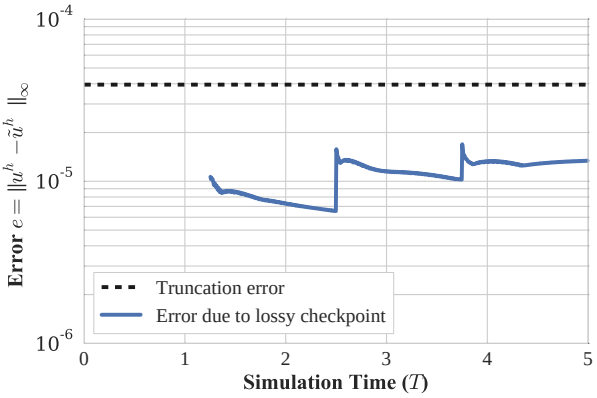


Figure 5. Maximum absolute error in a compressed numerical solution, $\widetilde{u}^h$ due to restarting from lossy compressed checkpoints for the 1D advection equation (6). Restarts from a lossy checkpoint occur at times $t = 1.25$, 2.5, and 3.75.

This section applies the approach of selecting lossy compression error tolerances from Section 3 to two production level HPC applications: PlasComCM and Nek5000. Results are collected on Blue Waters, a Cray machine managed by the National Center for Supercomputing Applications. This paper uses the XE6 nodes on the machine, which are equipped with 64GB of memory and two AMD Interlagos CPUs per node. The checkpointing routines of PlasComCM and Nek5000 are modified to lossy compress all state variables just before they are written to disk. Each state variable is compressed with SZ-1.3 Di and Cappello (2016) using relative and absolute error bounds. In production runs of both applications, checkpoints are taken every 1,000–3,000 time-steps which equates to around 1–10% of simulation time for the data sets used during testing. Simulations with higher percentage of execution time devoted to I/O exhibit a larger portion of time spent in checkpointing.

5.1 PlasComCM

The first example uses PlasComCM[7], a multi-physics plasma combustion code. The core functionality of the code targets the incompressible Navier-Stokes equations. Production runs checkpoint 1MB – 1GB per process. Here, the 2D homogeneous Euler equations are used as a model problem. Inside the domain, there is a fixed cylinder object obstructing the flow (see Figure 6). An overset mesh is placed around the cylinder to help resolve the obstructed flow. This problem is run with 4 processes each checkpointing about 1MB per process. The momentum solution of this problem after evolving for 60,020 time-steps is shown in Figure 6.

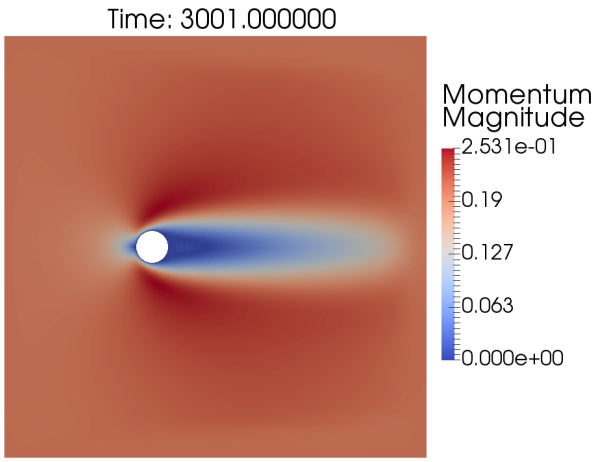


Figure 6. Momentum magnitude after 60,020 time-steps for PlasComCM. The fixed cylinder object (white circle) obstructs the flow causing momentum to slow around the object in the direction of flow (to the right).

The checkpoint file in PlasComCM consists of four state variables: density, x-momenta, y-momenta, and energy. Figure 7 shows a large difference in the compression factor depending on the selected compression error tolerance ranging from $104\times$ for $\epsilon = 1e{-}1$ to $2.2\times$ for $\epsilon = 1e{-}10$. Compression factors for small error tolerances are consistent with reported lossless compression factors Son et al. (2014). As the compression error tolerances decrease, there is a corresponding increase in compression time as shown in Figure 8.

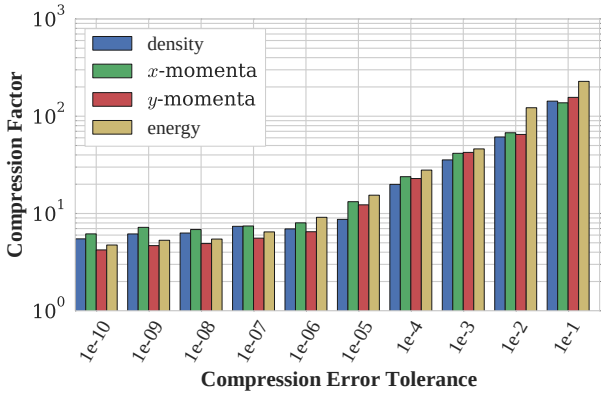


Figure 7. Compression factors for PlasComCM.

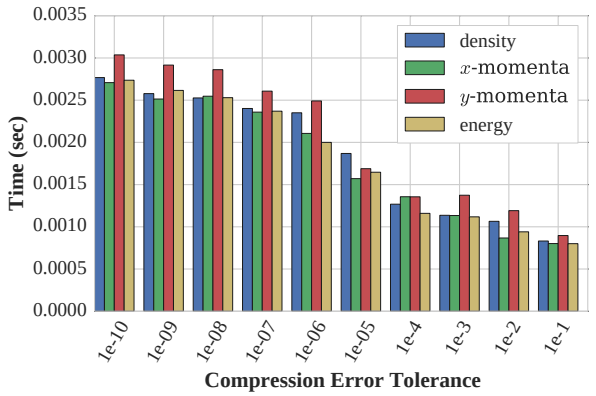


Figure 8. Compression time for PlasComCM (per process, single threaded).

The 2D flow problem is solved using fourth-order Runge-Kutta with $h_x = 0.065$ and $h_y = 0.065$. Applying the approach from Section 3, accuracy up to $e^h = 1.8e{-}5$ is assumed in both x and y. Thus, a compression error tolerance of $\epsilon < 1.8e{-}5$ is required such that error added due to compression into this does not impact simulation results. The tolerance $\epsilon = 1e{-}6$ is selected to allow for accumulation of error due to multiple restarts. This tolerance yields an average compression factor of $7\times$.

To study the propagation of error in the simulation, 75,000 time-steps are used to reach a fully developed flow. To simulate either a fail-stop failure or exceeding a time allocation at time-step 15,000, 30,000, and 45,000, the simulation restarts from a lossy compressed checkpoint. Figure 10 plots the max-norm between the numerical solution u^h and the compressed numerical solution $\tilde{u}^h$. With each checkpoint there is an increase in error in the system, but the error from lossy compression and advancing the simulation from lossy state is always less than truncation error.

Between the first and second restart (time-steps 15,000–30,000), there is a reduction in the error of all state variables. This is due to the selection of non-periodic boundary conditions, which allow the error to flow out of the domain. However, between the second and third restart (time-steps 30,000–45,000) the magnitude of error increases due to concentration of error in regions of the domain with low momentum — i.e., to the left of the cylindrical object. After the third restart, the simulation evolves another 30,000 time-steps in order to allow the error to accumulate. Over these time-steps the error remains near the compression tolerance of $\epsilon = 1e{-}6$, and there is a slow reduction as error moves from regions of low momentum to regions of higher momentum before exiting the domain.

The restart frequency used in this problem corresponds to a system MTBF of approximately 18 minutes. This MTBF is a much lower in comparison to current systems and expected exascale systems, but does highlight the nature of error propagation/accumulation for this problem. If the frequency of restarting from lossy checkpoints is high, then error accumulation above the level of truncation error can occur. In this case, a smaller compression tolerance ϵ should be used minimizing the risk of exceeding the level of truncation error.

Visualizing the error, Figure 9, in the simulation highlights two key properties about this problem's handling of error. First, the domain has non-periodic boundary conditions. This allows flow that is slightly perturbed to leave the simulation as time evolves reducing the amount of error. Any error that accumulates above or below the cylindrical object is removed from the domain as its path out of the domain is unobstructed. Second, in the same momentum error plots, Figure 9, a large spike in error is observed near the leftmost part of the fixed cylindrical object. Accumulation of error at this point is due to the near-zero momentum (cf. Figure 6). With almost no momentum to move this error though the domain, it becomes concentrated and slowly increases in magnitude as time evolves. These domain specific properties of error propagation suggests that lossy compression routines can improve by adjusting compression error tolerances to the physical properties of the domain that can accumulate or propagate error.

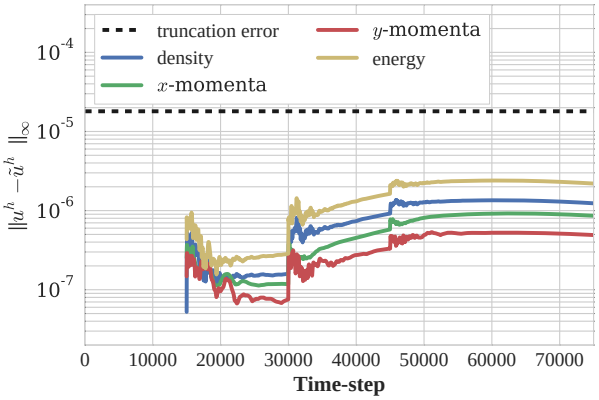


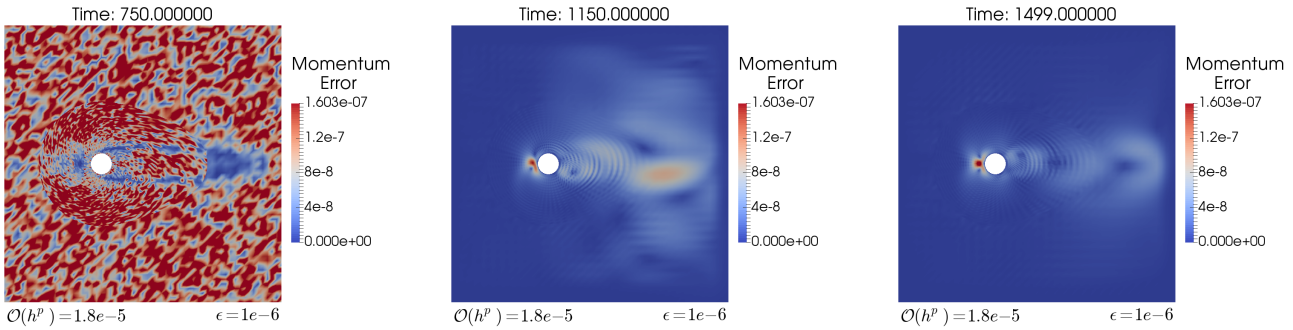
Figure 10. Max-norm between numerical solution u^h and compressed numerical solution $\tilde{u}^h$ for PlasComCM.

5.2 Nek5000

Nek5000[8] is a spectral element code developed at Argonne National Laboratory designed to simulate a variety of problem types such as heat transfer, unsteady incompressible Navier-Stokes flow, and incompressible magnetohydrodynamics. Production runs checkpoint 0.1MB – 2MB per process.

The test problem considered in this example is a 3D Navier-Stokes simulation of a channel flow with periodic boundary conditions. Figure 11 shows the solution of velocity at 30,000 time-steps. The simulation restarts twice during execution at time-step 1,500 and time-step 11,500 (MTBF of 50 min.). This test problem is run with 16 processes each checkpointing about 1 MB of data.

Compression factors for Nek5000, shown in Figure 12, are as high as $280\times$ for the x-component of velocity with compression tolerance $\epsilon = 1e{-}1$. As the compression error tolerance decreases, compression factors approach $2\times$ for all variables. The discrepancy in compression factors between the x-component of velocity and the other variables is due to the ability of SZ to more accurately predict the values. SZ adopts curve-fitting to predict future values in the array. If a section of data is easily represented with a curve-fitting



(a) Time-step 15000: Distribution of compression error due to lossy restart. Overset grid is visible in error.

(b) Time-step 23000: Error propagates though domain in direction of flow (to the right).

(c) Time-step 29980: Error accumulates in regions of low momentum, Figure 6

Figure 9. Propagation of momentum error in PlasComCM.

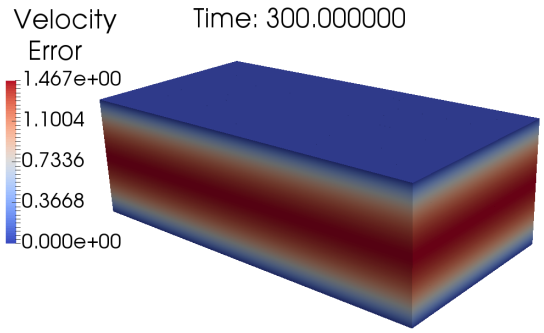


Figure 11. Magnitude of velocity after 30,000 time-steps for Nek5000.

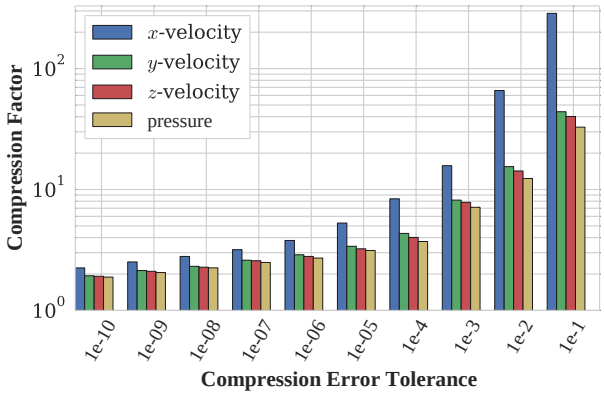


Figure 12. Smaller compression tolerances lead to higher compression factors for Nek5000.

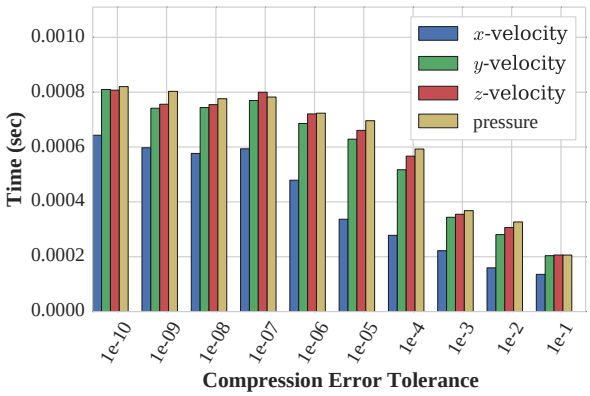


Figure 13. Time to compress (per process, single threaded) for Nek5000 increases as the error tolerance decreases. Easy to compress data compresses faster, Figure 12.

predictor, then compression factors are high. If the data is not easily predicted by curve-fitting, SZ utilizes binary analysis on the hard to compress regions. However, binary analysis is less effective at reducing data size in comparison to curve-fitting. This also suggests that compressors more directly designed for physical properties may yield improved compression factors.

Compression time of the Nek5000 state variables, shown in Figure 13, fall within two regimes that mirror the compression factor results. Variables that are easier to compress and exhibit large compression factors take less time than difficult-to-compress variables. Furthermore, selection of the lossy compressor can greatly influence the time to compress and compression factor. This highlights a need for further study into development of specialized compressors that are tailored to certain state variables in PDEs.

The 3D channel flow problem solves two linear systems at each time-step: one for pressure and one for velocity. The linear systems for pressure and velocity are solved using GMRES to a tolerance of $e^h = 1e{-}5$. The accuracy of the linear system solve suggests compression tolerances $\epsilon < 1e{-}5$ are required, due to periodic boundary conditions that do not permit error to escape the domain the compression tolerance is set at $\epsilon = 1e{-}7$, allowing for an increase in error due to multiple restarts.

Figure 14, shows the maximum error in the pressure along with the directional components of velocity between the numerical solution u^h and a compressed numerical solution

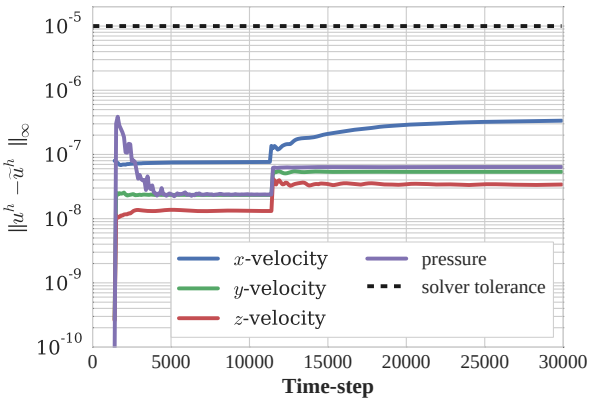


Figure 14. Max-norm between numerical solution u^h and compressed numerical solution $\tilde{u}^h$ for pressure and velocity for Nek5000. There is no error before time-step 1,500 as the simulation is not yet restarted.

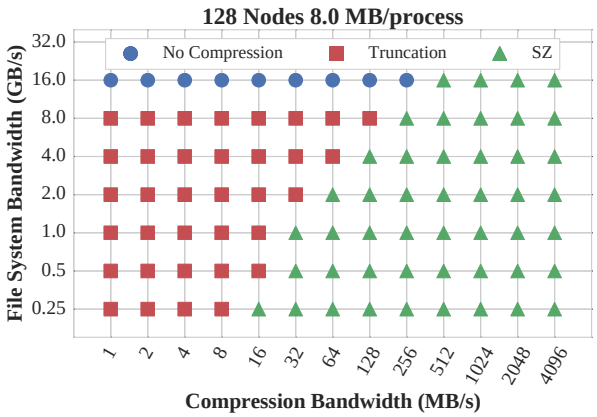


Figure 16. Fastest I/O configuration with various file system and compression bandwidths (64-bit floating-point dataset).

$\tilde{u}^h$. After restarting from a lossy checkpoint, there is an initial spike in error in the time-steps immediately following the restart, but it remains below discretization error. As time evolves, the error reduces as it migrates though the domain and approaches a steady state (due to boundary conditions). After the second restart, at time-step 11,500, there is another spike in error due to restarting from lossy state. As time evolves the error propagates though the domain, but remains less than discretization error.

Figure 15 shows the spatial location of error for the velocity solution in Figure 11 at time-step 30,000. Regions of large error correspond to regions of high velocity. The initial distribution of error after a lossy restart, Figure 15a, illustrates processor boundaries and data distribution. Even though ghost regions on other processors are needed to advance the simulation, SZ's compression algorithm only considers local data when compressing. This results in faster compression times, but results in compression artifacts around processor boundaries. Compression artifacts are magnified when the per process distribution of data is not spatially contiguous. Values that appear to be neighbors locally due to how data is stored may in fact be distant globally leading to compression artifacts and lower compression rates.

6 Modeling Time to Checkpoint

The previous sections show that leveraging the truncation error of HPC applications is an effective method of selecting a lossy compression error tolerance when compressing checkpoint files. Ultimately, however, lossy compression should be used for checkpointing only if it results in a more efficient time to checkpoint. To explore trade-offs for lossy compression, the time to checkpoint is modeled for an averaged size job run on a machine similar to Blue Waters. A simple model for time to checkpoint, T, is given by

$$T = c_b \cdot V + \frac{w_b \cdot V}{f}, \tag{7}$$

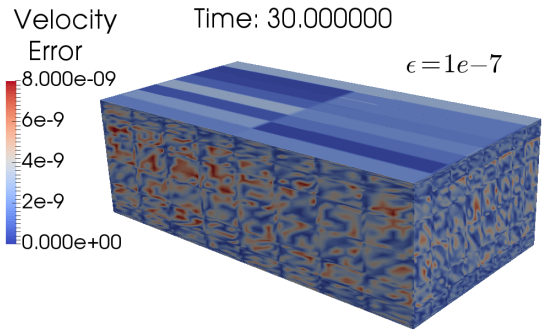
where c_b is the bandwidth of the compressor, V is the number of bytes being written per process, w_b is the file system bandwidth, and f is the compression factor of the compressor. For the case without lossy compression, $c_b = 0$ and $f = 1$. For lossy compression to be useful, the time to checkpoint using lossy compression, T_{lossy}, needs to be less than the time to checkpoint without compression, T_{reg}: $T_{lossy} < T_{reg}$. This simplifies to

$$f > \frac{w_b}{w_b - c_b}, \tag{8}$$

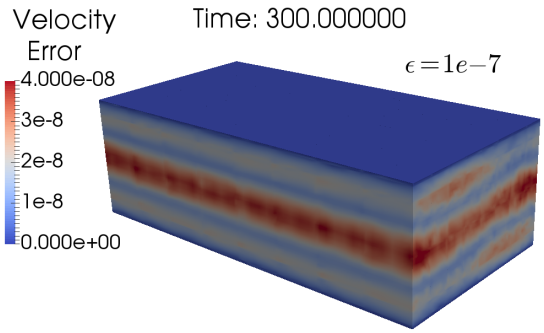
which is used to estimate the minimum compression factor for lossy compression to be viable. This model does not take into consideration the complex nature of the parallel I/O system. Despite this limitation however, the model is useful to estimate when compression improves performance. A more refined I/O model Isaila et al. (2015) that considers the parallel data exchanges in collective I/O reads/writes is parametrized for Blue Waters.

Figure 16 explores when lossy compressing checkpoint files improves performance for various combinations of compression bandwidth and file system bandwidth for a parallel job. The parallel job consists of 2048 processes on 128 nodes (1 process per core) each checkpointing 8 MB of data[9]. Lossy compression is not pipelined with file I/O in the modeling. Two lossy compressors are modeled in Figure 16. The first lossy compressor is *Truncation* which truncates 64-bit floating-point values to 32-bit values, resulting in a 2× reduction in checkpoint size. Since this compressor performs no extra computation beyond truncating each floating-point value, this compressor's performance is limited only by the available memory bandwidth. Therefore, the compression bandwidth is set as a fixed measured constant 5.4 GB/s per core. The second lossy compressor, *SZ* models SZ-1.3. The compression factor is set at 7×, and is compatible with observed factors in both PlasComCM and Nek5000. SZ's compression is more computationally intense than *Truncation*. Therefore, the compression bandwidth is a function of the checkpoint data. Easier to compress data compresses faster than more difficult to compress data. This results in variability in the observed compression bandwidth. Figure 16 scales the compression bandwidth for *SZ* from 1 MB/s to 4 GB/s.

Figure 16 shows lossy compression becomes increasingly attractive as compression bandwidth increases relative to I/O



(a) Time-step 1,500: Distribution of compression error due to lossy restart. Processor boundaries are visible.



(b) Time-step 30,000: Error in velocity magnitude for Nek5000 from Figure 11.

Figure 15. Propagation of velocity error in Nek5000.

bandwidth. Practice shows effective I/O bandwidth is much less than peak Luu et al. (2015), while compression can reach memory bandwidth. For example, PlasComCM runs of this size achieves an average aggregate file system bandwidth of around 2 GB/s on Blue Waters when checkpointing. A compression bandwidth of 64 MB/s is enough to improve the time to checkpoint. The measured compression bandwidth for PlasComCM[10] is 78 MB/s. For PlasComCM, lossy compression can improve checkpointing time by 1.8× compared to standard checkpointing.

If the computation is performed on 32-bit floating point values instead of 64-bit floating-point values, the default size of the checkpoint reduced from 8 MB/process to 4 MB/process. Moreover, this *a priori* reduction in data size decreases the performance of lossy compression in both compression factor and compression bandwidth. In our experiments, compressing 32-bit datasets results in compression factors that are roughly half of the compression factor for a 64-bit dataset. Furthermore, compression bandwidths are two-thirds of the bandwidth of compressing 64-bit datasets. Regardless, the resulting compressed checkpoint size for a 32-bit dataset is smaller than the compressed checkpoint size of 64-bit dataset. Figure 17 shows lossy compressing the 32-bit dataset improves time to checkpoint. For PlasComCM, performance can improve by 1.12×.

Using Amdahl's law, we compute the application speedup for various I/O fractions ranging from 1–90% in Figure 18. As the I/O fraction increases, the application speedup approaches the speedup obtained from using lossy compression during I/O. Improving I/O performance impacts the I/O fraction of the application. Figure 19 shows the new I/O fraction after using lossy compression. For applications that spend significant amounts of time in I/O, lossy compression is effective at reducing the I/O fraction. Moreover, as research continues to improve the performance of lossy compressors, we will see further improvement in application runtime and a reduction in the I/O faction of the application.

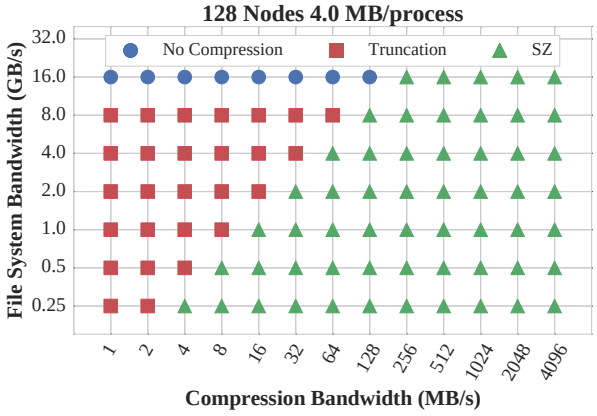


Figure 17. Fastest I/O configuration with various file system and compression bandwidths (32-bit floating-point dataset).

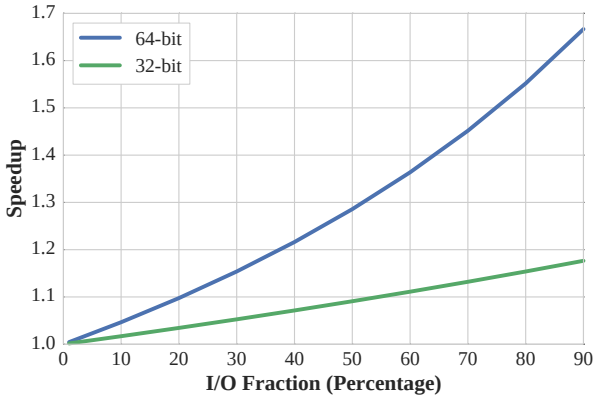


Figure 18. Application speedup due to improving the I/O with lossy compression.

7 Discussion

The previous sections explore the feasibility of setting lossy compressor error tolerances based on the numerical accuracy of the simulation. Results show that a compression

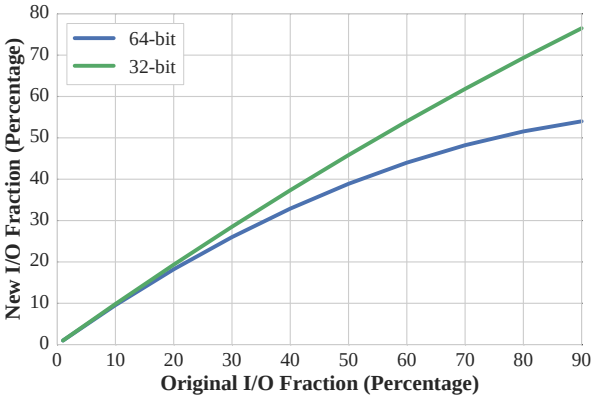


Figure 19. I/O fraction after using lossy compression.

error relative to the numerical error results in minimal impact on the simulation. Yet, there are several directions of development the would improve lossy checkpointing and increase its utility in practice.

Error Accumualtion: Error propagation after a restart and the frequency of restarts complicate the process of selecting a compression tolerance. Further analysis on the propagation of error due to lossy compression and quantifying the compression error is needed. In addition, the impact of multiple restarts and effect on the convergence rate should be explored.

Physics Based Compression: Simulations often rely on the conservation of key quantities — e.g. mass, energy — require symmetries in the domain, or compute statistics on state variables. Development of compressors that are designed specifically for certain physical properties could ensure key properties are satisfied.

Adaptive Compression: This paper uses one global compression error tolerance for all checkpoints and variables. Using different tolerances spatially and temporally for all state variables will allow new optimizations that reduce checkpoint size and mitigate the effect of error accumulation.

Lower Precision Computation: This work considers computation on double precision data. If computation is performed on lower precision, single or half precision, lossy compression can still be employed, but requires more from the lossy compressor in terms of compression factor and/or compression time. Using single or half precision reduces the data set size by $2\times$ or $4\times$ compared to double precision. Lossy compressing lower-precision datasets results in smaller compression factors and smaller compression bandwidths. However, performance improvement is possible (see Figure 17). Using lossy compression inside applications that use mixed-precision floating-point arithmetic is interesting future work because it allows for further refinement of the compression error tolerance based on accuracy needs at certain points in the computation.

8 Conclusion

Checkpoint restart is an integral part of long running HPC applications. Yet, data movement is becoming a key bottleneck on current and future machines. In response, lossy compression is a method to effectively reduce the volume of data required for a checkpoint, but at the expense of adding error into the simulation. The compression error tolerance is often difficult to determine *a priori*. This paper investigates the feasibility of using numerical truncation error as a basis for selecting the lossy compression error tolerance. First by demonstrating this use on a 1D heat and 1D advection equation, it is shown that error introduced through compression can have limited impact on the simulation. In addition, results for two production level HPC applications PlacComCM and Nek5000 also highlight its use. Results show that limiting lossy compression error to that of the discretization error allows simulations to restart from a lossy compressed checkpoint without significantly impacting the simulation. Finally, lossy compression is shown to reduce checkpoint time for compression error bounds that respect the truncation error of the application.

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

This work was sponsored by the Air Force Office of Scientific Research under grant FA9550–12–1–0478. This research is part of the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (awards OCI–0725070 and ACI–1238993) and the state of Illinois. Blue Waters is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications. This material is also based in part on work supported by the Department of Energy, National Nuclear Security Administration, under Award Number DE-NA0002374.

Notes

1. https://asc.llnl.gov/coral-info
2. https://www.olcf.ornl.gov/summit/
3. http://xpacc.illinois.edu/
4. https://nek5000.mcs.anl.gov/
5. The constant c may be large. To account for this, prior knowledge collected though verification runs and scaling tests can be used to create a bound on c. In this work, a large value of c is accounted for by selecting compression error tolerances that are less than the simulation's truncation error.
6. This work sets the tolerance based on the accuracy of the local truncation error and sets pessimistic, conservative bounds to mitigate accumulation of error during the remaining time-steps.
7. https://bitbucket.org/xpacc-dev/plascomcm
8. https://github.com/Nek5000/Nek5000
9. Increasing the per process data size causes the trade-off point between *SZ* and *Truncation* to shift to the right — i.e., *SZ* requires a faster compression bandwidth to become the most effective scheme for a constant compression factor. If the compression factor scales with dataset size, the trade-off point

between *SZ* and *Truncation* is similar to that presented in Figure 16.

10. Data compressed to $\epsilon = 1e{-}6$ with a single threaded unoptimized SZ-1.3 using relative and absolute error bounds.

References

Baker AH, Xu H, Dennis JM, Levy MN, Nychka D, Mickelson SA, Edwards J, Vertenstein M and Wegener A (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*, HPDC '14. New York, NY, USA: ACM. ISBN 978-1-4503-2749-7, pp. 203–214. DOI:10.1145/2600212.2600217. URL http://doi.acm.org/10.1145/2600212.2600217.

Bautista-Gomez L, Tsuboi S, Komatitsch D, Cappello F, Maruyama N and Matsuoka S (2011) FTI: high performance fault tolerance interface for hybrid systems. In: *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11. New York, NY, USA: ACM. ISBN 978-1-4503-0771-0, pp. 32:1–32:32. DOI:10.1145/2063384.2063427. URL http://doi.acm.org/10.1145/2063384.2063427.

Bergman K, Borkar S, Campbell D, Carlson W, Dally W, Denneau M, Franzon P, Harrod W, Hill K, Hiller J et al. (2008) Exascale computing study: Technology challenges in achieving exascale systems. *Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Tech. Rep* 15.

Burtscher M and Ratanaworabhan P (2007) High throughput compression of double-precision floating-point data. In: *Data Compression Conference, 2007. DCC '07*. pp. 293–302. DOI:10.1109/DCC.2007.44.

Cappello F, Geist A, Gropp B, Kale L, Kramer B and Snir M (2009) Toward exascale resilience. *International Journal of High Performance Computing Applications* .

Chen Z, Son SW, Hendrix W, Agrawal A, Liao Wk and Choudhary A (2014) Numarck: Machine learning algorithm for resiliency and checkpointing. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '14. Piscataway, NJ, USA: IEEE Press. ISBN 978-1-4799-5500-8, pp. 733–744. DOI:10.1109/SC.2014.65. URL http://dx.doi.org/10.1109/SC.2014.65.

Di S and Cappello F (2016) Fast error-bounded lossy HPC data compression with SZ. In: *Proceedings of the 2016 IEEE International Parallel and Distributed Processing Symposium*, IPDPS '16. Washington, DC, USA: IEEE Computer Society, pp. 730–739.

Fischer L, Götschel S and Weiser M (2017) Lossy data compression reduces communication time in hybrid time-parallel integrators. Technical Report 17-25, ZIB, Takustr.7, 14195 Berlin.

Ibtesham D, Arnold D, Bridges PG, Ferreira KB and Brightwell R (2012) On the viability of compression for reducing the overheads of checkpoint/restart-based fault tolerance. In: *2012 41st International Conference on Parallel Processing*. pp. 148–157. DOI:10.1109/ICPP.2012.45.

Isaila F, Balaprakash P, Wild SM, Kimpe D, Latham R, Ross RB and Hovland PD (2015) Collective I/O tuning using analytical and machine learning models. In: *2015 IEEE International Conference on Cluster Computing, CLUSTER 2015, Chicago, IL, USA, September 8-11, 2015*. pp. 128–137.

Lakshminarasimhan S, Shah N, Ethier S, Klasky S, Latham R, Ross R and Samatova NF (2011) *Euro-Par 2011 Parallel Processing: 17th International Conference, Euro-Par 2011, Bordeaux, France, August 29 - September 2, 2011, Proceedings, Part I*, chapter Compressing the Incompressible with ISABELA: In-situ Reduction of Spatio-temporal Data. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-642-23400-2, pp. 366–379. DOI:10.1007/978-3-642-23400-2_34. URL http://dx.doi.org/10.1007/978-3-642-23400-2_34.

Laney D, Langer S, Weber C, Lindstrom P and Wegener A (2013) Assessing the effects of data compression in simulations using physically motivated metrics. In: *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13. New York, NY, USA: ACM. ISBN 978-1-4503-2378-9, pp. 76:1–76:12. DOI:10.1145/2503210.2503283. URL http://doi.acm.org/10.1145/2503210.2503283.

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 and Isenburg M (2006) Fast and Efficient Compression of Floating-Point Data. *Visualization and Computer Graphics, IEEE Transactions on* 12(5): 1245–1250. DOI:10.1109/tvcg.2006.143. URL http://dx.doi.org/10.1109/tvcg.2006.143.

Liu N, Cope J, Carns PH, Carothers CD, Ross RB, Grider G, Crume A and Maltzahn C (2012) On the role of burst buffers in leadership-class storage systems. In: *IEEE 28th Symposium on Mass Storage Systems and Technologies, MSST 2012, April 16-20, 2012, Asilomar Conference Grounds, Pacific Grove, CA, USA*. pp. 1–11. DOI:10.1109/MSST.2012.6232369. URL http://dx.doi.org/10.1109/MSST.2012.6232369.

Luu H, Winslett M, Gropp W, Ross R, Carns P, Harms K, Prabhat M, Byna S and Yao Y (2015) A multiplatform study of I/O behavior on petascale supercomputers. In: *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '15. New York, NY, USA: ACM. ISBN 978-1-4503-3550-8, pp. 33–44. DOI:10.1145/2749246.2749269. URL http://doi.acm.org/10.1145/2749246.2749269.

Moody A, Bronevetsky G, Mohror K and Supinski BRd (2010) Design, modeling, and evaluation of a scalable multi-level checkpointing system. In: *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10. Washington, DC, USA: IEEE Computer Society. ISBN 978-1-4244-7559-9, pp. 1–11. DOI:10.1109/SC.2010.18. URL http://dx.doi.org/10.1109/SC.2010.18.

Ni X, Islam T, Mohror K, Moody A and Kale LV (2014) Lossy compression for checkpointing: Fallible or feasible? In: *Poster Session of the 2014 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '14. Washington, DC, USA: IEEE Computer

Society.

Sardashti S (2015) *Using Compression for Energy-Optimized Memory Hierarchies*. PhD Thesis, University of Wisconsin - Madison.

Sasaki N, Sato K, Endo T and Matsuoka S (2015) Exploration of lossy compression for application-level checkpoint/restart. In: *Proceedings of the 2015 IEEE International Parallel and Distributed Processing Symposium*, IPDPS '15. Washington, DC, USA: IEEE Computer Society. ISBN 978-1-4799-8649-1, pp. 914–922. DOI:10.1109/IPDPS.2015.67. URL http://dx.doi.org/10.1109/IPDPS.2015.67.

Shalf J, Dosanjh S and Morrison J (2010) Exascale computing technology challenges. In: *High Performance Computing for Computational Science–VECPAR 2010*. Springer, pp. 1–25.

Son SW, Chen Z, Hendrix W, Agrawal A, keng Liao W and Choudhary A (2014) Data compression for the exascale computing era - survey. *Supercomputing frontiers and innovations* 1(2). URL http://superfri.org/superfri/article/view/13.

Tao D, Di S, Chen Z and Cappello F (2017) Significantly improving lossy compression for scientific data sets based on multidimensional prediction and error-controlled quantization. In: *Proceedings of the 2017 IEEE International Parallel and Distributed Processing Symposium*, IPDPS '17. Washington, DC, USA: IEEE Computer Society, p. To Appear.

Author biography

Jon Calhoun received a BS in computer science from Arkansas State University in 2008, a BS in mathematics from Arkansas State University in 2008, and a Ph.D. in computer science from the University of Illinois at Urbana-Champaign in 2017. In 2017, he joined the Holcombe Department of Electrical and Computer Engineering at Clemson University as an assistant professor. His research interests lie in fault tolerance and resilience in high-performance computing (HPC) systems. In particular, he investigates soft errors and their impact on HPC applications, fault injection and analysis tools, checkpoint restart, lossy compression, and approximate computing.

Franck Cappello

Luke N. Olson

Marc Snir

William D. Gropp