algorithms

Article

# Compression Challenges in Large Scale Partial Differential Equation Solvers

Sebastian Götschel and Martin Weiser *

Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany; goetschel@zib.de
* Correspondence: weiser@zib.de

Received: 28 June 2019; Accepted: 14 September 2019; Published: 17 September 2019

**Abstract:** Solvers for partial differential equations (PDEs) are one of the cornerstones of computational science. For large problems, they involve huge amounts of data that need to be stored and transmitted on all levels of the memory hierarchy. Often, bandwidth is the limiting factor due to the relatively small arithmetic intensity, and increasingly due to the growing disparity between computing power and bandwidth. Consequently, data compression techniques have been investigated and tailored towards the specific requirements of PDE solvers over the recent decades. This paper surveys data compression challenges and discusses examples of corresponding solution approaches for PDE problems, covering all levels of the memory hierarchy from mass storage up to the main memory. We illustrate concepts for particular methods, with examples, and give references to alternatives.

**Keywords:** partial differential equation; data compression; floating point compression; lossy compression

**MSC:** 65-02; 65M60; 65N30; 68-02; 68P30

## 1. Introduction

Partial differential equations (PDEs) describe many phenomena in the natural sciences. Due to the broad spectrum of applications in physics, chemistry, biology, medicine, engineering, and economics, ranging from quantum dynamics to cosmology, from cellular dynamics to surgery planning, and from solid mechanics to weather prediction, solving PDEs is one of the cornerstones of modern science and economics. The analytic solution of PDEs in the form of explicit expressions or series representations is, however, only possible for the most simplistic cases. Numerical simulations using finite difference, finite element, and finite volume methods [1–3] approximate the solutions on spatio-temporal meshes, and are responsible for a significant part of the computational load of compute clusters and high performance computing facilities worldwide.

Achieving sufficient accuracy in large PDE systems and on complex geometries often requires huge numbers of spatial degrees of freedom (up to $10^9$) and many time steps. Thus, numerical simulation algorithms involve large amounts of data that need to be stored, at least temporarily, or transmitted to other compute nodes running in parallel. Therefore, large scale simulations face two main data-related challenges: communication bandwidth and storage capacity.

First, computing, measured in floating point operations per second (FLOPS), is faster than data transfer, measured in bytes per second. The ratio has been increasing for the last three decades, and continues to grow with each new CPU and GPU generation [4]. Today, the performance of PDE solvers is mostly limited by communication bandwidth, with the CPU cores achieving only a tiny fraction of their peak performance. This concerns the CPU-memory communication, the so-called "memory wall" [5–7], as well as inter-node communication in large distributed systems [8], and popularized the "roofline model" as a means to understand and interpret computer performance.

Second, storage capacity is usually a limited resource. Insufficient storage capacity can affect simulations in two different aspects. If needed for conducting the computation, it limits the size of problems that can be treated, and thus the accuracy of the results. Alternatively, data can spill over to the next larger and slower level of the memory hierarchy, with a corresponding impact on the simulation performance. If needed for storing the results, the capacity limits the number or resolution of simulation results that can be used for later interpretation, again affecting the accuracy of the conclusions drawn from the simulations. For both aspects, a variety of data compression methods have been proposed, both as pure software solutions and as improved hardware architecture.

The aspects of data compression that are specific for PDE solvers and the computed solutions are reviewed in the following Section 2, where we also classify compression methods proposed in the literature according to these aspects. After that, we discuss use cases of compression in PDE solvers along the memory hierarchy at prototypical levels of in-memory compression (Section 3), inter-node communication (Section 4), and mass storage (Section 5). For each use case, an example is presented in some detail along with the one or two compression methods applied, highlighting the different needs for compression on one hand and the different challenges and trade-offs encountered on the other hand.

## 2. Compression Aspects of PDE Solvers

PDE solvers have a requirement profile for compression that differs in several aspects from other widespread compression demands like text, image, video, and audio compression. The data to be compressed consists mainly of raw floating point coefficient vectors in finite difference, finite element, and finite volume methods.

### *2.1. High Entropy Data Necessitates Lossy Compression*

Usually, double precision is used for coefficient vectors in order to avoid excessive accumulation of rounding errors during computation, even if the accuracy offered by 53 mantissa bits is not required for representing the final result. Due to rounding errors, the less significant mantissa bits are essentially random, and incur a large entropy. Lossless compression methods are, therefore, not able to achieve substantial compression factors, i.e., ratios of original and compressed data sizes. Examples of lossless compression schemes tailored towards scientific floating point data are fpzip [9], FPC [10], SPDP (https://userweb.cs.txstate.edu/~burtscher/research/SPDP/) [11], Blosc (http://blosc.org/) [7], and Adaptive-CoMPI [12].

In contrast, lossy compression allows much higher compression factors, but requires a careful selection of compression error in order not to compromise the final result of the computation. In general, there is no need for the compression error to be much smaller than the discretization or truncation errors of the computation. Lossy compression schemes that have been proposed for scientific floating point data in different contexts include ISABELA (In-situ Sort-And-B-spline Error-bounded Lossy Abatement, http://freescience.org/cs/ISABELA/ISABELA.html) [13], SQE [14], zfp (https://computation.llnl.gov/projects/floating-point-compression) [15–17], SZ (https://collab.cels.anl.gov/display/ESR/SZ) 1.1 [18] and 1.4 [19,20], multilevel transform coding on unstructured grids (TCUG) [21,22], adaptive thinning (AT) [23,24] and adaptive coarsening (AC) [25,26], TuckerMPI [27,28], TTHRESH [29], MGARD [30], HexaShrink [31], and hybrids of different methods [32].

### *2.2. Data Layout Affects Compression Design Space*

One of the most important differences between PDE solvers from the compression point of view is the representation and organization of spatial data.

General-purpose floating point compression schemes essentially ignore the underlying structure and treat the values as an arbitrary sequence of values. Examples of this approach are ISABELA, SQE, and SZ-1.1. The advantage of direct applicability to any kind of discretization comes at the cost of moderate compression factors, since position-dependent correlations in the data cannot be directly exploited.

Structured Cartesian grids are particularly simple and allow efficient random access by index computations, but are limited to quasi-uniform resolutions and relatively simple geometries parametrized over cuboids. Cartesian structured data is particularly convenient for compression, since it allows the use of the Lorenzo predictor (fpzip) or its higher order variants (SZ-1.4), regular coarsening (AC), simple computation of multilevel decompositions (MGARD and HexaShrink), or exploiting tensor approaches such as factorized block transforms (zfp) or low-rank tensor approximations (TuckerMPI and TTHRESH).

If complex geometries need to be discretized or if highly local solution features need to be resolved, unstructured grids are used. Their drawback is that coefficients are stored in irregular patterns, and need to be located by lookup. Fewer methods are geared towards compression of data on unstructured grids. Examples include TCUG and AT. When storing values computed on unstructured grids, the grid connectivity needs to be stored as well. In most use cases, however, e.g., iterative solvers or time stepping, many coefficient vectors have to be compressed, but few grids do. Thus, the floating point data represents the bulk of the data to be compressed. Methods developed for compression of unstructured grid geometries can in some cases been used for storing solutions or coefficient data of PDE solvers, but are mostly tailored towards computer graphics needs. We refer to the survey [33] for 3D mesh compression, as well as [34] for grid compression in PDE applications. In addition to geometry and connectivity, recent research focuses on the compression of attribute data, such as color values or texture, taking the geometry into account [35], and using progressive compression methods [36], but without rigorous error control.

In contrast to the grid structure, the method of discretization (finite difference, finite element, or finite volume methods, or even spectral methods) is of minor importance for compression, but of course affects technical details.

PDE solvers can also benefit from compression of further and often intermediate data that arises during the solution process and may be of completely different structure. This includes in particular preconditioners (scalar quantization, mixed precision [37,38], hierarchical matrices [39]), discretized integral operators in boundary element methods (wavelets [40]), and boundary corrections in domain decomposition methods.

### *2.3. Error Metrics and Error Propagation Affect Compression Accuracy Needs*

The notion of "compression error" is not well-defined, but needs to be specified in view of a particular application. The impact of compression on data analysis has been studied empirically [30,41] and statistically [42,43]. Ideally, the compression scheme is tailored towards the desired error metric. In lack of knowledge about the application's needs, general error metrics such as pointwise error (maximum or $L^\infty$ norm) and mean squared error (MSE, $L^2$ norm) are ubiquitous, and generally used for compressor design. Integer Sobolev semi-norms and Hausdorff distances of level sets have been considered by Hoang et al. [44] for sorting level and bitplane contributions in wavelet compression. Broken Sobolev norms have been used by Whitney [45] for multilevel decimation.

If intermediate values are compressed for storage, in addition to the final simulation results compressed for analysis and archival, errors from lossy compressions are propagated through the following computations. Their impact on the final result depends very much on the type of equation that is solved and on the position in the solution algorithm where the compression errors enter. For example, inexact iterates in the iterative solution of equation systems will be corrected in later iterations, but may lead to an increase in iteration count. The impact of initial value or source term errors on the solution of parabolic equations is described by negative Sobolev norms, since high-frequency components are damped out quickly. In contrast, errors affecting the state of an explicit time stepping method for hyperbolic equations will be propagated up to the final result. Such analytical considerations are, however, qualitative, and do not allow designing quantization tolerances for meeting a quantitative accuracy requirement.

Fortunately, a posteriori error estimates are often available and can be used for controlling compression errors as well as for rate-distortion optimization. As an example, error estimators have been used in [46] for adaptive selection of state compression in the adjoint computation of gradients for optimal control.

### *2.4. Compression Speed and Complexity Follow the Memory Hierarchy*

Compression plays different roles on all levels of the memory hierarchy, depending on application, problem size, and computer architecture. Due to the speed of computation growing faster than memory, interconnect, and storage bandwidth, a multi-level memory hierarchy has developed, ranging from several memory cache levels over main memory, nonvolatile memory (NVRAM) and solid state disk burst buffers down to large storage systems [47]. Lower levels exhibit larger capacity, but less bandwidth than higher levels. The larger the data to be accessed, the deeper in the memory hierarchy it needs to be stored, and the slower is the access. Here, data compression can help to reduce the time to access the data and to exploit the available capacity on each level better. While this can, in principle, be considered and tuned for any of the many levels of current memory hierarchies, we limit the discussion to three prototypical levels: main memory, interconnects, and storage systems.

Compression of in-memory data aims at avoiding the "memory wall" and reducing the run time of the simulation (see Section 3). The available bandwidth is quite high, even if not sufficient for saturating the computed units. In order to observe an overall speedup, the overhead of compression and decompression must be very small, such that only rather simple compression schemes working on small chunks of data can be employed.

In distributed systems, compression of inter-node communication can be employed to mitigate the impact of limited network bandwidth on the run time of simulations (see Section 4). The bandwidth of communication links is about an order of magnitude below the memory bandwidth, and the messages exchanged are significantly larger than the cache lines fetched from memory, such that more sophisticated compression algorithms can be used.

Mass storage comes into play when computed solutions need to be stored for archiving or later analysis. Here, data size reduction is usually of primal interest, such that complex compression algorithms exploiting correlations, both local and global, in large data sets can be employed (see Section 5). For an evaluation of compression properties on several real-world data sets we refer to [41]. Due to the small available bandwidth and the correspondingly long time for reading or writing uncompressed data, the execution time even of complex compression algorithms can be compensated when storing only smaller compressed data sets. This aspect is relevant for the performance of out-of-core algorithms for very large problems. If compression data can be kept completely in memory, out-of-core algorithms can even be turned to in-core algorithms. A recent survey of use cases for reducing or avoiding the I/O bandwidth and capacity requirements in high performance computing, including results using mostly SZ and zfp, is given by Cappello et al. [48].

## 3. In-Memory Compression

The arithmetic density of a numerical algorithm, i.e., the number of floating point operations performed per byte that is read from or written to memory, is one of the most important properties that determines the actual performance. With respect to that quantity, the performance can be described by the roofline model [49]. It includes two main bounds, the peak performance, and the peak memory bandwidth; see Figure 1. Most PDE solvers are memory bound, in particular finite element methods working on unstructured grids and making heavy use of sparse linear algebra, but also stencil-based finite difference schemes in explicit time stepping codes.

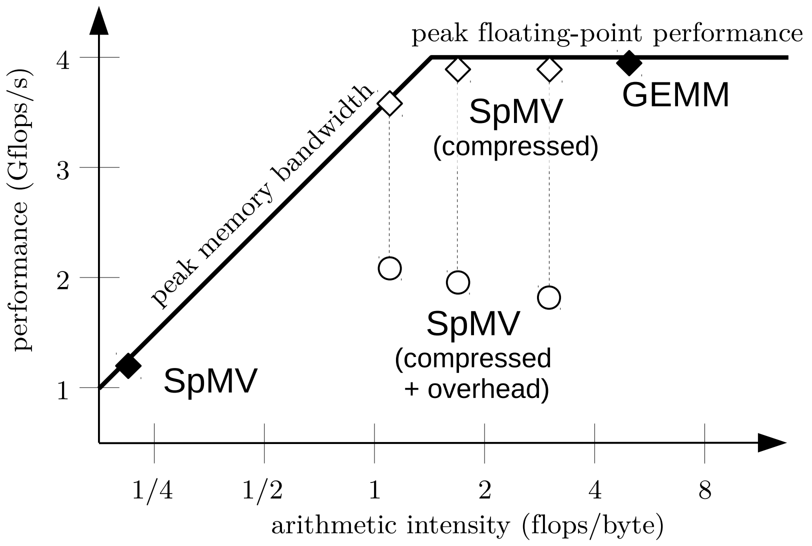

**Figure 1.** Naive roofline model showing achievable performance vs. the arithmetic intensity. Some computations, e.g., dense matrix-matrix multiplication (GEMM), perform many flops per byte fetched or written to memory, such that their execution speed is bounded by the peak floating point performance. Others, such as sparse matrix-vector multiplication (SpMV), require many bytes to be fetched from memory for each flop, and are therefore memory-bound (filled diamonds). Data compression methods for memory-bound computations can reduce the amount of data to be read or written, and therefore increase the arithmetic intensity. Different compression schemes can achieve different compression factors (empty diamonds on top) and thus different arithmetic intensities. The computational overhead of compression and decompression can, however, reduce the performance gain (empty circles bottom), depending on the complexity of the compression method used.

Performance improvements can be obtained by increasing the memory bandwidth, e.g., exploiting NUMA architectures or using data layouts favoring contiguous access patterns, or by reducing the amount of data read from and written to the memory, increasing the arithmetic intensity. Besides larger caches, data compression is an effective means to reduce the memory traffic. Due to the reduced total data size, it can also improve cache hit rates or postpone the need for paging or out-of-core algorithms for larger problem instances.

While compression can reduce the memory traffic such that the algorithm becomes compute-bound, the overhead of compression and decompression realized in the software reduces the budget available for payload flops. This is illustrated in Figure 1. Sparse matrix-vector products are usually memory bound with an arithmetic intensity of less than 0.25 flops/byte. Data compression increases the arithmetic intensity and moves the computation towards the peak floating point roofline. For illustration, compression factors between 4 and 16 are shown, representing different compression schemes. The (de)compression overhead of one to three flops per payload flop reduces the performance delivered to the original computation. Depending on the complexity of the compression scheme, and hence its computational overhead, the resulting performance can even be worse than before. This implies that to overcome the memory wall, only very fast, and therefore rather simple, compression schemes are beneficial.

Dedicated hardware support can raise the complexity barrier for compression algorithms, and several such approaches have been proposed. While completely transparent approaches [50–52] must rely on lossless compression and will therefore achieve only small compression factors on floating point data, intrusive approaches can benefit from application-driven accuracy of floating point representations [53]. They need, however, compiler support and extended instruction set architectures, and cannot be realized on commodity systems. For a survey on hardware architecture aspects for compression we refer to [54].

An important issue is the transport of compression errors through the actual computation and the influence on the results. While it is hard to envisage quantization being guided by a posteriori error estimates due to their computational overhead, careful analysis can sometimes provide quantitative worst-case bounds. Such an example, an iterative solver, is presented in the following section. Iterative solvers are particularly well-suited candidates for in-memory compression for two reasons. First, they often form the inner loops of PDE solvers and therefore cause the most memory traffic that can benefit from compression. Second, many can tolerate a considerable amount of relative error while still converging to the correct result. Thus, the compression error trade-off is not between compression factor and accuracy, but between compression factor and iteration count.

### *3.1. Scalar Quantization for Overlapping Schwarz Smoothers*

One particularly simple method of data compression is a simple truncation of mantissa and exponent bits, i.e., using IEEE 754 single precision (4 bytes) instead of double precision (8 bytes) representations [55], or even the half precision format (2 bytes) popularized by recent machine learning applications. Conversion between the different formats is done in hardware on current CPUs and integrated into load/store operations, such that the compression overhead is minimal. Consequently, using mixed precision arithmetics has been considered for a long time, in particular in dense linear algebra [56] and iterative solvers [57,58]. Depending on the algorithm's position in the roofline model, either the reduced memory traffic (Basic Linear Algebra Subroutines (BLAS) level 1/2, vector-vector and matrix-vector operations) or the faster execution of lower precision floating point operations (BLAS level 3, matrix-matrix operations) is made use of.

An important building block of solvers for elliptic PDEs of the type

$$\begin{aligned} -\operatorname{div}(\sigma\nabla u) &= f && \text{in } \Omega \\ n^T\sigma\nabla u + \alpha u &= \beta && \text{on } \partial\Omega \end{aligned}$$

is the iterative solution of the sparse, positive definite, and ill-conditioned linear equation systems $Ax = b$ arising from finite element discretizations. For this task, usually preconditioned conjugate gradient (CG) methods are employed, often combining a multilevel preconditioner with a Jacobi smoother [2]. For higher order finite elements, with polynomial ansatz order $p > 2$, the effectivity of the Jacobi smoother quickly decreases, leading to slow convergence. Then it needs to be replaced by an overlapping block Jacobi smoother $B$, with the blocks consisting of all degrees of freedom associated with cells around a grid vertex. Application of this smoother then involves a large number of essentially dense matrix-vector multiplications of moderate size:

$$B^{-1} = \sum_{\xi\in\mathcal{N}} P_\xi A_\xi^{-1} P_\xi^T. \tag{1}$$

Here, $\mathcal{N}$ is the set of grid vertices, $A_\xi$ is the symmetric submatrix of $A$ corresponding to the vertex $\xi$, and $P_\xi$ distributes the subvector entries into the global vector. Application of this smoother dominates the solver run time, and is strictly memory bound due to the large number of dense matrix-vector multiplications.

Compressed storage of $A_\xi^{-1}$ as $\tilde{A}_\xi^{-1}$ using low precision representation of its entries has been investigated in [38]. A detailed analysis reveals that the impact on the preconditioner effectivity and hence the CG convergence is determined by $\|A^{-1} - \tilde{A}^{-1}\|_2$. This suggests that a uniform quantization of submatrix entries should be preferable in view of rate-distortion optimization. Accordingly, fixed point representations have been considered as an alternative to low precision floating point representations. Moreover, the matrix entries exhibit a certain degree of correlation, which can be exploited by dividing $A_\xi^{-1}$ into square blocks to be stored independently. A uniform quantization of their entries $c$ within the block entries' range $[c_{\min}, c_{\max}]$ as $l = q(c)$ and corresponding dequantizing $\tilde{c} = q^+(l)$ is given by

$$q(c) = \begin{cases} \lfloor \frac{c-c_{\min}}{\Delta} \rfloor, & c < c_{\max} \\ 2^k - 1, & c = c_{\max}, \end{cases} \qquad q^+(l) = c_{\min} + \Delta\left(l + \frac{1}{2}\right), \tag{2}$$

with step size $\Delta = 2^{-k}(c_{\max} - c_{\min})$, providing minimal entry-wise quantization error $\Delta/2$ for the given bit budget.

Decompression can then be performed inline during application of the preconditioner, i.e., during the matrix-vector products. The computational overhead is sufficiently small as long as conversions between arithmetic data types are performed in hardware, which restricts the possible compression factors to $\{2, 4, 8\}$, for which the speedup reaches almost the compression factor; see Figure 2. With direct hardware support for finer granularity of arithmetic data types to be stored [53], an even better fit of compression errors to the desired accuracy could be achieved with low overhead.

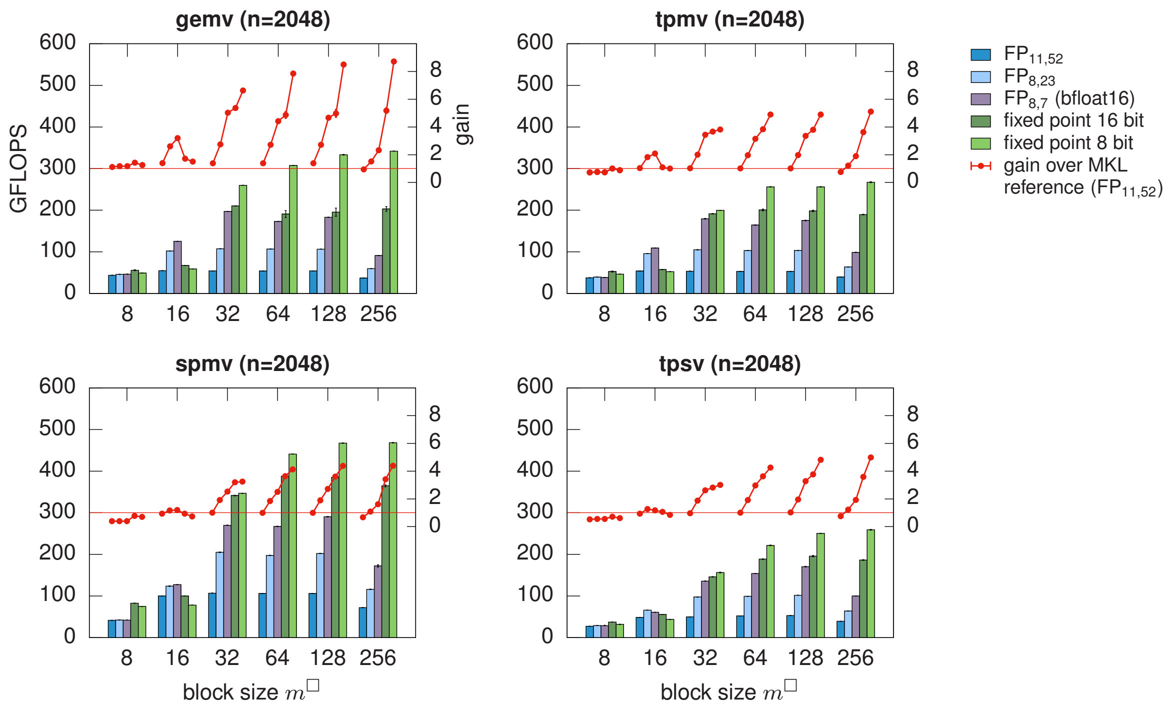

**Figure 2.** Run times of BLAS level 2 operations on 2048 × 2048-matrices for overlapping Schwarz smoothers with mixed precision. Depending on the access patterns, a speedup over the Intel Math Kernel Library (MKL) almost on par with the compression factor can be achieved [38].

The theoretical error estimates together with typical condition numbers of local matrices $A_\xi$ suggest that using a 16 bit fixed-point representation should increase the number of CG iterations by not more than 10% due to preconditioner degradation, up to an ansatz order $p = 5$. In fact, this is observed in numerical experiments, leading to a speedup of preconditioner application by a factor of up to four. With moderate ansatz order $p \leq 6$, even 8 bit fixed point representations can be used, achieving a speedup of up to six [38].

Similar results have been obtained for non-overlapping block-Jacobi preconditioners for Krylov methods applied to general sparse systems [37] and for substructuring domain decomposition methods [59].

We would like to stress that the bandwidth reduction is the driving motivation rather than the possible speedup due to faster single precision arithmetics, in contrast to BLAS level 3 algorithms. Not only is the preconditioner application memory bound such that the data size is the bottleneck, but there is also a compelling mathematical reason for performing the actual computations in high precision arithmetics: Storing the inverted submatrices $A_{\xi}^{-1}$ in low precision results in a valid, though less effective, symmetric positive definite preconditioner as long as the individual submatrices remain positive definite. Performing the dense matrix-vector products in low precision, however, will destroy the preconditioner's symmetry sufficiently to affect CG convergence, and therefore leave the well-understood theory of subspace correction methods.

### *3.2. Fixed-Rate Transform Coding*

To achieve compression factors that are higher than possible with reduced precision storage, more sophisticated and computationally more expensive approaches are required. Competing aims are high compression factors, low computational overhead, as well as transparent and random access. One particularly advanced approach is transform coding on Cartesian grids [15], which is the core of zfp. Such structured grids, though restrictive, are used in those areas of scientific computing where no complex geometries have to be respected and the limited locality of solution features does not reward the overhead of local mesh refinement.

The straightforward memory layout of the data allows considering tensor blocks of values that are compressed jointly. For 3D grids, $4 \times 4 \times 4$ blocks appear to be a reasonable compromise between locality, which is necessary for random access, and compression factors due to exploitation of spatial correlation. These blocks are transformed by an orthogonal transform. While well-known block transforms such as the discrete cosine transform [60] can be used, a special transform with slightly higher decorrelation efficiency has been developed in [15]. Such orthogonal transforms can be applied efficiently by exploiting separability and lifting scheme for factorization, i.e., applying 1D transforms in each dimension, and realizing these 1D transforms by a sequence of cheap in-place modifications. This results in roughly 11 flops per coefficient. The transform coefficients are then coded bitplane by bitplane using group testing, similar to set partitioning in hierarchical trees [61]. This embedded coding schemes allows decoding data at variable bit rate, despite the fixed-rate compression enforced by random access ability.

Despite a judicious choice of algorithm parameters, which allow an efficient integer implementation of the transform using mainly bit shifts and additions, the compression and decompression are heavily compute-bound. The effective single-core throughput, depending on the compression factor, is reported to lie around 400 MB/s, which is about a factor of ten below contemporary memory bandwidth. Here, dedicated hardware support for transform coding during read and write operations would be beneficial. In order to be flexible enough to support different coding schemes, transforms, and data sizes, however, this would need to be configurable.

In conclusion, simple and less effective compression schemes such as mixed precision approaches appear to be today's choice for addressing the memory wall in PDE computations. Complex and more effective schemes are currently of interest mainly to fit larger problems into a given memory budget. This is, however, likely to change in the future: As the hardware continues to trend to more cores per CPU socket, and thus the gap between computing performance and memory bandwidth widens, higher complexity of in-memory compression will pay off. But even then, adaptive quantization based on a posteriori error estimates for the impact of compression error will probably be out of reach.

## **4. Communication in Distributed Systems**

The second important setting in which data compression plays an increasingly important role in PDE solvers is communication in distributed systems. The ubiquitous approach for distribution is to partition the computational domain into several subdomains, which are then distributed to the different compute nodes. Due to the locality of interactions in PDEs, communication happens at the boundary shared by adjacent subdomains. A prime example are domain decomposition solvers for elliptic problems [62].

The inter-node bandwidth in such systems ranges from around 5 GB/s per link with high-performance interconnects such as InfiniBand down to shared 100 MB/s in clusters made of commodity hardware such as gigabit ethernet. This is about one to two orders of magnitude below the memory bandwidth. Consequently, distributed PDE solvers need to have a much higher arithmetic density with respect to inter-node communication than with respect to memory access. In domain decomposition methods, the volume of subdomains in $\mathbb{R}^d$ with diameter $h$ scales with $h^d$, as does the computational work per subdomain. The surface and hence communication, however, scale only with $h^{d-1}$, such that high arithmetic intensity can be achieved by using sufficiently large subdomains—which impedes on weak scaling and limits the possible parallelism. Consequently, communication can become a severe bottleneck.

Data compression has been proposed for increasing the effective bandwidth. Burtscher and Ratanaworabhan [10] consider lossless compression of floating point data streams, focusing on high throughput due to low computational overhead. Combining two predictors based on lookup tables trained online from already seen data results in compression factors on par with other lossless floating point compression schemes and general-purpose codes like GZIP, at a vastly higher throughput. Being lossless and not exploiting the spatial correlation of PDE solution values limits the compression factor, however, to values between 1.3 and 2.0, depending on the size of the lookup tables. Filgueira et al. [12] present a transparent compression layer for MPI communication, choosing adaptively between different lossless compression schemes. Again, with low redundancy of floating point data, as is characteristic for PDE coefficients, compression factors below two are achieved.

Higher compression factors can be achieved with lossy compression. As in in-memory compression, analytical a priori error estimates can provide valuable guidance on the selection of quantization tolerances. Here, however, the inter-node communication bandwidth is relatively small, such that the computational cost of a posteriori error estimators might be compensated by the additional compression opportunities they can reveal—an interesting topic for future research.

### *4.1. Inexact Parallel-in-Time Integrators*

An example of communication in distributed systems is the propagation of initial values in parallel-in-time integrators for initial value problems $\dot{u} = f(u)$, $u(t_0) = a$, in particular of hybrid parareal type [63]. Here, the initial value problem is interpreted as large equation system

$$F(U) = \begin{bmatrix} a & -u^0(t_0) & & & \\ & \dot{u}^0 - f(u^0) & & & \\ & u^0(t^1) & -u^1(t^1) & & \\ & & \dot{u}^1 - f(u^1) & & \\ & & u_1(t^2) & -u^2(t^2) & \\ & & & \ddots & \end{bmatrix} = 0 \tag{3}$$

for a set $U = (u^0, \dots, u^N)$ of subtrajectories $u^n \in C^1([t^n, t^{n+1}])$ on a time grid $t^0, t^1, \dots, t^{N+1}$. Instead of the inherently sequential triangular solve, i.e., time stepping, the system is solved by a stationary iterative method with an approximate solver $S$:

$$U_{j+1} = U_j + S(F(U_j)). \tag{4}$$

The advantage is that a large part of the approximate solver $S$ can be parallelized, by letting $S^{-1} = \mathcal{F}^{-1} + \mathcal{G}^{-1}$, where $\mathcal{G}$ is an approximation of the derivative $-F'$ on a spatial and/or temporal coarse grid and provides the global transport of information, while $\mathcal{F}$ is a block-diagonal approximation of $-F'$ on the fine grid and cares for the local reduction of fine grid residuals. The bulk of the work is done in applying the fine grid operator $\mathcal{F}^{-1}$, where all blocks can be treated in parallel. Only the coarse grid solution operator $\mathcal{G}^{-1}$ needs to be applied sequentially. If the parallelized application of

$S$ is significantly faster than computing a single subtrajectory up to fine grid discretization accuracy, reasonable parallel efficiencies above 0.5 can be achieved [64].

For a fast convergence, however, the terminal values $u^n(t^{n+1})$ have to be propagated sequentially as initial values of $u^{n+1}(t^{n+1})$ over all subintervals during each application of the approximate solver $S$. Thus, communication time can significantly affect the overall solution time [65]. Compressed communication can therefore improve the time per iteration, but may also impede on the convergence speed and increase the number of iterations. A judicious choice of compression factor and distortion must rely on error estimates and run time models.

The worst-case error analysis presented in [65] provides a bound of the type

$$\|U_j - U_*\| \le c_j^n \left( \frac{1+\Delta_C}{1-\Delta_C/\rho} \right)^{n+2}, \tag{5}$$

depending on the relative compression error $\Delta_C$, the local contraction rate $\rho$ of $S$, and factors $c_j^n$ independent of communication. This can be used to compute an upper bound on the number $J(\Delta_C)$ of iterations in dependence of the compression error. The run time of the whole computation is $T_{\text{par}} = N(t_G + t_C(\Delta_C)) + J(\Delta_C)(t_G + t_F)$, where $t_G$ is the time required for the sequential part of $S$, $t_F$ for the parallel part, and the function $t_C$ is the communication time depending on the compression error, including compression time. Minimizing $T_{\text{par}}$ can be used to optimize for $\Delta_C$, as long as the relation between $\Delta_C$ and $t_C$ is known. For finite element coefficients, most schemes will lead to $t_C \approx -c \log \Delta_C$, i.e., a communication time proportional to the bits spent per coefficient, with the proportionality factor $c$ depending on bandwidth, problem size, and efficiency of the compressor.

Variation of different parameters in this model around a nominal scenario of contemporary compute clusters as shown in Figure 3 suggest that in many current HPC situations, the expected benefit for the run time is small. The predicted run time improvement for the nominal scenario is about 5%, and does not vary much with, for example, the requested final accuracy (Figure 3, right). The pronounced dependence on smaller bandwidth shown in Figure 3, left, however, makes this approach interesting for a growing imbalance of compute power and bandwidth. Situations where this is already the case is in compute clusters with commodity network hardware and HPC systems where the communication network is nearly saturated due to concurrent communication going on, for example due to the use of spatial domain decomposition.

Indeed, using the cheap transform coding discussed in Section 4.2 below, an overall run time reduction of 10% has been observed on contemporary compute nodes connected by gigabit Ethernet [65].

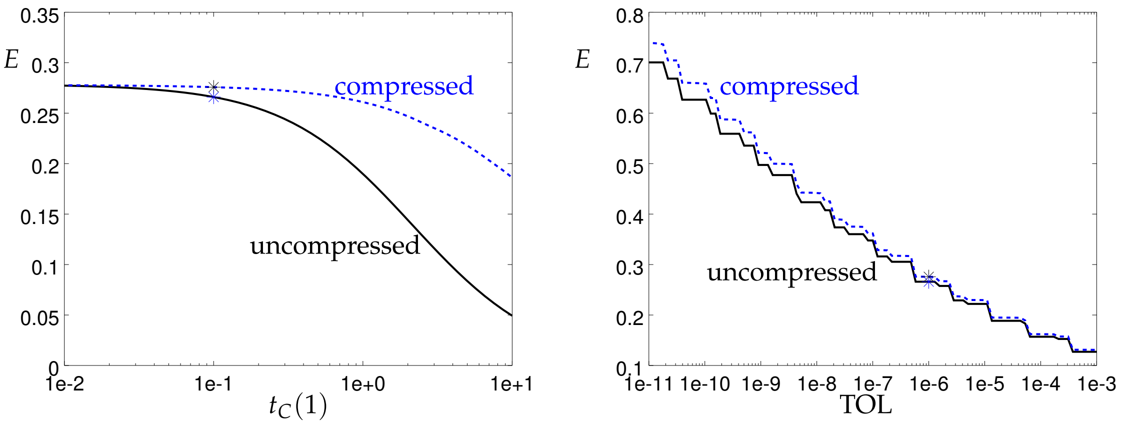

**Figure 3.** Theoretically estimated parallel efficiency $E = T_{\text{seq}}/(NT_{\text{par}})$ for variation of different parameters around the nominal scenario (marked by *). *Left:* varying communication bandwidth in terms of the communication time for uncompressed data. *Right:* varying requested tolerance.

## *4.2. Multilevel Transform Coding on Unstructured Grids for Compressed Communication*

Due to the larger gap between computing power and bandwidth, transform coding is more attractive for compressing communication in distributed systems than for in-memory compression. While methods based on Cartesian grid structures can be used for some computations, many finite element computations are performed on unstructured grids that do not exhibit the regular tensor structure exploited for designing an orthogonal transform.

An unstructured conforming simplicial grid covers the computational domain $\Omega \subset \mathbb{R}^d$ with non-overlapping simplices $T_i \in \mathcal{T}$, the corners of which meet in the grid vertices $\mathcal{N} = \{x_i \mid i = 1, \ldots, m\}$; see Figure 9 for a 2D example. The simplest finite element discretization is then with piecewise linear functions, i.e., the solution is sought in the space $V_h = \{u \in C^0(\Omega) \mid \forall T \in \mathcal{T} : u|_T \in \mathbb{P}_1\}$. The ubiquitous basis for $V_h$ is the nodal basis $(\varphi_i)_{i=1,\ldots,m}$ with the Lagrangian interpolation property $\varphi_i(x_j) = \delta_{ij}$, which makes all computations local and leads to sparse matrices.

The drawback of the nodal basis is that elliptic systems then lead to ill-conditioned matrices and slow convergence of Krylov methods. Many finite element codes therefore use hierarchies of $\ell + 1$ nested grids, resulting from adaptive mesh refinement, for efficient multilevel solvers [2]. The restriction and prolongation operators implemented for those solvers realize a frequency decomposition of the solution

$$u_h = \sum_{l=0}^{\ell} u_{h_l}, \tag{6}$$

see Figure 4 for a 1D illustration. Using the necessary subset of the nodal basis on grid level $l$ for representing $u_{h_l}$ leads to the hierarchical basis. This hierarchical basis transform allows an efficient in-place computation of optimal complexity and with low overhead, and is readily available in many finite element codes. The transform coefficients can then be quantized uniformly according to the required accuracy and entropy coded [21], e.g., using a range coder [66]. Typically, this transform coding scheme (TCUG) takes much less than 5% of the iterative solution time, see [21,34,46].

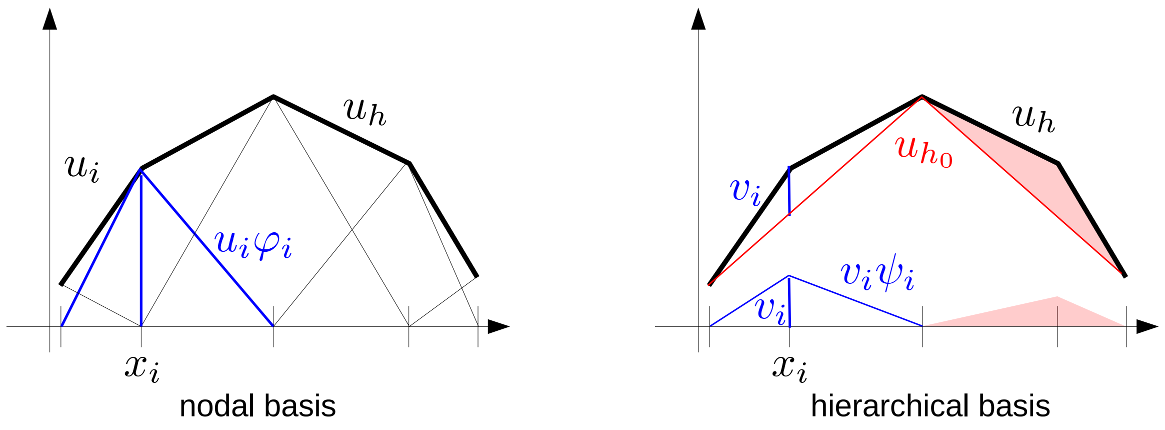

**Figure 4.** Representation of 1D linear finite element functions $u_h$ in the nodal and hierarchical basis.

A priori error estimates for compression factors and induced distortion can be derived for functions in Lebesgue or Sobolev spaces. The analysis in [21] shows that asymptotically 2.96 bits/value (in 2D, compression factor 21.6 compared to double precision) are sufficient to achieve a reconstruction error equal to $L^\infty$-interpolation error bounds for functions with sufficient regularity, as is common in elliptic and parabolic equations. In 3D, the compression factor is slightly higher; see Figure 5.

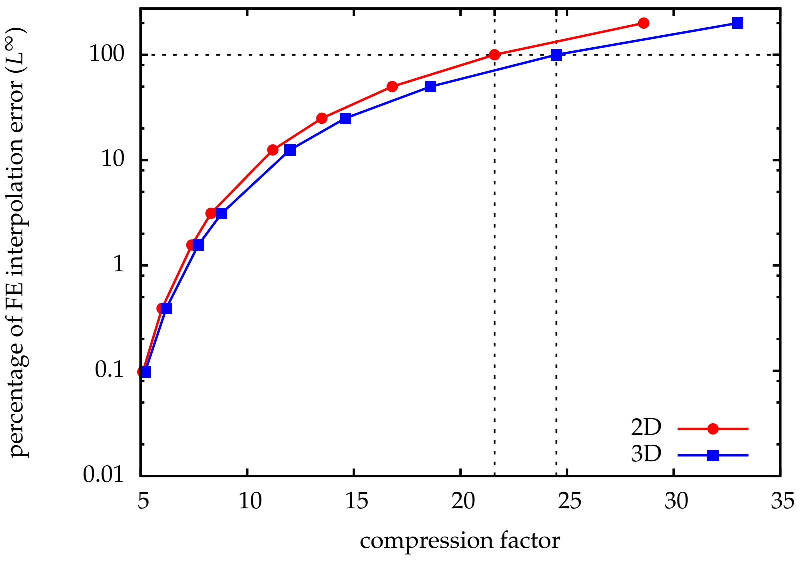

**Figure 5.** Error vs. compression factor: A priori estimates for transform coding of finite element functions with hierarchical basis transform, cf. [21].

## *4.3. Error Metrics*

An important aspect of compressor design is the norm in which to measure compression errors. While in some applications pointwise error bounds are important and the $L^\infty$-norm is appropriate, other applications have different requirements. For example, if the inexact parallel-in-time method sketched above is applied to parabolic equations, spatially high-frequency error components are quickly damped out. There, the appropriate measure of error is the $H^{-1}$-norm.

Nearly optimal compression factors for given $H^{-1}$-distortion can be achieved in TCUG by replacing the hierarchical basis transform with a wavelet transform, which can efficiently be realized by lifting [67] on unstructured mesh hierarchies. Level-dependent quantization can be used for near-optimal compression factors matching a prescribed reconstruction error in $H^s$. Rigorous theoretical norm-equivalence results are available for $|s| < 3/2$ with a rather sophisticated construction [68]. A simpler finite element wavelet construction yields norm equivalences for $-0.114 < s < 3/2$ [69], but in numerical practice it works perfectly well also for $s = -1$. A potential further improvement could be achieved by using rate-distortion theory for allocating quantization levels, as has been done for compression of quality scores in genomic sequencing data [70].

Figure 6 shows the quantization errors for the 2D test function $f(x) = \sin(12(x_0 - 0.5)(x_1 - 0.5))$ on a uniform mesh of 16,641 nodes, with a grid hierarchy of seven levels. Using a wavelet transform almost doubles the compression factor here, while keeping the same $H^{-1}$ error bound as the hierarchical basis transform [22].

A closely related aspect is the order of quantization and transform. In the considerations above, a transform-then-quantize approach has been assumed. An alternative is the quantize-then-transform sequence, which then employs an integer transform. It allows guaranteeing strict pointwise reconstruction error bounds directly, and is therefore closely linked to $L^\infty$ error concepts. In contrast, quantization errors of several hierarchical basis or wavelet coefficients affect a single point, i.e., a single nodal basis coefficient. The drawback of quantize-then-transform is that the quantization step shifts energy from low-frequency levels to high-frequency levels, leading to less efficient decorrelation if error bounds in Sobolev spaces are important. A brief discussion and numerical comparison of the two approaches for some test functions can be found in [21].

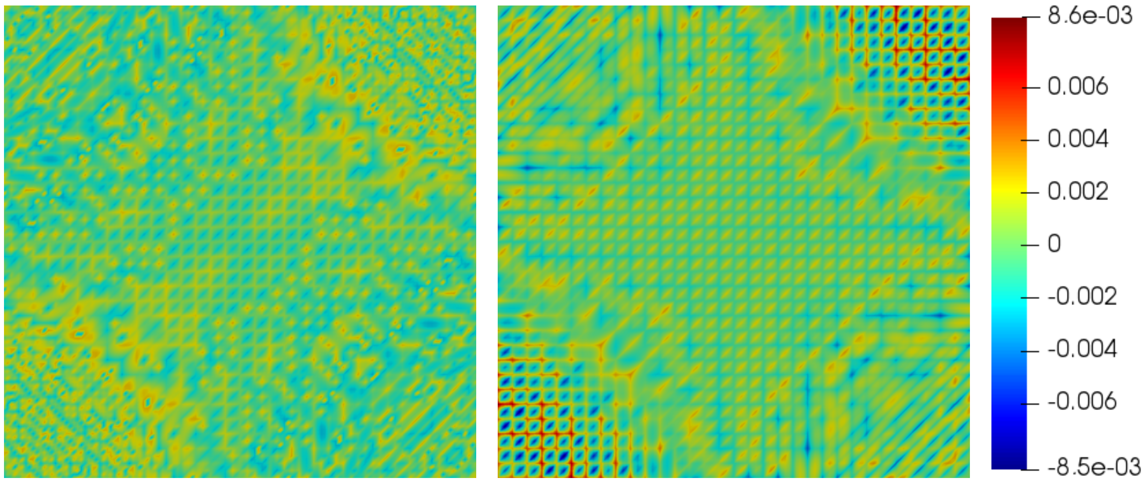

**Figure 6.** Comparison of quantization errors, i.e., error in the reconstructed solution yielding the same $H^{-1}$ error norm. Left: hierarchical basis. Right: wavelets. Using the $H^{-1}$ norm to measure the error allows larger pointwise absolute reconstruction errors compared to the $L^\infty$ error metric, thus higher compression factors.

## 5. Mass Storage

The third level in the compression hierarchy is mass storage. Often, single hard disks or complete storage systems are again slower than inter-node communication links in distributed memory systems. The significantly higher flops/byte ratio makes more sophisticated and more effective compression schemes attractive, and in particular allows employing a posteriori error estimators for a better control of the compression error tolerance. These schemes are necessarily application-specific, since they need to predict error transport into the final result, and to anticipate the intended use of reconstructions as well as the required accuracy. Examples are adjoint gradient computation in PDE-constrained optimization problems (Section 5.1) and checkpoint/restart for fault tolerance (Section 5.2).

In the extreme case, the size of the data to store is the limiting factor, and the computational effort for compression does not play a significant role. This is typically the case in solution archiving (Section 5.3).

### *5.1. Adjoint Solutions*

Adjoint, or dual, equations are important in PDE-constrained optimization problems, e.g., optimal control in electrophysiology [71] or inverse problems [72,73], and goal-oriented error estimation [74]. Consider the abstract variational problem

$$\text{find } x \in X \text{ such that } c(x;\varphi) = 0 \;\forall \varphi \in \Phi, \tag{7}$$

for a differentiable semilinear form $c : X \times \Phi \to Z$ with suitable function spaces $X, \Phi, Z$, and a quantity of interest given as a functional $J : X \to \mathbb{R}$. During the numerical solution, Equation (7) is typically only fulfilled up to a nonzero residual $r$, i.e., $c(x;\varphi) = r$. Naturally the question arises, how does the residual $r$ influence the quantity of interest $J(x)$. For instationary PDEs, answering this question leads to solving an adjoint equation backwards-in-time. As the adjoint operator and/or right-hand sides depend on the solution $x$, storage of the complete trajectory is needed, thus requiring techniques to reduce the enormous storage demand for large-scale, real-life applications. We note in passing, that, obviously, compression is not only useful for storage on disk, but can also be used in-memory, thus allowing more data to be kept in RAM and potentially avoiding disk access.

Lossy compression for computing adjoints can be done using transform coding as discussed in Section 4.2. In addition to the spatial smoothness, correlations in time can be exploited for compression. Since the stored values are only accessed backwards in time, following the adjoint equation integration direction, it is sufficient to store the state at the final time and its differences between successive time steps. This predictive coding with a constant state model, also known as delta encoding, can be efficiently implemented, requiring only to keep one additional time step in memory. Linear or higher order models can be used for prediction in time as well, but already the most simple delta encoding can significantly increase—in some cases double—the compression factor at very small computational cost. For more details and numerical examples we refer to [21,34].

Before presenting examples using lossy compression for PDE-constrained optimization and goal-oriented error estimation, let us briefly mention so-called checkpointing methods for data reduction in adjoint computations, first introduced by Volin and Ostrovskii [75], and Griewank [76]. Instead of keeping track of the whole forward trajectory, only the solution at some intermediate timestep is stored. During the integration of the adjoint equation, the required states are re-computed, starting from the snapshots, see, e.g., [77] for details. This increases the computational cost for typical settings (compression factors around 20) by two to four additional solves of the primal PDE. Moreover, due to multiple read- and write-accesses of checkpoints during the re-computations for the adjoint equation, the reduction in memory *bandwidth* requirements is significantly smaller. We refer the reader to [34] for a more detailed discussion and additional references.

### 5.1.1. PDE-Constrained Optimization

For PDE-constrained optimization, typically $X = Y \times U, x = (y, u)$ in the abstract problem (7), where the influence of the control $u$ on the state $y$ is given by the PDE. Here, $J$ is the objective to be minimized, e.g., penalizing the deviation of $y$ from some desired state. Especially in time-dependent problems, often the reduced form is considered: There, the PDE (7) is used to compute for a given control $u$ the associated (locally) unique solution $y = y(u)$. With only the control remaining as the optimization variable, the reduced problem reads $\min_u j(u)$, with $j(u) := J(y(u), u)$. Computation of the reduced gradient then leads to the adjoint equation for $p \in Z^\star$

$$c_y^\star(p; (y,u), \varphi) = -J_y((y,u), \varphi), \tag{8}$$

where $\star$ denotes the dual operator/dual function spaces, and $c_y, J_y$ are the derivatives of $c(y, u; \varphi), J(y, u)$ with respect to the $y$-component.

Exemplarily, we consider optimal control of the monodomain equations on a simple 2D unit square domain $\Omega$. This system describes the electrical activity of the heart (see, e.g., [78]), and consists of a parabolic PDE for the transmembrane voltage $v$, coupled to pointwise ODEs for the gating variable $w$ describing the state of ion channels:

$$\begin{aligned} v_t &= \operatorname{div}(\sigma \nabla v) - I_{\text{ion}}(v, w) + I_{\text{e}} && \text{in } \Omega \times (0, T) \\ w_t &= G(v, w) && \text{in } \Omega \times (0, T). \end{aligned} \tag{9}$$

Homogeneous Neumann boundary conditions are prescribed. The functions $I_{\text{ion}}(v, w)$ and $G(v, w)$ are specified by choosing a membrane model. For the optimal control problem an initial excitation in some subdomain $\Omega_{\text{exi}}$ is prescribed. The external current stimulus is $I_{\text{e}}(x, t) = \chi_{\Omega_c}(x) u(t)$, where the control $u$ is spatially constant on a control domain $\Omega_c$. Defining some observation domain $\Omega_{\text{obs}}$, the objective is given by

$$J(y, u) = \frac{1}{2} \|v\|^2_{L^2(\Omega_{\text{obs}} \times (0,T))} + \frac{\alpha}{2} \|u\|^2_{L^2(0,T)}, \tag{10}$$

i.e., we aim at damping out the excitation wave. For details, see [79]. Solution of the optimization problem with inexact Newton-CG methods and lossy compression is investigated in [46,71]; here we use the quasi-Newton method due to Broyden, Fletcher, Goldfarb, and Shanno (BFGS, see, e.g., [80,81]). For time discretization, we use a linearly implicit Euler method with fixed timestep size $dt = 0.04$. Using linear finite elements, spatial adaptivity is performed individually for state and adjoint using a hierarchical error estimator [82], with a restriction to at most 25,000 vertices in space. The adaptively refined grids were stored using the methods from [83], which reduced the storage space for the mesh to less than 1 bit/vertex (see [34]).

Lossy compression of state values, i.e., the finite element solutions $v$ and $w$, at all time steps, affects the accuracy of the reduced gradient computed by adjoint methods, and results in inexact quasi-Newton updates. Error analysis [22] shows that BFGS with inexact gradients converges linearly, if the gradient error $e_g$ in each step fulfills

$$\|e_g\| \leq \frac{\varepsilon}{\kappa(B)^{1/2}} \|\tilde{g}\| \tag{11}$$

for $\varepsilon < 1/2$. Here, $\kappa(B)$ is the condition number of the approximate Hessian $B$, and $\tilde{g}$ denotes the inexactly computed gradient. The error bound (11) allows computing adaptive compression error tolerances from pre-computed worst-case gradient error estimates analogously to [46], see [22] for details.

Figure 7 shows the progress of the optimization method. For trajectory compression, different fixed as well as the adaptively chosen quantization tolerances were used. We estimate the spatial discretization error in the reduced gradient by using a solution on a finer mesh as a reference. Up to discretization error accuracy, lossy compression has no significant impact on the optimization progress. The adaptively chosen quantization tolerances for the state values are shown in Figure 8. The resulting compression factors when using TCUG as discussed in Section 4.2 together with delta encoding in time shown as well. In the first iteration, a user-prescribed tolerance was used. The small compression factors are on one hand due to the compression on adaptively refined grids, as discussed in Section 5.1.3, and on the other hand due to overestimation of the error in the worst case error estimates. The latter is apparent from the comparison with prescribed fixed quantization tolerances (see Figure 7). To increase the performance of the adaptive method, tighter, cheaply computable error estimates are required.

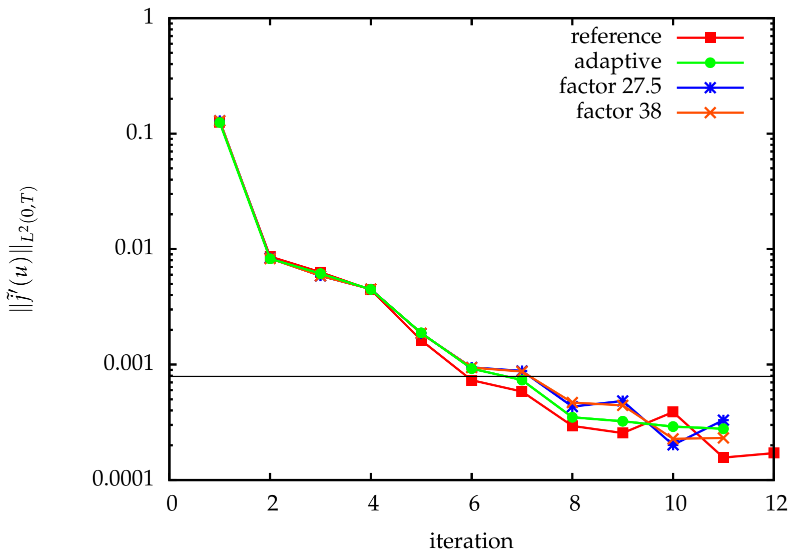

**Figure 7.** Optimization progress of Broyden, Fletcher, Goldfarb, and Shanno (BFGS) for the monodomain example (9), (10), using different quantization tolerances for the state trajectory. No delta-encoding between timesteps was used. The horizontal line shows the approximate discretization error of the reduced gradient. See also [34].

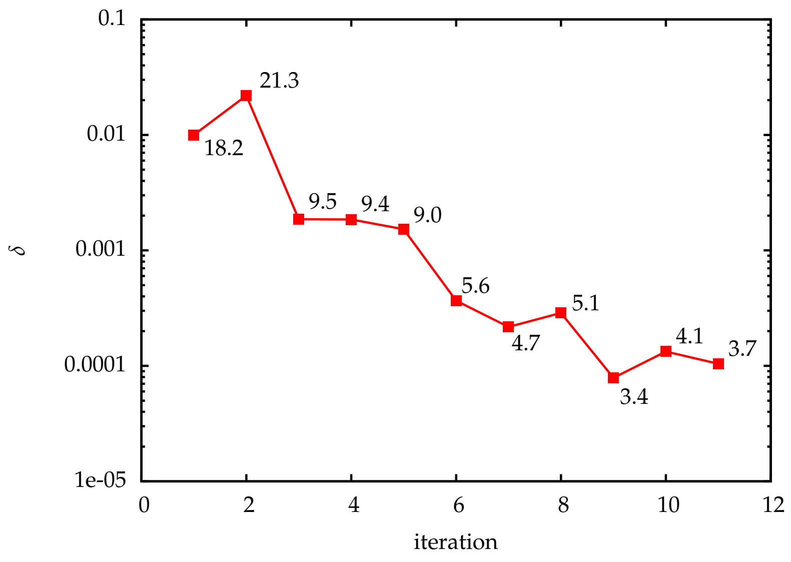

**Figure 8.** Adaptively chosen quantization tolerances $\delta$ and corresponding compression factors for the monodomain example (9), (10) using BFGS. The relatively small adaptive compression factors are due to using worst case error estimates in (11), as well as a fixed maximum mesh size.

### 5.1.2. Goal-Oriented Error Estimation

For goal-oriented adaptivity, we consider solving the PDE (7) by a Galerkin approximation,

$$\text{find } x_h \in X_h \text{ such that } c(x_h; \varphi_h) = 0 \quad \forall \varphi_h \in \Phi_h, \tag{12}$$

with suitable finite dimensional subspaces $X_h \subset X, \Phi_h \subset \Phi, Z_h \subset Z$. Here the functional $J$ measures some quantity of interest, e.g., the solution's value at a certain point, or in case of optimal control problems the objective, with the aim that

$$|J(x_h) - J(x)| \leq \epsilon. \tag{13}$$

The dual weighted residual (DWR) method [84,85] now seeks to refine the mesh used to discretize the PDE by weighting (local) residuals with information about their global influence on the goal functional $J$ [86]. These weights are computed by the dual problem

$$\text{find } p \in Z^\star \text{ such that } c_x^\star(p; x_h, \varphi) = J_x(x_h, \varphi) \quad \forall \varphi \in \Phi^\star, \tag{14}$$

which depends on the approximate primal solution $x_h$, which therefore needs to be stored.

The effect of compression on error estimation is illustrated in Figure 9. For simplicity we use a linear-quadratic elliptic optimal control problem here (Example 3b in [87]), with the objective as goal functional. Extension to time-dependent problems using the method of time layers (Rothe method) for time discretization is straightforward. Meshes were generated using weights, according to Weiser [87], for estimating the error in the reduced functional $J(y(u_h), u_h) - J(\bar{y}, \bar{u})$, as well as due to Becker et al. [85] for the all-at-once error $J(y_h, u_h) - J(\bar{y}, \bar{u})$. The compression tolerance for TCUG was chosen such that the error estimation is barely influenced, resulting in only slightly different meshes. For the final refinement step, i.e., on the finest mesh, compression resulted in data reduction by a factor of 32. Thus, at this factor the impact of lossy compression is barely noticeable in the estimated error as well as in the resulting mesh, and is much smaller than, e.g., the influence of the chosen error concept for mesh refinement. Numerical experiments using various compression techniques can be found in [88]. Instead of the ad-hoc choice of compression tolerances, a thorough analysis of the influence of compression error on the error estimators is desirable; this is, however, left for future work.

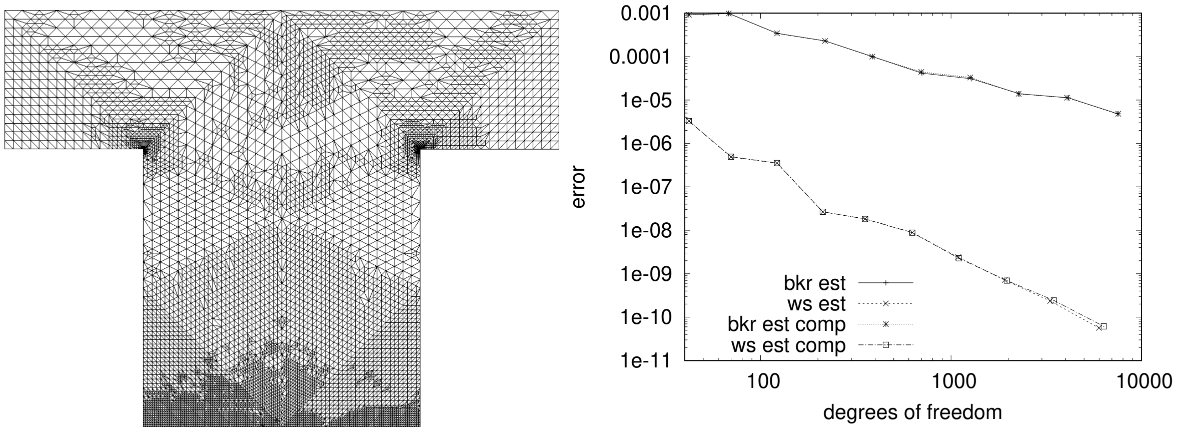

**Figure 9.** Goal-oriented error estimation: generated meshes with (left) and without (middle) compression (with compression factor up to 32) for Example 3b in [87]. Meshes were generated using weights according to Weiser [87]. Estimated errors (right) are shown for weights due [87] (ws) and to Becker et al. [85] (bkr), both with and without compression. Differences between estimated errors with and without compression are barely visible, and negligible compared to the differences between the two error concepts.

### 5.1.3. Adaptive Grid Refinement

PDE solutions often exhibit spatially local features, e.g., corner singularities or moving fronts as in the monodomain Equation (9), which need to be resolved with small mesh width. Uniform grids with small mesh width lead to huge numbers of degrees of freedom, and therefore waste computing effort, bandwidth, and storage capacity in regions where the solution is smooth. Adaptive mesh refinement, based on error estimators and local mesh refinement, has been established as an efficient means to reduce problem size and solution time, cf. [2], and acts at the same time as a decimation/interpolation-based method for lossy data compression. The construction of adaptive meshes has been explicitly exploited by Demaret et al. [23] and Solin and Andrs [24] for scientific data compression.

Interestingly, even though decimation and hierarchical transform coding both compete for the same spatial correlation of data, i.e., smoothness of functions to compress, their combination in PDE solvers can achieve better compression factors for a given distortion than each of the approaches alone. A simple example is shown in Figure 10 with compression factors for a fixed error tolerance given in Table 1. Using both, adaptive mesh refinement and transform coding, is below the product of individual compression factors, which indicates that there is in fact some overlap and competition for the same correlation budget. Nevertheless, it shows that even on adaptively refined grids there is a significant potential for data compression. The compression factor of adaptive mesh refinement as given in Table 1 is, however, somewhat too optimistic, since it only counts the number of coefficients to be stored. For reconstruction the mesh has to be stored as well. Fortunately, knowledge about the mesh refinement algorithm can be used for extremely efficient compression of the mesh structure [34].

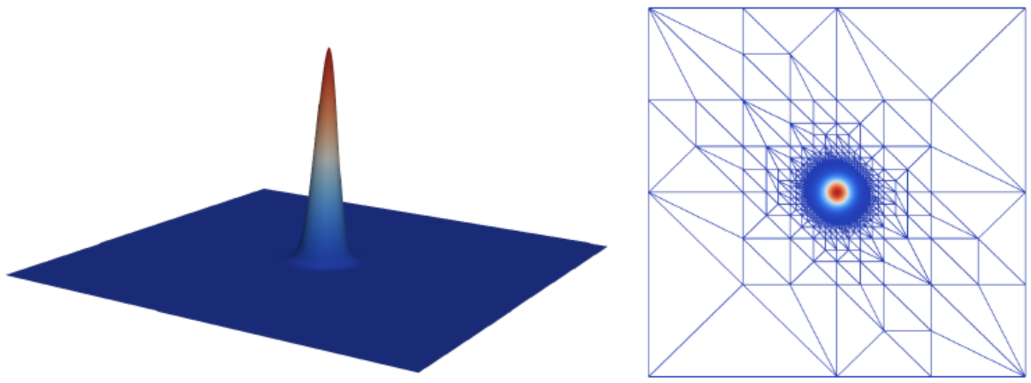

**Figure 10.** *Left:* highly local peak function. *Right:* adaptively refined mesh with 4237 vertices for minimal finite element interpolation error. A uniform grid with the same local resolution has 263,169 vertices.

**Table 1.** Compression factors for adaptive mesh refinement and transform coding of the peak function shown in Figure 10. Adaptive and uniform mesh refinement yield the same interpolation error; the same error tolerance for lossy compression was used in the two cases.

| | Double | Transform Coding |
|---|---|---|
| uniform | 1 | 54 |
| adaptive | 62 | 744 |

Another reason why adaptive mesh refinement and transform coding can be combined effectively for higher compression factors, is that the accuracy requirements for the PDE solution, and the solution storage can be very different, depending on the error propagation. Thus, the decimation by mesh adaptivity may need to retain information that the transform coder can safely neglect, allowing the latter to achieve additional compression on top of the former. For example, in adjoint gradient computation, the mesh resolution affects the accuracy of all future time steps and via them the reduced gradient, whereas the solution storage for backwards integration of the adjoint equation affects only one time step.

## 5.2. Checkpoint/Restart

In exa-scale HPC systems, node failure will be a common event. Checkpoint/restart is thus mandatory, but snapshotting for fault tolerance is increasingly expensive due to checkpoint sizes. Application-based checkpointing aims at reducing the overhead by optimizing snapshot times, i.e., when to write a checkpoint, and what to write, i.e., store only information that cannot be re-computed in a reasonable amount of time. Moreover, information should only be stored with required accuracy, which might be significantly smaller than double precision values.

Lossy compression for fault-tolerant iterative methods to solve large-scale linear systems is discussed by Tao et al. [89]. They derive a model for the computational overhead of checkpointing both with and without lossy compression, and analyze the impact of lossy checkpointing. Numerical experiments demonstrate that their lossy checkpointing method can significantly reduce the fault tolerance overhead for the Jacobi, GMRES, and CG methods.

Calhoun et al. [90] investigate using lossy compression to reduce checkpoint sizes for time stepping codes for PDE simulations. For choosing the compression tolerance they aim at an error less than the simulation's discretization error, which is estimated a priori using information about the mesh width of the space discretization and order of the numerical methods. Compression is performed using SZ [18,19]. Numerical experiments for two model problems (1D-heat and 1D-advection equations) and two HPC applications (2D Euler equations with PlacComCM, a multiphysics plasma combustion code, and 3D Navier-Stokes flow with the code Nek5000) demonstrate that restart from lossy compressed checkpoints does not significantly impact the simulation, but reduces checkpoint time.

Application-specific fault-tolerance including computing optimum checkpoint intervals [91,92] or multilevel checkpointing techniques [93] has been a research topic for many years. In order to illustrate the influence of lossy compression, we derive a simple model similar to [92], relating probability of failure and checkpoint times to the overall runtime of the application. We assume equidistant checkpoints and aim at determining the optimum number of checkpoints $n$. For this we consider a parallel-in-time simulation application (see Section 4.1) and use the notation summarized in Table 2. The overall runtime of the simulation consists of the actual computation time $T_C$, the time it takes to write a checkpoint $T_{\mathrm{CP}}$, and the restart time $T_{\mathrm{RS}}$ for $N_{\mathbf{RS}}$ failures/restarts, $T = T_C + nT_{\mathrm{CP}} + T_{\mathrm{RS}}N_{\mathrm{RS}}$. Note that here $T_C$ depends implicitly on the number of cores used. The time for restart $T_{\mathrm{RS}}$ consists of the average required re-computation from the last written checkpoint to the time of failure, here for simplicity assumed as $\frac{1}{2}\frac{T}{n}$ (see also [91]) and time to recover data structures $T_{\mathrm{R}}$. For $N$ compute cores, and probability of failure per unit time and core $p_{\mathrm{RS}}$, we get the estimated number of restarts $N_{\mathrm{RS}} = p_{\mathrm{RS}}TN$. Bringing everything together, the overall runtime amounts to

$$T(n) = T_{\mathrm{C}} + nT_{\mathrm{CP}} + T_{\mathrm{RS}}N_{\mathrm{RS}} = \frac{n\left(b - \sqrt{b^2 - \frac{2}{n}p_{\mathrm{RS}}N(T_{\mathrm{C}} + nT_{\mathrm{CP}})}\right)}{p_{\mathrm{RS}}N}, \tag{15}$$

where for brevity we use the unit-less quantity $b = 1 - T_{\mathrm{R}}p_{\mathrm{RS}}N$. As $T_{\mathrm{RS}}$ and $N_{\mathrm{RS}}$ depend on the total time $T$, this model includes failures during restart as well as multiple failures during the computation of one segment between checkpoints. Given parameters of the HPC system and the application, an optimal number of checkpoints can be determined by solving the optimization problem

$$\min_n T(n). \tag{16}$$

Taking into account the condition

$$b^2 \geq \frac{2}{n}p_{\mathrm{RS}}N(T_{\mathrm{C}} + nT_{\mathrm{CP}}) = \frac{2}{n}p_{\mathrm{RS}}NT_c + 2p_{\mathrm{RS}}NT_{\mathrm{CP}} \tag{17}$$

required for the existence of a real solution, the minimization can be done analytically, yielding

$$n_{\mathrm{opt}} = \frac{T_{\mathrm{C}}\left(2p_{\mathrm{RS}}\,N\,T_{\mathrm{CP}} + b\sqrt{2p_{\mathrm{RS}}\,N\,T_{\mathrm{CP}}}\right)}{2T_{\mathrm{CP}}\left(b^2 - 2N\,p_{\mathrm{RS}}\,T_{\mathrm{CP}}\right)}. \tag{18}$$

In this model the only influence of lossy compression is given by the time to read/write checkpoints and to recover data structures. While a simple model for time to checkpoint is given, e.g., in [90], here we just exemplarily show the influence in Figure 11, by comparing different write/read times for checkpointing.

Reducing checkpoint size by lossy compression, thus reducing $T_{\mathrm{CP}}$, $T_{\mathrm{DS}}$ has a small but noticeable effect on the overall runtime. Note that this model neglects the impact of inexact checkpoints on the re-computation time, which might increase, e.g., due to iterative methods requiring additional steps to reduce the compression error. For iterative linear solvers this is done in [89]; a thorough analysis for the example of parallel-in-time simulation with hybrid parareal methods can be done along the lines of [65].

**Table 2.** Notation used for optimal checkpointing. Where applicable, units are given in brackets.

| | | | |
|---|---|---|---|
| $n$ | number of checkpoints | $T_{\mathrm{C}}$ | time for actual computation [s] |
| $N$ | number of compute cores | $T_{\mathrm{CP}}$ | time to write/read a checkpoint [s] |
| $p_{\mathrm{RS}}$ | probability of failure per unit time and core [1/s] | $T_{\mathrm{DS}}$ | time to recover data structures [s] |
| | | $T_{\mathrm{R}}$ | recovery time $= T_{\mathrm{CP}} + T_{\mathrm{DS}}$ [s] |
| $N_{\mathrm{RS}}$ | number of restarts | $T_{\mathrm{RS}}$ | time for restart [s] |
| $T$ | overall runtime (wall clock) [s] | $b$ | $:= 1 - T_{\mathrm{R}} p_{\mathrm{RS}} N$ |

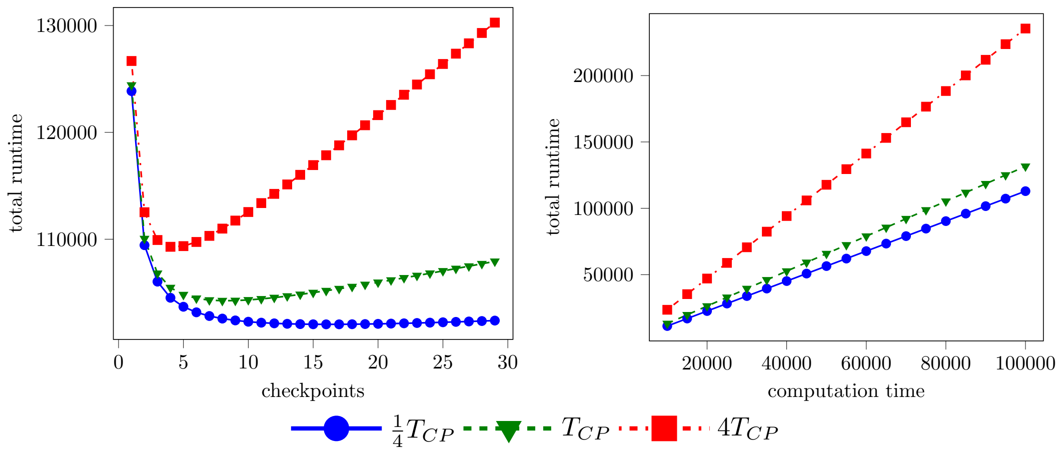

**Figure 11.** Influence of $T_{\mathrm{CP}}$, comparing the nominal scenario (green) to scenarios with $T_{\mathrm{CP}}$ reduced (blue) and increased (red) by a factor 4. *Left:* overall runtime vs. number of checkpoints for $N = 4$ and $T_C$ = 100,000 s. *Right:* overall runtime vs. actual computation time for $N = 100$. In both cases $p_{\mathrm{RS}} = 7.74 \times 10^{-7}$, $T_{\mathrm{CP}} = 245.583$ s and $T_R = 545.583$ s were used. While the runtimes were measured for the parallel-in-time solution of a 3D heat equation on the HLRN-III Cray XC30/XC40 supercomputer (www.hlrn.de), the probability of failure was determined from HLRN-III logfiles.

## 5.3. Postprocessing and Archiving

Storage and postprocessing of results from large-scale simulations requires techniques to efficiently handle the vast amount of data. Assuming that computation requires higher accuracy than needed for postprocessing (due to, e.g., error propagation and accumulation over time steps), lossy compression will be beneficial here as well. In the following, we briefly present examples from three application areas.

### 5.3.1. Crash Simulation

Simulation is a standard tool in the automotive industry, e.g., for the simulation of crash tests. For archiving data generated by the most commonly used crash simulation programs, the lossy compression code FEMzip (https://www.sidact.com/femzip0.html); FEMzip is a registered trademark of Fraunhofer Gesellschaft, Munich. [94] achieves compression factors of 10–20 [95,96], depending on the prescribed error tolerance. More recently, correlations between different simulation results were exploited by using a predictive principal component analysis to further increase the compression factor, reporting an increase by a factor of 4.4 for a set of 14 simulation results [96].

### 5.3.2. Weather and Climate

Today, prediction of weather and climate is one major use of supercomputing facilities, with a tremendous amount of data to be stored (In 2017, ECMWF's data archive grew by about 233 TB per day https://www.ecmwf.int/en/computing/our-facilities/data-handling-system [97]). Thus, using compression on the whole I/O system (main memory, communication, storage) can significantly affect performance [98]. Typically, ensemble simulations are used, allowing to exploit correlations.

Three different approaches are investigated in [97]. The most successful one takes forecast uncertainties into account, such that higher precision is provided for less uncertain variables. Naturally, applying data compression should not introduce artifacts, or change, e.g., statistics of the outputs of weather and climate models. The impact of lossy compression on several postprocessing tasks is investigated in [42,43], e.g., whether artifacts due to lossy compression can be distinguished from the inherent variability of climate and weather simulation data. Here, avoiding smoothing of the data due to compression via transform coding can be important, favouring quantize-then-transform methods or simpler truncation approaches like fpzip [9].

### 5.3.3. Computational Fluid Dynamics

Solution statistics of turbulent flow are used in [99] to assess error tolerances for lossy compression. Data reduction is performed by spatial transform coding with the discrete Legendre transform [100], which matches the spectral discretization on quadrilateral grids. For turbulence statistics of turbulent flow through a pipe, Otero et al. [99] reported a reduction of 98% for an admissible $L^2$ error level of 1%, which is the order of the typical statistical uncertainty in the evaluation of turbulence quantities from direct numerical simulation data.

## 6. Conclusions

Data compression methods are valuable tools for accelerating PDE solvers, addressing larger problems, or archiving computed solutions. Due to floating-point data being compressed, only lossy compression can be expected to achieve reasonable compression factors; this matches perfectly with the fact that PDE solvers incur discretization and truncation errors. An important aspect is to model and predict the impact of quantization or decimation errors on the ultimate use of the computed data, in order to be able to achieve high compression factors while meeting the accuracy requirements. This requires construction and use of fast and accurate a posteriori error estimators complementing analytical estimates, and remains one of the challenging future research directions.

Meeting such application-dependent accuracy requirements calls for problem-specific approaches to compression, e.g., in the form of transforms where the quantization of transform coefficients directly corresponds to the compression error in relevant spatially weighted Sobolev norms. Discretization-specific approaches that are able to exploit the known structure of spatial data layout, e.g., Cartesian grids or adaptively refined mesh hierarchies, are also necessary for achieving high compression factors.

Utility and complexity of such methods are largely dictated by their position in the memory hierarchy. Sophisticated compression schemes are available and regularly used for reducing the required storage capacity when archiving solutions. On the other hand, accelerating PDE solvers by data compression is still in the active research phase, facing the challenge that computational overhead for compression can thwart performance gains due to reduced data transmission time. Thus, simpler compression schemes dominate, in particular when addressing the memory wall. Consequently, only a moderate, but nevertheless consistent, benefit of compression has been shown in the literature.

The broad spectrum of partially contradicting requirements faced by compression schemes in PDE solvers suggests that no single compression approach will be able to cover the need, and that specialized and focused methods will increasingly be developed—a conclusion also drawn in [48].

The trend of growing disparity between computing power and bandwidth, which could be observed during the last three decades and will persist for the foreseeable future of hardware development, means that data compression methods will only become more important over time. Thus, we can expect to see a growing need for data compression in PDE solvers in the coming years.

**Author Contributions:** Conceptualization, methodology, software, writing–original draft preparation, writing–review and editing, project administration, visualization: S.G. and M.W.; supervision, funding acquisition: M.W.

**Funding:** This work has been partially supported by the German Ministry for Eduction and Research (BMBF) under project grant 01IH16005 (HighPerMeshes).

**Acknowledgments:** We thank Florian Wende for implementing mixed-precision preconditioners, Alexander Kammeyer for implementation and testing of checkpoint/restart, and Thomas Steinke for many helpful discussions.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Strikwerda, J. *Finite Difference Schemes and Partial Differential Equations*; SIAM: Philadelphia, PA, USA, 2007.
2. Deuflhard, P.; Weiser, M. *Adaptive Numerical Solution of PDEs*; de Gruyter: Berlin, Germany, 2012.
3. Zienkiewicz, O.; Taylor, R.; Zhu, J. *The Finite Element Method*; Elsevier Butterworth-Heinemann: Oxford, UK, 2005.
4. McCalpin, J. *Memory Bandwidth and System Balance in HPC Systems*. 2016. Available online: https://sites.utexas.edu/jdm4372/2016/11/22/sc16-invited-talk-memory-bandwidth-and-system-balance-in-hpc-systems/ (accessed on 16 September 2019).
5. McCalpin, J. Memory Bandwidth and Machine Balance in Current High Performance Computers. *IEEE Tech. Comm. Comput. Archit. (TCCA) Newsl.* **1995**, *2*, 19–25.
6. McKee, S. Reflections on the memory wall. In Proceedings of the Conference Computing Frontiers, Ischia, Italy, 14–16 April 2004; pp. 162–167.
7. Alted, F. Why Modern CPUs Are Starving and What Can Be Done about It. *Comp. Sci. Eng.* **2010**, *12*, 68–71. [CrossRef]
8. Reed, D.; Dongarra, J. Exascale computing and big data. *Comm. ACM* **2015**, *58*, 56–68. [CrossRef]
9. Lindstrom, P.; Isenburg, M. Fast and Efficient Compression of Floating-Point Data. *IEEE Trans. Vis. Comput. Graphics* **2006**, *12*, 1245–1250. [CrossRef] [PubMed]
10. Burtscher, M.; Ratanaworabhan, P. FPC: A High-Speed Compressor for Double-Precision Floating-Point Data. *IEEE Trans. Comp.* **2009**, *58*, 18–31. [CrossRef]
11. Claggett, S.; Azimi, S.; Burtscher, M. SPDP: An Automatically Synthesized Lossless Compression Algorithm for Floating-Point Data. In Proceedings of the IEEE 2018 Data Compression Conference, Snowbird, UT, USA, 27–30 March 2018; p. 17936904.
12. Filgueira, R.; Singh, D.; Carretero, J.; Calderón, A.; García, F. Adaptive-Compi: Enhancing MPI-Based Applications' Performance and Scalability by using Adaptive Compression. *Int. J. High Perform. Comput. Appl.* **2011**, *25*, 93–114. [CrossRef]
13. Lakshminarasimhan, S.; Shah, N.; Ethier, S.; Ku, S.H.; Chang, C.; Klasky, S.; Latham, R.; Ross, R.; Samatova, N. ISABELA for effective in situ compression of scientific data. *Concurr. Comp. Pract. Exp.* **2013**, *25*, 524–540, [CrossRef]
14. Iverson, J.; Kamath, C.; Karypis, G. Fast and Effective Lossy Compression Algorithms for Scientific Datasets. In *Euro-Par 2012 Parallel Processing*; Kaklamanis, C., Papatheodorou, T., Spirakis, P., Eds.; Springer: Berlin, Germany, 2012; pp. 843–856.
15. Lindstrom, P. Fixed-Rate Compressed Floating-Point Arrays. *IEEE Trans. Vis. Comp. Graphics* **2014**, *20*, 2674–2683. [CrossRef]
16. Lindstrom, P. Error distributions of lossy floating-point compressors. In Proceedings of the Joint Statistical Meetings, Baltimore, MD, USA, 29 July–3 August 2017; Volume 2017, pp. 2574–2589.
17. Diffenderfer, J.; Fox, A.; Hittinger, J.; Sanders, G.; Lindstrom, P. Error Analysis of ZFP Compression for Floating-Point Data. *SIAM J. Sci. Comput.* **2019**, *41*, A1867–A1898. [CrossRef]
18. Di, S.; Cappello, F. Fast error-bounded lossy HPC data compression with SZ. In Proceedings of the 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS), Chicago, IL, USA, 23–27 May 2016; pp. 730–739.
19. Tao, D.; Di, S.; Chen, Z.; Cappello, F. 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), Orlando, FL, USA, 29 May–2 June 2017; pp. 1129–1139. [CrossRef]

20. Liang, X.; Di, S.; Tao, D.; Li, S.; Li, S.; Guo, H.; Chen, Z.; Cappello, F. Error-controlled lossy compression optimized for high compression ratios of scientific datasets. In Proceedings of the 2018 IEEE International Conference on Big Data (Big Data), Seattle, WA, USA, 10–13 December 2018; pp. 438–447.
21. Weiser, M.; Götschel, S. State Trajectory Compression for Optimal Control with Parabolic PDEs. *SIAM J. Sci. Comp.* **2012**, *34*, A161–A184. [CrossRef]
22. Götschel, S. Adaptive Lossy Trajectory Compression for Optimal Control of Parabolic PDEs. Ph.D. Thesis, Department of Mathematics and Computer Science, Freie Universität Berlin, Berlin, Germany, 2015.
23. Demaret, L.; Dyn, N.; Floater, M.; Iske, A. Adaptive Thinning for Terrain Modelling and Image Compression. In *Advances in Multiresolution for Geometric Modelling*; Dodgson, N., Floater, M., Sabin, M., Eds.; Springer: Berlin, Germany, 2005; pp. 319–338.
24. Solin, P.; Andrs, D. On Scientific Data and Image Compression Based on Adaptive Higher-Order FEM. *Adv. Appl. Math. Mech.* **2009**, *1*, 56–68.
25. Shafaat, T.; Baden, S. A method of adaptive coarsening for compressing scientific datasets. In *Applied Parallel Computing. State of the Art in Scientific Computing*; Kåström, B., Elmroth, E., Dongarra, J., Waśniewski, J., Eds.; Springer: Berlin, Germany, 2007; pp. 774–780.
26. Unat, D.; Hromadka, T.; Baden, S. An Adaptive Sub-sampling Method for In-memory Compression of Scientific Data. In Proceedings of the IEEE 2009 Data Compression Conference, Snowbird, UT, USA, 16–18 March 2009; p. 10666336.
27. Austin, W.; Ballard, G.; Kolda, T. Parallel Tensor Compression for Large-Scale Scientific Data. In Proceedings of the IEEE International Parallel and Distributed Processing Symposium, Chicago, IL, USA, 23–27 May 2016; p. 16158560.
28. Ballard, G.; Klinvex, A.; Kolda, T. TuckerMPI: A Parallel C++/MPI Software Package for Large-scale Data Compression via the Tucker Tensor Decomposition. *arXiv* **2019**, arXiv:1901.06043.
29. Ballester-Ripoll, R.; Lindstrom, P.; Pajarola, R. TTHRESH: Tensor Compression for Multidimensional Visual Data. *IEEE Trans. Vis. Comp. Graph.* **2019**. [CrossRef]
30. Ainsworth, M.; Tugluk, O.; Whitney, B.; Klasky, S. Multilevel techniques for compression and reduction of scientific data – the multilevel case. *SIAM J. Sci. Comput.* **2019**, *41*, A1278–A1303. [CrossRef]
31. Peyrot, J.L.; Duval, L.; Payan, F.; Bouard, L.; Chizat, L.; Schneider, S.; Antonini, M. HexaShrink, an exact scalable framework for hexahedral meshes with attributes and discontinuities: Multiresolution rendering and storage of geoscience models. *Comput. Geosci.* **2019**, *23*, 723–743. [CrossRef]
32. Tao, D.; Di, S.; Liang, X.; Chen, Z.; Cappello, F. Optimizing lossy compression rate-distortion from automatic online selection between sz and zfp. *IEEE Trans. Parallel Distrib. Syst.* **2019**, *30*, 1857–1871. [CrossRef]
33. Maglo, A.; Lavoué, G.; Dupont, F.; Hudelot, C. 3D Mesh Compression: Survey, Comparisons, and Emerging Trends. *ACM Comput. Surv.* **2015**, *47*, 44. [CrossRef]
34. Götschel, S.; von Tycowicz, C.; Polthier, K.; Weiser, M. Reducing Memory Requirements in Scientific Computing and Optimal Control. In *Multiple Shooting and Time Domain Decomposition Methods*; Carraro, T., Geiger, M., Körkel, S., Rannacher, R., Eds.; Springer: Berlin, Germany, 2015; pp. 263–287.
35. Nasiri, F.; Bidgoli, N.M.; Payan, F.; Maugey, T. A Geometry-aware Framework for Compressing 3D Mesh Textures. In Proceedings of the ICASSP 2019—2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Brighton, UK, 12–17 May 2019; pp. 4015–4019. [CrossRef]
36. Caillaud, F.; Vidal, V.; Dupont, F.; Lavoué, G. Progressive compression of arbitrary textured meshes. *Comput. Graphics Forum* **2016**, *35*, 475–484. [CrossRef]
37. Anzt, H.; Dongarra, J.; Flegar, G.; Higham, N.; Quintana-Ortí, E. Adaptive precision in block-Jacobi preconditioning for iterative sparse linear system solvers. *Concurr. Comput.* **2019**, *31*, e4460. [CrossRef]
38. Schneck, J.; Weiser, M.; Wende, F. *Impact of Mixed Precision and Storage Layout on Additive Schwarz Smoothers*; Report 18-62; Zuse Institute: Berlin, Germany, 2018.
39. Hackbusch, W. A sparse matrix arithmetic based on $\mathcal{H}$-matrices, Part I: introduction to $\mathcal{H}$-matrices. *Computing* **1999**, *62*, 89–108. [CrossRef]
40. Dahmen, W.; Harbrecht, H.; Schneider, R. Compression techniques for boundary integral equations – asymptotically optimal complexity estimates. *SIAM J. Numer. Anal.* **2006**, *43*, 2251–2271. [CrossRef]

41. Lu, T.; Liu, Q.; He, X.; Luo, H.; Suchyta, E.; Choi, J.; Podhorszki, N.; Klasky, S.; Wolf, M.; Liu, T.; et al. Understanding and Modeling Lossy Compression Schemes on HPC Scientific Data. In Proceedings of the 32nd IEEE International Parallel and Distributed Processing Symposium, Vancouver, BC, USA, 21–25 May 2018; p. 17974767.
42. Poppick, A.; Nardi, J.; Feldman, N.; Baker, A.; Hammerling, D. A Statistical Analysis of Compressed Climate Model Data. In Proceedings of the 4th International Workshop Data Reduction for Big Scientific Data, Frankfurt, Germany, 28 June 2018.
43. Baker, A.H.; Hammerling, D.M.; Mickelson, S.A.; Xu, H.; Stolpe, M.B.; Naveau, P.; Sanderson, B.; Ebert-Uphoff, I.; Samarasinghe, S.; De Simone, F.; et al. Evaluating lossy data compression on climate simulation data within a large ensemble. *Geosci. Model Dev.* **2016**, *9*, 4381–4403. [CrossRef]
44. Hoang, D.; Klacansky, P.; Bhatia, H.; Bremer, P.T.; Lindstrom, P.; Pascucci, V. A Study of the Trade-off Between Reducing Precision and Reducing Resolution for Data Analysis and Visualization. *IEEE Trans. Vis. Comp. Graph.* **2019**, *25*, 1193–1203. [CrossRef]
45. Whitney, B. Multilevel Techniques for Compression and Reduction of Scientific Data. Ph.D. Thesis, Brown University, Providence, RI, USA, 2018.
46. Götschel, S.; Weiser, M. Lossy Compression for PDE-constrained Optimization: Adaptive Error Control. *Comput. Optim. Appl.* **2015**, *62*, 131–155. [CrossRef]
47. Jacob, B.; Ng, S.; Wang, D. *Memory Systems: Cache, DRAM, Disk*; Morgan Kaufman: Burlington, MA, USA, 2010.
48. Cappello, F.; Di, S.; Li, S.; Liang, X.; Gok, A.; Tao, D.; Yoon, C.; Wu, X.C.; Alexeev, Y.; Chong, F. Use cases of lossy compression for floating-point data in scientific data sets. *Int. J. High Perf. Comp. Appl.* **2019**. [CrossRef]
49. Williams, S.; Waterman, A.; Patterson, D. Roofline: An insightful visual performance model for multicore architectures. *Comm. ACM* **2009**, *52*, 65–76. [CrossRef]
50. Pekhimnko, G.; Seshadri, V.; Kim, Y.; Xin, H.; Mutlu, O.; Gibbons, P.; Kozuch, M.; Mowry, T. Linearly compressed pages: A low-complexity, low-latency main memory compression framework. In Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture, Davis, CA, USA, 7–11 December 2013; p. 16657326.
51. Shafiee, A.; Taassori, M.; Balasubramonian, R.; Davis, A. MemZip: Exploring unconventional benefits from memory compression. In Proceedings of the 20th International Symposium on High Performance Computer Architecture, Orlando, FL, USA, 15–19 February 2014; p. 14393980.
52. Young, V.; Nair, P.; Qureshi, M. DICE: Compressing DRAM caches for bandwidth and capacity. In Proceedings of the 44th Annual International Symposium on Computer Architecture, Toronto, ON, Canada, 24–28 June 2017; p. 17430274.
53. Jain, A.; Hill, P.; Lin, S.C.; Khan, M.; Haque, M.; Laurenzano, M.; Mahlke, S.; Tang, L.; Mars, J. Concise loads and stores: The case for an asymmetric compute-memory architecture for approximation. In Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture, Taipei, Taiwan, 15–19 October 2016; p. 16545476.
54. Mittal, S.; Vetter, J. A Survey Of Architectural Approaches for Data Compression in Cache and Main Memory Systems. *IEEE Trans. Parallel Distrib. Syst.* **2016**, *27*, 1524–1536. [CrossRef]
55. Kahan, W. *754-2008—IEEE Standard for Floating-Point Arithmetic*; IEEE: Los Alamitos, CA, USA, 2008. [CrossRef]
56. Baboulin, M.; Buttari, A.; Dongarra, J.; Kurzak, J.; Langou, J.; Langou, J.; Luszczek, P.; Tomov, S. Accelerating scientific computations with mixed precision algorithms. *Comput. Phys. Commun.* **2009**, *180*, 2526–2533. [CrossRef]
57. Anzt, H.; Luszczek, P.; Dongarra, J.; Heuveline, V. GPU-Accelerated Asynchronous Error Correction for Mixed Precision Iterative Refinement. In *Euro-Par 2012 Parallel Processing*; Lecture Notes in Computer Science; Kaklamanis, C., Papatheodorou, T., Spirakis, P., Eds.; Springer: Berlin, Germany, 2012, Volume 7484.
58. Grout, R. *Mixed-Precision Spectral Deferred Correction*; Preprint CP-2C00-64959; National Renewable Energy Laboratory: Golden, CO, USA, 2015.
59. Giraud, L.; Haidar, A.; Watson, L. Mixed-Precision Preconditioners in Parallel Domain Decomposition Solvers. In *Domain Decomposition Methods in Science and Engineering XVII*; Lecture Notes in Computational Science and Engineering; Langer, U., Discacciati, M., Keyes, D., Widlund, O., Zulehner, W., Eds.; Springer: Berlin, Germany, 2008, Volume 60; pp. 357–364.

60. Ahmed, N.; Natarajan, T.; Rao, K.R. Discrete Cosine Transform. *IEEE Trans. Comput.* **1974**, *C-23*, 90–93. [CrossRef]
61. Said, A.; Pearlman, W. A new, fast, and efficient image codiec based on set partitioning in hierarchical trees. *IEEE Trans. Circ. Syst. Video Technol.* **1996**, *6*, 243–250. [CrossRef]
62. Toselli, A.; Widlund, O. *Domain Decomposition Methods—Algorithms and Theory*; Computational Mathematics; Springer: Berlin, Germany, 2005, Volume 34.
63. Gander, M. 50 Years of Time Parallel Time Integration. In *Multiple Shooting and Time Domain Decomposition Methods*; Carraro, T., Geiger, M., Körkel, S., Rannacher, R., Eds.; Springer: Cham, Switzerland, 2015; Volume 9, pp. 69–113.
64. Emmett, M.; Minion, M. Toward an efficient parallel in time method for partial differential equations. *Comm. Appl. Math. Comp. Sci.* **2012**, *7*, 105–132. [CrossRef]
65. Fischer, L.; Götschel, S.; Weiser, M. Lossy data compression reduces communication time in hybrid time-parallel integrators. *Comput. Vis. Sci.* **2018**, *19*, 19–30. [CrossRef]
66. Martin, G. Range encoding: An algorithm for removing redundancy from a digitised message. In Proceedings of the Video & Data Recording Conference, Southampton, Hampshire, UK, 24–27 July 1979.
67. Sweldens, W. The Lifting Scheme: A Construction of Second Generation Wavelets. *SIAM J. Math. Anal.* **1998**, *29*, 511–546. [CrossRef]
68. Stevenson, R. Locally supported, piecewise polynomial biorthogonal wavelets on nonuniform meshes. *Constr. Approx.* **2003**, *19*, 477–508. [CrossRef]
69. Cohen, A.; Echeverry, L.M.; Sun, Q. *Finite Element Wavelets*; Technical Report; Université Pierre et Marie Curi: Paris, France, 2000.
70. Ochoa, I.; Asnani, H.; Bharadia, D.; Chowdhury, M.; Weissman, T.; Yona, G. QualComp: A new lossy compressor for quality scores based on rate distortion theory. *BMC Bioinform.* **2013**, *14*, 187. [CrossRef]
71. Götschel, S.; Chamakuri, N.; Kunisch, K.; Weiser, M. Lossy Compression in Optimal Control of Cardiac Defibrillation. *J. Sci. Comp.* **2014**, *60*, 35–59. [CrossRef]
72. Böhm, C.; Hanzich, M.; de la Puente, J.; Fichtner, A. Wavefield compression for adjoint methods in full-waveform inversion. *Geophysics* **2016**, *81*, R385–R397. [CrossRef]
73. Lindstrom, P.; Chen, P.; Lee, E.J. Reducing disk storage of full-3D seismic waveform tomography (F3DT) through lossy online compression. *Comput. Geosci.* **2016**, *93*, 45–54. [CrossRef]
74. Oden, J.T.; Prudhomme, S. Goal-oriented error estimation and adaptivity for the finite element method. *Comput. Math. Appl.* **2001**, *41*, 735–756. [CrossRef]
75. Volin, Y.M.; Ostrovskii, G.M. Automatic computation of derivatives with the use of the multilevel differentiating techniques—1. Algorithmic basis. *Comput. Math. Appl.* **1985**, *11*, 1099–1114. [CrossRef]
76. Griewank, A. Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation. *Optim. Methods Softw.* **1992**, *1*, 35–54. [CrossRef]
77. Griewank, A.; Walther, A. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*; SIAM: Philadelphia, PA, USA, 2008.
78. Colli Franzone, P.; Deuflhard, P.; Erdmann, B.; Lang, J.; Pavarino, L. Adaptivity in Space and Time for Reaction-Diffusion Systems in Electrocardiology. *SIAM J. Sci. Comput.* **2006**, *28*, 942–962. [CrossRef]
79. Nagaiah, C.; Kunisch, K.; Plank, G. Numerical solution for optimal control of the reaction-diffusion equations in cardiac electrophysiology. *Comput. Optim. Appl.* **2011**, *49*, 149–178. [CrossRef]
80. Dennis, J.E., Jr.; Moré, J.J. Quasi-Newton methods, motivation and theory. *SIAM Rev.* **1977**, *19*, 46–89. [CrossRef]
81. Borzí, A.; Schulz, V. Computational Optimization of Systems Governed by Partial Differential Equations. In *Computational Science and Engineering*; SIAM: Philadelphia, PA, USA, 2012.
82. Deuflhard, P.; Leinen, P.; Yserentant, H. Concepts of an Adaptive Hierarchical Finite Element Code. *Impact Comput. Sci. Engrgy* **1989**, *1*, 3–35. [CrossRef]
83. von Tycowicz, C.; Kälberer, F.; Polthier, K. Context-Based Coding of Adaptive Multiresolution Meshes. *Comput. Graphics Forum* **2011**, *30*, 2231–2245. [CrossRef]
84. Becker, R.; Rannacher, R. A feed-back approach to error control in finite element methods: Basic analysis and examples. *East West J. Numer. Math.* **1996**, *4*, 237–264.
85. Becker, R.; Kapp, H.; Rannacher, R. Adaptive finite element methods for optimal control of partial differential equations: Basic concepts. *SIAM J. Control Optim.* **2000**, *39*, 113–132. [CrossRef]

86. Rannacher, R. On the adaptive discretization of PDE-based optimization problems. In *PDE Constrained Optimization*; Heinkenschloss, M., Ed.; Springer: Berlin, Germany, 2006.
87. Weiser, M. On goal-oriented adaptivity for elliptic optimal control problems. *Optim. Meth. Softw.* **2013**, *28*, 969–992. [CrossRef]
88. Cyr, E.; Shadid, J.; Wildey, T. Towards efficient backward-in-time adjoint computations using data compression techniques. *Comput. Methods Appl. Mech. Eng.* **2015**, *288*, 24–44. [CrossRef]
89. Tao, D.; Di, S.; Liang, X.; Chen, Z.; Cappello, F. Improving Performance of Iterative Methods by Lossy Checkponting. In Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing, Tempe, AZ, USA, 11–15 June 2018; ACM: New York, NY, USA, 2018; pp. 52–65. [CrossRef]
90. Calhoun, J.; Cappello, F.; Olson, L.; Snir, M.; Gropp, W. Exploring the feasibility of lossy compression for PDE simulations. *Int. J. High Perform. Comput. Appl.* **2019**, *33*, 397–410. [CrossRef]
91. Young, J.W. A First Order Approximation to the Optimum Checkpoint Interval. *Commun. ACM* **1974**, *17*, 530–531. [CrossRef]
92. Daly, J.T. A higher order estimate of the optimum checkpoint interval for restart dumps. *Future Gerner. Comp. Syst.* **2006**, *22*, 303–312. [CrossRef]
93. Di, S.; Robert, Y.; Vivien, F.; Cappello, F. Toward an Optimal Online Checkpoint Solution under a Two-Level HPC Checkpoint Model. *IEEE Trans. Parallel Distrib. Syst.* **2017**, *28*, 244–259. [CrossRef]
94. Thole, C.A. Compression of LS-DYNA3D$^{TM}$ Simulation Results using FEMZIP©. In Proceedings of the 3rd LS-DYNA Anwenderforum, Bamberg, Germany, 14–15 October 2004; pp. E-III-1–E-III-5.
95. Teran, R.I.; Thole, C.A.; Lorentz, R. New Developments in the Compression of LS-DYNA Simulation Results using FEMZIP. In Proceedings of the 6th European LS-DYNA Users' Conference, Salzburg, Austria, 9–11 May 2007.
96. Mertler, S.; Müller, S.; Thole, C. Predictive Principal Component Analysis as a Data Compression Core in a Simulation Data Management System. In Proceedings of the 2015 Data Compression Conference, Snowbird, UT, USA, 7–9 April 2015, pp. 173–182. [CrossRef]
97. Düben, P.D.; Leutbecher, M.; Bauer, P. New methods for data storage of model output from ensemble simulations. *Mon. Weather Rev.* **2019**, *147*, 677–689. [CrossRef]
98. Kuhn, M.; Kunkel, J.M.; Ludwig, T. Data compression for climate data. *Supercomput. Front. Innov.* **2016**, *3*, 75–94.
99. Otero, E.; Vinuesa, R.; Marin, O.; Laure, E.; Schlatter, P. Lossy data compression effects on wall-bounded turbulence: Bounds on data reduction. *Flow Turbul. Combust.* **2018**, *101*, 365–387. [CrossRef]
100. Marina, O.; Schanena, M.; Fischer, P. *Large-Scale Lossy Data Compression Based on an a Priori Error Estimator in a Spectral Element Code*; Technical Report; ANL/MCS-p6024-0616; Argonne National Laboratory: Lemont, IL, USA, 2016.