



SEBASTIAN GÖTSCHEL¹, MARTIN WEISER²

Lossy Compression for Large Scale PDE Problems

¹  0000-0003-0287-2120
²  0000-0002-1071-0044

Zuse Institute Berlin
Takustr. 7
14195 Berlin
Germany

Telephone: +49 30-84185-0
Telefax: +49 30-84185-125

E-mail: bibliothek@zib.de
URL: <http://www.zib.de>

ZIB-Report (Print) ISSN 1438-0064
ZIB-Report (Internet) ISSN 2192-7782

Lossy Compression for Large Scale PDE Problems

Sebastian Götschel¹, Martin Weiser¹

July 1, 2019

Abstract

Solvers for partial differential equations (PDE) are one of the cornerstones of computational science. For large problems, they involve huge amounts of data that needs to be stored and transmitted on all levels of the memory hierarchy. Often, bandwidth is the limiting factor due to relatively small arithmetic intensity, and increasingly so 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 during the last decades.

This paper surveys data compression challenges and corresponding solution approaches for PDE problems, covering all levels of the memory hierarchy from mass storage up to main memory. Exemplarily, we illustrate concepts at particular methods, and give references to alternatives.

Keywords: partial differential equation, data compression, floating point compression, lossy compression

MSC 2010: 65-02, 65M60, 65N30, 68-02, 68P30

1 Introduction

Partial differential equations (PDEs) describe many phenomena mostly in the natural sciences. Due to their 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 corner stones of modern science and economy. The analytic solution of PDEs in form of explicit expressions or series representations is, however, only possible for the most simplistic cases. Numerical simulation using finite element and finite volume methods [17, 66] approximates the solutions on spatio-temporal meshes, and is 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 amounts of spatial degrees of freedom (up to 10^9) and many time steps. Thus, numerical simulation algorithms produce large amounts

¹Zuse Institute Berlin, Takustr. 7, 14195 Berlin, {goetschel,weiser}@zib.de

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. Today, the performance of PDE solvers is mostly limited by communication bandwidth, with the CPU kernels achieving only a tiny fraction of their peak performance. This concerns the CPU-memory communication, the so-called “memory wall” [42, 43], as well as inter-node communication in large distributed systems [49], 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. If needed for storing the results, it 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.

1.1 Compression aspects of PDE applications

PDE solvers have a requirements 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, mostly coefficient vectors representing, e.g., finite element functions, consists of floating point numbers. Usually, double precision is used 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, though, the less significant mantissa bits are essentially random, and incur a large entropy. Lossless compression methods are, therefore, not able to achieve substantial compression rates. Lossy compression allows much higher compression rates, but requires a careful selection of quantization error in order not to compromise the final result of the computation. Fortunately, quantitative error estimates are often available for rate-distortion optimization.

Compression plays different roles on all levels of the memory hierarchy, depending on application, problem size, and computer architecture. Compression of in-memory data aims at avoiding the “memory wall” and reducing the run time of the simulation (see Sec. 2). The available bandwidth is quite high, even if not sufficient for saturating the compute 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 Sec. 3). 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 investigation. Here, data size reduction is of primal interest, and not the run time reduction. Complex compression algorithms exploiting correlations in large data sets can be employed (see Sec. 4).

General purpose floating point compression General purpose lossy compression methods for floating point data are usable as black-box compressors. However, as they cannot make use of the structure of data from specific applications, like finite element solutions on unstructured grids, they are typically outperformed by specialized algorithms. For this reason, instead of giving a more detailed overview, we briefly discuss two examples. For more details we refer to the survey [28]. The *fpzip* algorithm based on [39] traverses floating point data arrays in some specified order and predicts values based on a subset of already encoded data. The least significant bits are truncated, reducing the precision. Residuals are encoding by range coding [41]. More recently, *zfp*¹ [19, 37] was introduced, improving accuracy and throughput compared to its predecessor *fpzip*. As *fpzip* and *zfp* assume the data to be on a regular n -dimensional grid ($n = 1, \dots, 4$), exhibiting spatial correlation, they are not entirely general purpose compressors. Other recent developments include [56], who use a series of prediction formulas with adaptive error control, and evaluate their methods on a various scientific data sets.

2 The memory wall

One of the most important properties of an algorithm determining its actual performance is its arithmetic intensity, i.e., the number of floating point operations performed per byte that is read from or written to memory. With respect to that quantity, the performance can be described by the roofline model [65]. It includes two main bounds, the peak performance, and the peak memory bandwidth, see Fig. 1. Most finite element solvers, in particular those working on unstructured grids, make heavy use of sparse linear algebra, and are usually memory bound.

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 reduces the budget available for payload flops. This is illustrated in Fig. 1. Sparse matrix-vector products are usually memory bound with an arithmetic intensity of less than 0.25 flops/byte. Data compression with different compression schemes by a factor 8 (diamonds) or 16 (circles) increases the arithmetic intensity and moves the computation to the peak floating point roofline. The (de)compression overhead of one (diamonds) or three flops (circles), respectively,

¹ <https://computation.llnl.gov/projects/floating-point-compression>

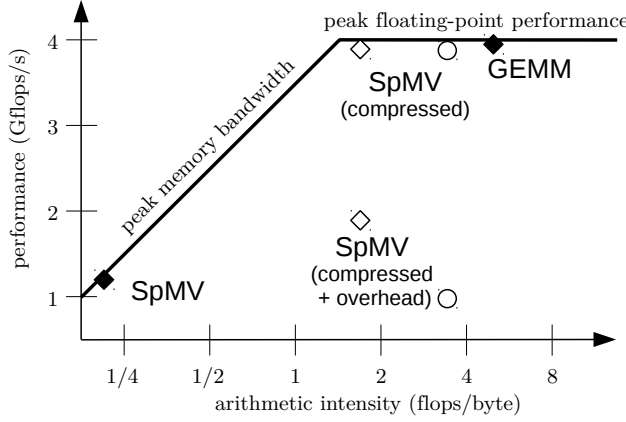


Figure 1: Naive roofline model showing achievable performance vs. the arithmetic intensity. Some computations, e.g., dense matrix-matrix multiplication (GEMM), perform many flops per byte fetched or written to memory, such that their execution speed is bounded by the peak floating point performance. 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 can reduce the amount of data to be read or written, and therefore increase the arithmetic intensity (empty markers on top). The computational overhead of compression and decompression can, however, reduce the performance gain (empty markers bottom).

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 for overcoming the memory wall, only very fast and therefore rather simple compression schemes are beneficial.

2.1 Mixed-precision arithmetics

One particularly simple way 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 [1], 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 [5] and iterative solvers [4, 34]. Depending on the algorithm’s position in the roofline model, either the reduced memory traffic (BLAS level 1/2) or the faster execution of lower precision floating point operations (BLAS level 3) 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 [17]. For higher order finite elements, with polynomial ansatz order $p > 2$, the Jacobi smoother quickly deteriorates. Then it needs to be replaced by an overlapping block Jacobi smoother B , with the blocks consisting of all degrees of freedom associated to cells around a grid vertex. Application of this smoother then involves a large number of essentially dense matrix-vector multiplications of moderate size:

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

Here, \mathcal{N} is the set of grid vertices, A_{ξ} is the symmetric submatrix of A corresponding to the vertex ξ , and P_{ξ} 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_{ξ}^{-1} as \tilde{A}_{ξ}^{-1} by using low precision representation of its entries has been investigated in [52]. A detailed analysis reveals that the impact on the preconditioner effectivity and hence the CG convergence is determined by $\|A^{-1} - \tilde{A}^{-1}\|$. 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 alternative to low precision floating point representations. Moreover, the matrix entries exhibit a certain degree of correlation, which can be exploited by dividing A_{ξ}^{-1} into quadratic blocks to be stored independently, and quantizing the difference of entries c with respect to the block entries' range $[c_{\min}, c_{\max}]$ as

$$c \mapsto \begin{cases} \lfloor 2^k \frac{c - c_{\min}}{c_{\max} - c_{\min}} \rfloor, & c < c_{\max} \\ 2^k - 1, & c = c_{\max}. \end{cases}$$

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 is performed in hardware, which restricts the possible compression factors to $\{2, 4, 8\}$, for which the speedup reaches almost the compression factor, see Fig. 2.

The theoretical error estimates together with typical condition numbers of local matrices A_{ξ} 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 4. With moderate ansatz order $p \leq 6$, even 8 bit fixed point representations can be used, achieving a speedup of up to 6.

Similar results have been obtained for non-overlapping block-Jacobi preconditioners for Krylov methods applied to general sparse systems [3] and for substructuring domain decomposition methods [25].

We'd 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 and hence the data size the bottleneck, but there is also a

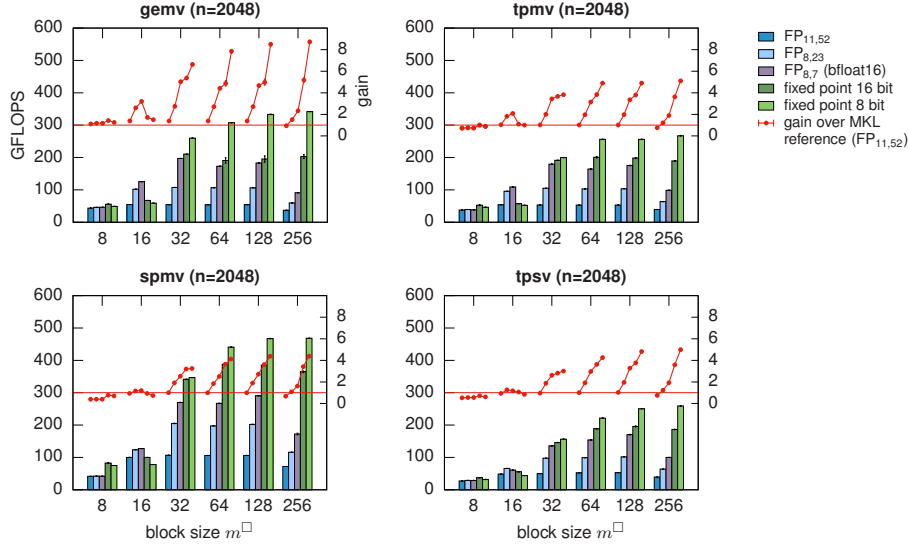


Figure 2: Run times of BLAS level 2 operations on 2048×2048 -matrices for overlapping Schwarz smoothers with mixed precision. Depending on the access patterns, a speedup almost on par with compression factor can be achieved [52].

compelling mathematical reason for performing the actual computations in high precision arithmetics: Storing the inverted submatrices A_{ξ}^{-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, and therefore leave the well-understood theory of subspace correction methods.

2.2 Fixed-rate transform coding

For higher compression rates than achievable with reduced precision storage, more sophisticated and computationally more expensive approaches are required. Competing aims are high compression rate, low computational overhead, as well as transparent and random access. One particularly advanced approach is transform coding on cartesian grids [36]. 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 to consider 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, necessary for random access, and compression rate 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 [2] can be used, a special transform with slightly higher decorrelation efficiency has been developed in [36]. Such orthogonal transforms can be applied efficiently by exploiting separabil-

ity and lifting scheme for factorization, i.e., applying 1D transforms in each dimensions, and realizing these 1D transforms by a sequence of cheap in-place modifications. This results in roughly 11 flops per coefficient. The transformed coefficients are then coded bitplane by bitplane using group testing, similar to set partitioning in hierarchical trees [50]. This embedded coding schemes allows to decode 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 rate, is reported to lie around 400 MB/s, which is around a factor of ten below contemporary memory bandwidth.

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 future: As the hardware trend to more cores per CPU socket continues, and thus the gap between computing performance and memory bandwidth widens, higher complexity of in-memory compression will pay off.

3 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 locality of interaction in PDEs, communication happens at the boundary shared by adjacent subdomains. A prime example are domain decomposition solvers for elliptic problems [60].

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. As the volume of subdomains with diameter h and hence the computational work scales with h^d in \mathbb{R}^d but their surface and hence communication only with h^{d-1} , 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 [11] 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 rates 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 rate, however, to values between

1.3 and 2.0, depending on the size of lookup tables. Filgueira et al. [22] 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 rates below 2 are achieved.

Higher compression rates can only be achieved with lossy compression. In contrast to in-memory compression, the inter-node communication bandwidth is small enough to allow for adaptive selection of quantization tolerances based on error estimators. Thus, error propagation analysis becomes important for compression.

3.1 Inexact parallel-in-time integrators

One example is the communication 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 [24]. Here, the initial value problem is interpreted as large equation system

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

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 :

$$U_{j+1} = U_j + S(F(U_j))$$

The advantage is, that a large part of the approximate solver S can be parallelized. If the application of S is significantly faster than computing a single subtrajectory up to discretization accuracy, reasonable parallel efficiencies above 0.5 can be achieved [21].

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 [23]. 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 rate and distortion must rely on error estimates and run time models.

The worst-case error analysis presented in [23] provides a bound of the type

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

depending on the relative communication accuracy Δ_C , the local contraction rate ρ 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 communication accuracy. The run time of the whole computation is

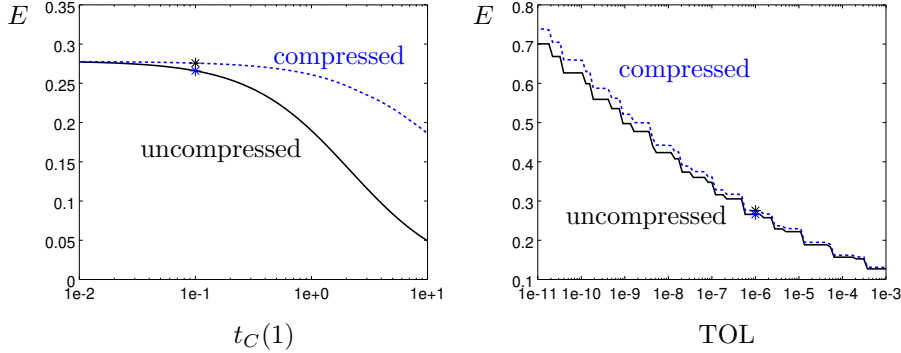


Figure 3: Theoretically estimated parallel efficiency $E = T_{\text{seq}}/(NT_{\text{par}})$ for variation of different parameters around the nominal scenario (marked by *). *Left*: varying communication bandwidth in terms of the communication time for uncompressed data. *Right*: varying requested tolerance.

$T_{\text{par}} = N(t_G + t_C(\Delta_C)) + J(\Delta_C)(t_G + t_F)$, where t_G is the time required for the sequential part of S , t_F for the parallel part, and t_C the communication time. Minimizing T can be used to optimize the compression scheme, as long as the relation between Δ_C and t_C is known. For finite element coefficients, most schemes will lead to $t_C \approx -c \log \Delta_C$, where the constant c depends crucially on the type of compression.

Variation of different parameters in this model around a nominal scenario of contemporary compute clusters as shown in Fig. 3 suggest that in many current HPC situations, the expected benefit for the run time is small. The pronounced dependence on smaller bandwidth, however, makes this approach interesting for 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.

Indeed, using the cheap transform coding discussed in Sec. 3.2 below, overall run time reduction by 10% has been observed on contemporary compute clusters.

3.2 Transform coding on unstructured grids

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 for some computations the method from [36] can be used, 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 $\mathcal{N} = \{x_i \mid i = 1, \dots, m\}$, see Fig. 10 for a 2D example. The simplest finite element discretization is then with piecewise linear functions, i.e., the solution is sought in the ansatz space $V_h = \{u \in C^0(\Omega) \mid \forall T \in \mathcal{T} : u|_T \in \mathbb{P}_1\}$.

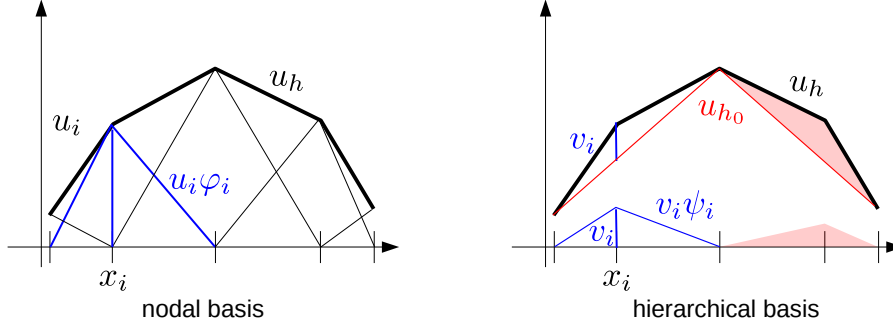


Figure 4: Representation of 1D linear finite element functions u_h in nodal and hierarchical basis.

The ubiquitous basis for V_h is the nodal basis $(\varphi_i)_{i=1,\dots,m}$ 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 for efficient multilevel solvers [17]. The restriction and prolongation operators implemented for those solvers realize a frequency decomposition of the solution

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

see Fig. 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 FE codes. The transformed coefficients can then be quantized according to the required accuracy and entropy coded, e.g., using a range coder [41]. Typically, this transform coding scheme takes much less than 5% of the iterative solution time.

A priori error estimates for compression rates and induced distortion can be derived for functions in Lebesgue or Sobolev spaces. The analysis in [64] 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^∞ -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 Fig. 5.

Error metrics. An important aspect of the transform coding design is the norm in which to measure compression errors. While in some applications point-wise error bounds are important and the L^∞ -norm is appropriate, other applications have different requirements. E.g., 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.

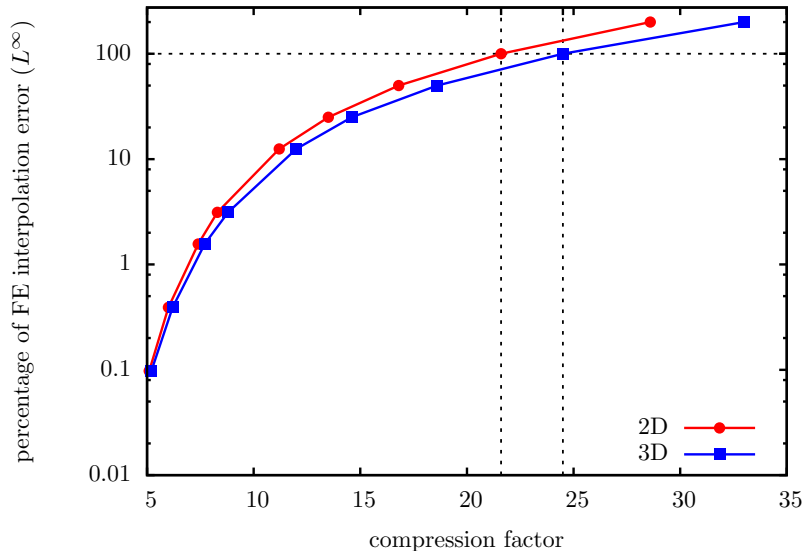


Figure 5: Error vs. compression factor: a-priori estimates for transform coding of finite element functions with hierarchical basis transform.

Nearly optimal compression rates for given H^{-1} -distortion can be achieved by replacing the hierarchical basis transform by a wavelet transform, which can efficiently be realized by lifting [55] on unstructured mesh hierarchies. Level-dependent quantization can be used for near-optimal compression rates matching a prescribed reconstruction error in H^s . Rigorous theoretical norm-equivalence results are available for $|s| < 3/2$ with a rather sophisticated construction [54]. A simpler finite element wavelet construction yields norm equivalences for $-0.114 < s < 3/2$ [13], but in numerical practice it works perfectly well also for $s = -1$.

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 [26].

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 to guarantee strict pointwise reconstruction error bounds directly, and is therefore closely linked to L^∞ 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-frequent levels to high-frequent levels, leading to less efficient decorrelation if error bounds in Sobolev spaces are important.

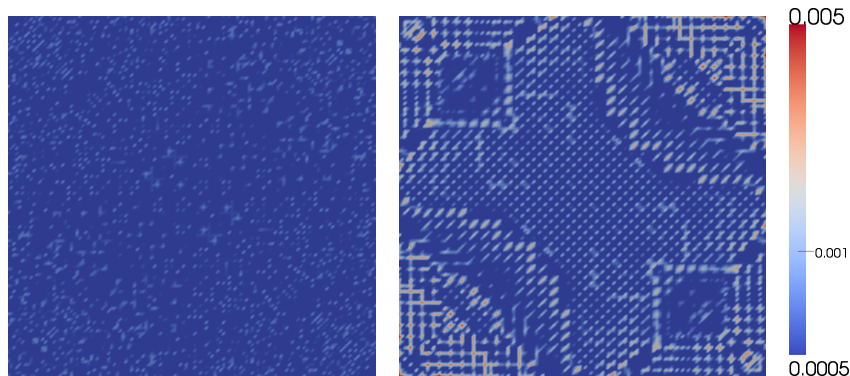


Figure 6: Comparison of quantization errors yielding the same H^{-1} error. Left: hierarchical basis. Right: wavelets.

Adaptive grid refinement. PDE solutions often exhibit spatially local features, e.g., corner singularities or moving fronts, 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 effort 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. [17].

Of course, a coarser discretization implies the storage of fewer coefficients, and is therefore, compared to uniform fine grids, also a method for lossy data compression. This has been explicitly exploited by Solin and Andrs [53] for image compression by adaptive mesh refinement.

Interestingly, even though adaptive mesh refinement and hierarchical transform coding both compete for the same spatial correlation of data, i.e., smoothness of functions to compress, their combination can achieve better compression rates for a given distortion than each of the approaches alone. A simple example is shown in Fig. 7 with compression rates given in Tab. 1. Using both, adaptive mesh refinement and transform coding, does not yield the product of individual compression factors, which tells 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 rate of adaptive mesh refinement as given in Tab. 1 is, however, somewhat too optimistic, since it only counts the number of coefficient 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 [29].

Another reason why adaptive mesh refinement and transform coding can be combined effectively for higher compression rates, is that the accuracy requirements for the mesh refinement and the solution storage can be very different. Thus, one approach may need to retain information that the other can safely neglect, allowing the latter to achieve additional compression on top of the former. E.g., in time-dependent problems, the mesh refinement affects the accuracy of all future time steps, whereas the solution storage for later postprocessing affects only one time step. A further example is adjoint gradient computation as

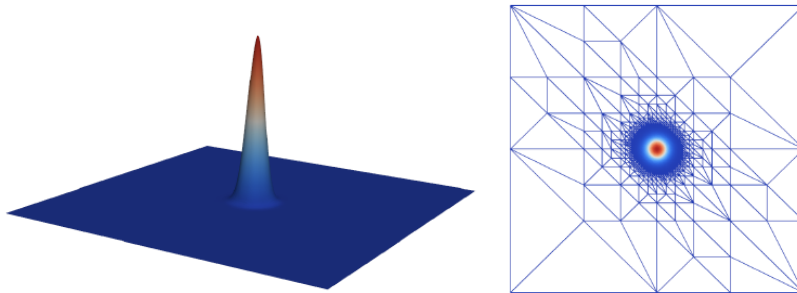


Figure 7: *Left*: highly local peak function. *Right*: adaptively refined mesh with 4,237 vertices for minimal finite element interpolation error. A uniform grid with the same local resolution has 263,169 vertices.

discussed in Sec. 4.1 below.

4 Mass storage

The third level in 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 to employ a posteriori error estimators for a better control of the quantization 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 (Sec. 4.1) and checkpoint/restart for fault tolerance (Sec. 4.2).

In the extreme case, the size of the data to store is the limiting factor, and to a large extent independent of the compression effort. This is typically the case in solution archiving (Sec. 4.3).

4.1 Adjoint solutions

Adjoint, or dual, equations are important in PDE-constrained optimization problems, e.g., optimal control in electrophysiology [27] or inverse problems [9, 38], and goal-oriented error estimation [46]. Consider the abstract variational problem

$$\text{find } x \in X \text{ such that } c(x; \varphi) = 0 \quad \forall \varphi \in \Phi, \quad (1)$$

	double	transform coding
uniform	1	54
adaptive	62	744

Table 1: Compression factors for adaptive mesh refinement and transform coding of the peak function shown in Fig. 7.

for a differentiable semilinear form $c : X \times \Phi \rightarrow Z$ with suitable function spaces X, Φ, Z . Let further be given a functional $J : X \rightarrow \mathbb{R}$. During the numerical solution, eq. (1) is typically only fulfilled up to a nonzero residual r , i.e., $c(x; \varphi) = r$. Naturally the question arises, how the residual r influences the evaluation of J . 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 to keep more data in RAM and potentially avoid disk access.

Lossy compression for computing adjoints can be done using transform coding as discussed in Sec. 3.2. In addition to the spatial smoothness, correlations in time can be exploited for compression. Since the stored values are only accessed backwards in time, delta encoding can be used. Here, only the quantized coefficients at the final time are fully stored, encoding the difference to the next time point at other times. This can be efficiently implemented, requiring only to keep one additional time step in memory. Instead of predicting values as constant, linear or even higher order prediction in time can be used as well. Even 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 [28, 64].

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 [61], and Griewank [32]. Instead of keeping track of the whole forward trajectory, only the solution at some intermediate timesteps is stored. During the integration of the adjoint equation, the required states are re-computed, starting from the snapshots, see, e.g., [33] 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 [28] for a more detailed discussion and additional references.

PDE-constrained optimization. For PDE-constrained optimization, typically $X = Y \times U$, $x = (y, u)$ in the abstract problem (1), where the influence of the control u on the state y is given by the PDE. Here, J is the cost functional 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 (1) 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 $p \in Z^*$

$$c_y^*(p; (y, u), \varphi) = -J_y((y, u), \varphi),$$

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 Ω . This system describes the electrical activity of the heart and consists of a parabolic PDE for the transmembrane voltage v , coupled to pointwise ODEs for the gating variable w ,

$$\begin{aligned} v_t &= \nabla \cdot \sigma \nabla v - I_{\text{ion}}(v, w) + I_e && \text{in } \Omega \times (0, T) \\ w_t &= G(v, w) && \text{in } \Omega \times (0, T), \end{aligned} \quad (2)$$

with

$$\begin{aligned} I_{\text{ion}}(v, w) &= gv \left(1 - \frac{v}{v_{th}}\right) \left(1 - \frac{v}{v_p}\right) + \eta_1 vw \\ G(v, w) &= \eta_2 \left(\frac{v}{v_p} - \eta_3 w\right) \end{aligned}$$

and homogeneous Neumann boundary conditions. In this 2D model $\sigma : \mathbb{R}^2 \rightarrow \mathbb{R}^{2 \times 2}$ and $g, \eta_i, v_p, v_{th} \in \mathbb{R}_+$ are given parameters (see, e.g., [14]). For the optimal control problem an initial excitation in some subdomain Ω_{exi} is prescribed. The external current stimulus $I_e(x, t) = \chi_{\Omega_c}(x)u(t)$, where the control u is spatially constant on a control domain Ω_c . Defining some observation domain Ω_{obs} , the objective functional is given by

$$J(y, u) = \frac{1}{2} \|v\|_{L^2(\Omega_{\text{obs}} \times (0, T))}^2 + \frac{\alpha}{2} \|u\|_{L^2(0, T)}^2, \quad (3)$$

i.e., we aim at damping out the excitation wave. We use $T = 4$, and $\alpha = 3 \cdot 10^{-6}$; for details, see [45]. Solution of the optimization problem with inexact Newton-CG methods and lossy compression is investigated in [27, 30]; here we use the BFGS-quasi-Newton method (e.g., [10, 15]). For time discretization we use a linearly implicit Euler method with fixed timestep size $dt = 0.04$. Spatial adaptivity is performed individually for state and adjoint using the hierarchical DLY error estimator [16], with a restriction to at most 25 000 vertices in space. The adaptively refined grids were stored using the methods from [62], which reduced the storage space for the mesh to less than 1 bit/vertex (see [28]).

Lossy compression of state values for adjoint gradient computation results in inexact quasi-Newton updates. Error analysis [26] shows that BFGS with inexact gradients converges linearly, if the gradient error e_g in each step fulfills

$$\|e_g\| \leq \frac{\varepsilon}{\kappa(B)^{1/2}} \|\tilde{g}\| \quad (4)$$

for $\varepsilon < \frac{1}{2}$. Here, $\kappa(B)$ is the condition number of the approximate Hessian B , and \tilde{g} denotes the inexactly computed gradient.

Figure 8 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 9. In the first iteration, a user-prescribed tolerance was used. The estimated condition number of the reduced Hessian varies between 200–230. We note that the adaptively chosen tolerances are too restrictive due to overestimation of the error in the worst case error estimates, and the fixed tolerance for the discretization.

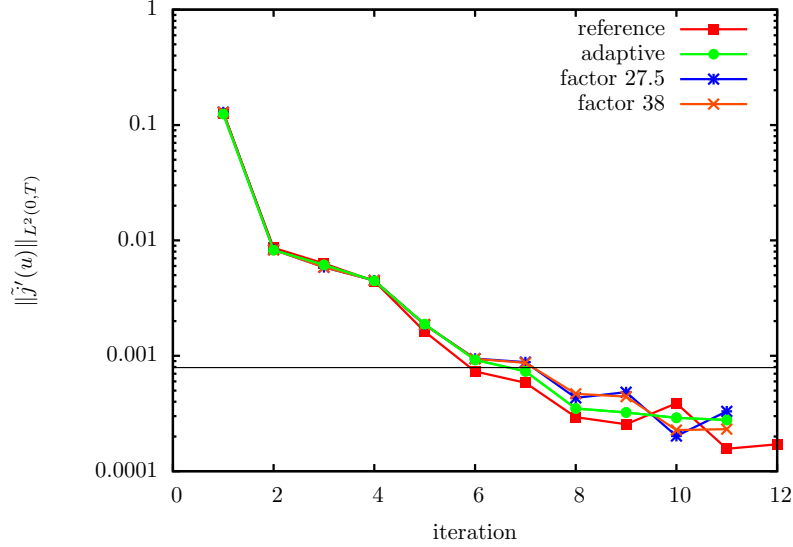


Figure 8: Optimization progress of BFGS for the monodomain example (2), (3), using different quantization tolerances for the state trajectory. No delta-encoding between timesteps was used. The horizontal line shows the approximate discretization error of the reduced gradient. See also [28].

Goal-oriented error estimation. For goal-oriented adaptivity, we consider solving the PDE (1) by a Galerkin approximation,

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

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 average of the solution, or in case of optimal control problems the objective functional, with the aim that

$$|J(x_h) - J(x)| \leq \epsilon.$$

The dual weighted residual (DWR) method [7, 8] now seeks to refine the approximation (typically the finite element mesh used to discretize the PDE in space) by weighting (local) residuals with information about their global influence on the goal functional J [48]. These weights are computed by the dual problem

$$\text{find } p \in Z^* \text{ such that } c_x^*(p; x, \varphi) = J_x(x, \varphi) \quad \forall \varphi \in \Phi^*.$$

The effect of compression on error estimation is illustrated in Fig. 10. For simplicity we use a linear-quadratic elliptic optimal control problem here (example 3b in [63]), with the objective functional 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 [63] 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. [7] for the all-at-once error $J(y_h, u_h) - J(\bar{y}, \bar{u})$. The compression tolerance was chosen such that the error estimation is barely influenced, resulting in only slightly different meshes.

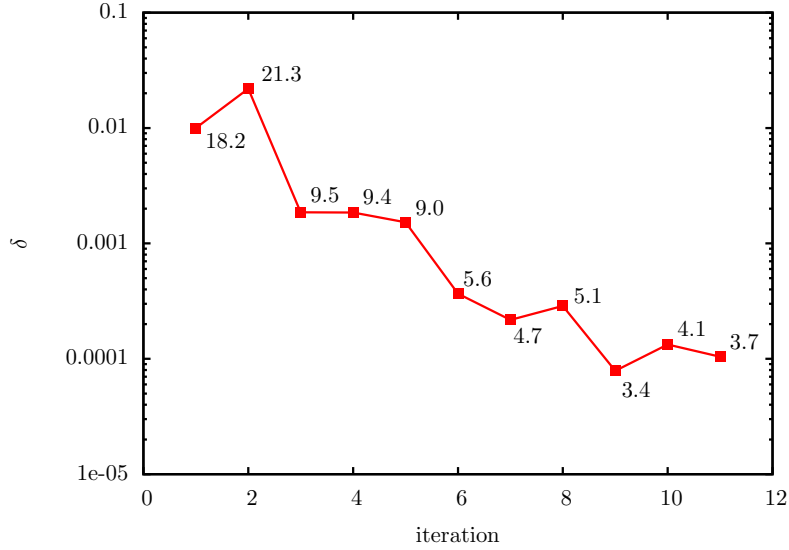


Figure 9: Adaptively chosen quantization tolerances δ and corresponding compression factors for the monodomain example (2), (3) using BFGS. The relatively small adaptive compression factors are due to using worst case error estimates in (4), as well as a fixed maximum mesh size.

For the final refinement step, i.e., on the finest mesh, compression resulted in data reduction by a factor of 32. 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.

4.2 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. [57]. 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 Jacobi, GMRES, and CG methods.

Calhoun et al. [12] 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,

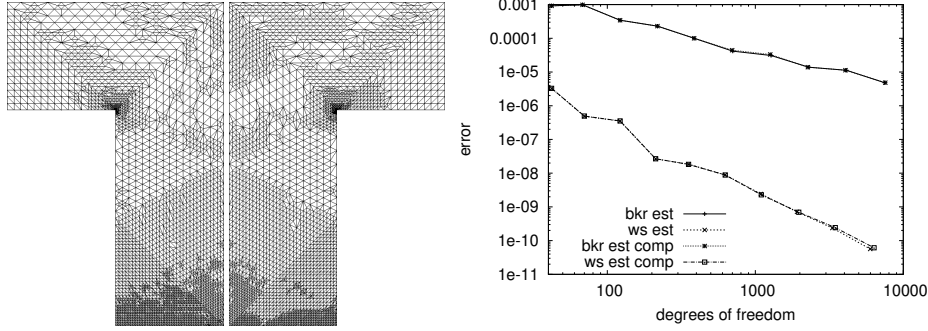


Figure 10: Goal-oriented error estimation: generated meshes with (left) and without (middle) compression (with compression factor up to 32) for example 3b in [63]. Meshes were generated using weights according to Weiser [63]. Estimated errors (right) are shown for weights due [63] (ws) and to Becker et al. [7] (bkr), both with and without compression.

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, 56]. 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.

For a parallel-in-time simulation application (see Sec. 3.1) and using the notation in Tab. 2, we derive a simple model relating probability of failure and checkpoint times to the overall runtime of the application, $T = T_C + nT_{CP} + T_{RS}N_{RS}$. Note that the time for actual computation T_C depends on the number of cores used. With time for restart $T_{RS} = \frac{1}{2} \frac{T}{n} + T_R$ consisting of the average required re-computation from the last written checkpoint to the time of failure and time to recover data structures, and $N_{RS} = p_{RS}TN$, the overall runtime is given as

$$T(n) = \frac{n(b - \sqrt{b^2 - \frac{2}{n}p_{RS}N(T_C + nT_{CP})})}{p_{RS}N},$$

n	number of checkpoints	T_C	time for actual computation
N	number of nodes	T_{CP}	time to write/read a checkpoint
p_{RS}	probability of failure per unit time and node	T_{DS}	time to recover data structures
N_{RS}	number of restarts	T_R	recovery time = $T_{CP} + T_{DS}$
T	overall runtime (wall clock)	T_{RS}	time for restart

Table 2: Notation used for optimal checkpointing.

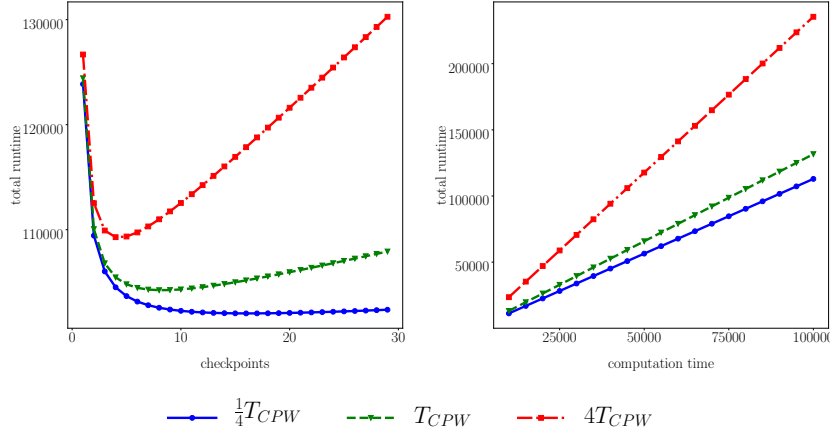


Figure 11: Influence of T_{CP} . *Left*: overall runtime vs. number of checkpoints for $N = 4$ and $T_C = 100000$ s. *Right*: overall runtime vs. actual computation time for $N = 100$. In both cases $p_{RS} = 7.74 \cdot 10^{-7}$, $T_{CP} = 245.583$ s and $T_R = 545.583$ s were used. The parameters were estimated for the parallel-in-time solution of a 3D heat equation on the HLRN-III Cray XC30/XC40 supercomputer (www.hlrn.de).

with $b := 1 - T_R p_{RS} N$. For the existence of a real solution the condition

$$b^2 \geq \frac{2}{n} p_{RS} N (T_C + n T_{CP}) = \frac{2}{n} p_{RS} N T_C + 2 p_{RS} N T_{CP}$$

has to be satisfied. Given parameters of the HPC system and the application, an optimal number of checkpoints can be determined by solving the optimization problem

$$\min_n T(n).$$

This can be done analytically, yielding

$$n_{\text{opt}} = \frac{T_C (2 p_{RS} N T_{CP} + b \sqrt{2 p_{RS} N T_{CP}})}{2 T_{CP} (b^2 - 2 N p_{RS} T_{CP})}.$$

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 [12], here we just exemplarily show the influence in Fig. 11. Reducing checkpoint size by lossy compression, thus reducing T_{CP} , T_{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 [57]; a thorough analysis for the example of parallel-in-time simulation with hybrid parareal methods can be done along the lines of [23].

4.3 Postprocessing and archiving

Storage and postprocessing of results from large-scale simulations require 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.

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 FEMzip² [59] achieves compression factors of 10–20 [44,58], obviously 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 rate, reporting an increase by a factor of 4.4 for a set of 14 simulation results [44].

Weather and climate. Today, prediction of weather and climate is one major use of supercomputing facilities, with tremendous amount of data to be stored³. Thus, using compression on the whole I/O system (main memory, communication, storage) can significantly improve performance [35]. Typically, ensemble simulations are used, allowing to exploit correlations. Three different approaches are investigated in [20]. 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 [6], 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* [39].

Computational fluid dynamics. Solution statistics of turbulent flow are used in [47] to assess error tolerances for lossy compression. Data reduction is performed by spatial transform coding with the Discrete Legendre Transform [40], which matches the spectral discretization on quadrilateral grids. For turbulence statistics of turbulent flow through a pipe, they report 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 [47].

For illustration, we consider laminar flow around a circular obstacle in a rectangular domain of 6×12 m at Reynolds number $Re = 100$, a test case along the lines of the unsteady 2D benchmark problem from [51]. The Navier-Stokes equations are numerically solved using the finite element toolbox Kaskade 7 [31].

²<https://www.sidact.com/femzip0.html>; FEMzip is a registered trademark of Fraunhofer Gesellschaft, Munich.

³In 2017, ECMWF's data archive grew by about 233 TB per day <https://www.ecmwf.int/en/computing/our-facilities/data-handling-system> [20].

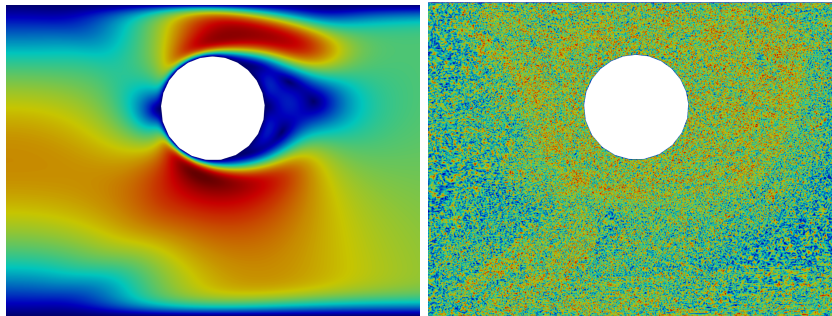


Figure 12: *Left*: Reconstructed velocity magnitude of the flow at $t = 2.5$ s, compression factor 11 (color coding: blue 0m/s to red 2m/s). *Right*: compression error (range 0 to 10^{-4} m/s). Only the left half of the computational domain is shown.

For space discretization, Taylor-Hood elements (quadratic for velocities u, v and linear for pressure p) are used on a triangular mesh consisting of nearly 120 000 cells. In time, a linearly implicit Euler method with stepsize 0.0025s is used. Inflow condition on the left boundary is a parabolic velocity profile, which is scaled linearly from zero until a maximum of $u = 1.5$ m/s, $v = 0$ is reached at $t = 1$ s. On the top and bottom boundary, no-slip conditions $u = v = 0$, on the right $p = 0$ are prescribed.

The data has been compressed using the hierarchical basis transform coding described in Sec. 3.2. Fig. 12 shows the reconstruction at $t = 2.5$ s from compressed storage with compression factor 11 (compression tolerance: absolute error 10^{-4} , leading to a relative error of $5 \cdot 10^{-5}$), as well as a plot of the compression error. Here, no difference between reconstructed and original solution is noticeable.

5 Conclusions

Data compression methods are valuable tools for accelerating PDE solvers, addressing larger problems, or archiving computed solutions. Due to floating-point data to be compressed, only lossy compression can be expected to achieve reasonable compression rates – which 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 errors on the ultimate use of the computed data, in order to be able to achieve high compression rates while meeting the accuracy requirements. This, in turn, calls for problem-specific approaches.

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 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, makes data compression methods more important over time. Thus, we can expect to see a growing need for data compression in PDE solvers in the next years.

Acknowledgements. This work has been partially supported by the German Ministry for Education and Research (BMBF) under project grant 01IH16005 (HighPerMeshes). We thank Florian Wende for implementing mixed-precision preconditioners, Alexander Kammeyer for implementation and testing of check-point/restart, and Thomas Steinke for many helpful discussions.

References

- [1] 754-2008 – IEEE standard for floating-point arithmetic. IEEE, 2008.
- [2] N. Ahmed, T. Natarajan, and K. R. Rao. Discrete cosine transform. *IEEE Transactions on Computers*, C-23(1):90–93, Jan 1974.
- [3] H. Anzt, J. Dongarra, G. Flegar, N. Higham, and E. Quintana-Ortí. Adaptive precision in block-Jacobi preconditioning for iterative sparse linear system solvers. *Concurrency and Computation: Practice and Experience*, 31(6):e4460, 2019.
- [4] H. Anzt, P. Luszczek, J. Dongarra, and V. Heuveline. GPU-accelerated asynchronous error correction for mixed precision iterative refinement. In C. Kaklamanis, T. Papatheodorou, and P. Spirakis, editors, *Euro-Par 2012 Parallel Processing*, volume 7484 of *Lecture Notes in Computer Science*. Springer.
- [5] M. Baboulin, A. Buttari, J. Dongarra, J. Kurzak, J. Langou, J. Langou, P. Luszczek, and S. Tomov. Accelerating scientific computations with mixed precision algorithms. *Compu. Phys. Comm.*, 180(12):2526–2533, 2009.
- [6] A. H. Baker, D. M. Hammerling, S. A. Mickelson, H. Xu, M. B. Stolpe, P. Naveau, B. Sanderson, I. Ebert-Uphoff, S. Samarasinghe, F. De Simone, F. Carbone, C. N. Gencarelli, J. M. Dennis, J. E. Kay, and P. Lindstrom. Evaluating lossy data compression on climate simulation data within a large ensemble. *Geoscientific Model Development*, 9(12):4381–4403, 2016.
- [7] R. Becker, H. Kapp, and R. Rannacher. Adaptive finite element methods for optimal control of partial differential equations: basic concepts. *SIAM J. Control Optim.*, 39:113–132, 2000.
- [8] R. Becker and R. Rannacher. A feed-back approach to error control in finite element methods: Basic analysis and examples. *East-West J. Numer. Math.*, 4:237–264, 1996.
- [9] C. Böhm, M. Hanzich, J. de la Puente, and A. Fichtner. Wavefield compression for adjoint methods in full-waveform inversion. *GEOPHYSICS*, 81(6):R385–R397, 2016.

- [10] A. Borzí and V. Schulz. *Computational Optimization of Systems Governed by Partial Differential Equations*. Computational Science and Engineering. SIAM, Philadelphia, 2012.
- [11] M. Burtcher and P. Ratanaworabhan. FPC: A high-speed compressor for double-precision floating-point data. *IEEE Trans. Comp.*, 58(1):18–31, 2009.
- [12] J. Calhoun, F. Cappello, L. Olson, M. Snir, and W. Gropp. Exploring the feasibility of lossy compression for pde simulations. *The International Journal of High Performance Computing Applications*, 33(2):397–410, 2019.
- [13] A. Cohen, L. M. Echeverry, and Q. Sun. Finite element wavelets. Technical report, Université Pierre et Marie Curie, Paris, 2000.
- [14] P. Colli Franzone, P. Deuffhard, B. Erdmann, J. Lang, and L. Pavarino. Adaptivity in space and time for reaction-diffusion systems in electrocardiology. *SIAM J. Sci. Comput.*, 28(3):942–962, 2006.
- [15] J. E. Dennis, Jr. and J. J. Moré. Quasi-Newton methods, motivation and theory. *SIAM Rev.*, 19(1):46–89, 1977.
- [16] P. Deuffhard, P. Leinen, and H. Yserentant. Concepts of an adaptive hierarchical finite element code. *IMPACT Comput. Sci. Engrg.*, 1:3–35, 1989.
- [17] P. Deuffhard and M. Weiser. *Adaptive numerical solution of PDEs*. de Gruyter, 2012.
- [18] S. Di and F. Cappello. Fast error-bounded lossy HPC data compression with SZ. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 730–739. 2016.
- [19] J. Diffenderfer, A. Fox, J. Hittinger, G. Sanders, and P. Lindstrom. Error analysis of zfp compression for floating-point data. *SIAM J. Sci. Comput.*, 41:A1867–A1898, 2019.
- [20] P. D. Düben, M. Leutbecher, and P. Bauer. New methods for data storage of model output from ensemble simulations. *Monthly Weather Review*, 147(2):677–689, 2019.
- [21] M. Emmett and M. Minion. Toward an efficient parallel in time method for partial differential equations. *Comm. Appl. Math. Comp. Sci.*, 7(1):105–132, 2012.
- [22] R. Filgueira, D. Singh, J. Carretero, A. Calderón, and F. García. Adaptive-compi: Enhancing MPI-based applications’ performance and scalability by using adaptive compression. *The International Journal of High Performance Computing Applications*, 25(1):93–114, 2011.
- [23] L. Fischer, S. Götschel, and M. Weiser. Lossy data compression reduces communication time in hybrid time-parallel integrators. *Comput. Vis. Sci.*, 19(1):19–30, 2018.

- [24] M. Gander. 50 years of time parallel time integration. In T. Carraro, M. Geiger, S. Körkel, and R. Rannacher, editors, *Multiple Shooting and Time Domain Decomposition Methods*, volume 9 of *Contributions in Mathematical and Computational Sciences*, pages 69–113. Springer, 2015.
- [25] L. Giraud, A. Haidar, and L. Watson. Mixed-precision preconditioners in parallel domain decomposition solvers. In U. Langer, M. Discacciati, D. Keyes, O. Widlund, and W. Zulehner, editors, *Domain Decomposition Methods in Science and Engineering XVII*, volume 60 of *Lecture Notes in Computational Science and Engineering*, pages 357–364. Springer, 2008.
- [26] S. Götschel. *Adaptive Lossy Trajectory Compression for Optimal Control of Parabolic PDEs*. Phd thesis, Freie Universität Berlin, Dept. Math. and Comp. Sci., 2015.
- [27] S. Götschel, N. Chamakuri, K. Kunisch, and M. Weiser. Lossy compression in optimal control of cardiac defibrillation. *J. Sci. Comp.*, 60(1):35–59, 2014.
- [28] S. Götschel, C. von Tycowicz, K. Polthier, and M. Weiser. Reducing memory requirements in scientific computing and optimal control. In T. Carraro, M. Geiger, S. Körkel, and R. Rannacher, editors, *Multiple Shooting and Time Domain Decomposition Methods*, pages 263–287. Springer, 2015.
- [29] S. Götschel, C. von Tycowicz, K. Polthier, and M. Weiser. Reducing memory requirements in scientific computing and optimal control. In *Multiple Shooting and Time Domain Decomposition Methods*, pages 263–287. Springer, 2015.
- [30] S. Götschel and M. Weiser. Lossy compression for PDE-constrained optimization: Adaptive error control. *Comput. Optim. Appl.*, 62:131–155, 2015.
- [31] S. Götschel, M. Weiser, and A. Schiela. Solving optimal control problems with the Kaskade 7 finite element toolbox. In A. Dedner, B. Flemisch, and R. Klöforn, editors, *Advances in DUNE*, pages 101–112. Springer, 2012.
- [32] A. Griewank. Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation. *Optim. Methods Softw.*, 1(1):35–54, 1992.
- [33] A. Griewank and A. Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. SIAM, Philadelphia, 2008.
- [34] R. Grout. Mixed-precision spectral deferred correction. Preprint CP-2C00-64959, National Renewable Energy Laboratory, 2015.
- [35] M. Kuhn, J. M. Kunkel, and T. Ludwig. Data compression for climate data. *Supercomputing Frontiers and Innovations*, 3(1):75–94, 2016.
- [36] P. Lindstrom. Fixed-rate compressed floating-point arrays. *IEEE Trans. Vis. Comp. Graphics*, 20(12):2674–2683, 2014.
- [37] P. Lindstrom. Error distributions of lossy floating-point compressors. In *Joint Statistical Meetings*, volume 2017, pages 2574–2589, 2017.

- [38] P. Lindstrom, P. Chen, and E.-J. Lee. Reducing disk storage of full-3d seismic waveform tomography (f3dt) through lossy online compression. *Computers & Geosciences*, 93:45–54, 2016.
- [39] P. Lindstrom and M. Isenburg. Fast and efficient compression of floating-point data. *IEEE Trans. Visual. Comput. Graphics*, 12(5):1245–1250, 2006.
- [40] O. Marina, M. Schanena, and P. Fischer. Large-scale lossy data compression based on an a priori error estimator in a spectral element code. Technical report, 2016. ANL/MCS-p6024-0616.
- [41] G. Martin. Range encoding: an algorithm for removing redundancy from a digitised message. Presented at Video & Data Recording Conference, Southampton, 1979.
- [42] J. D. McCalpin. Memory bandwidth and machine balance in current high performance computers. *IEEE Technical Committee on Computer Architecture (TCCA) Newsletter*, Dec 1995.
- [43] S. McKee. Reflections on the memory wall. In *Conf. Computing Frontiers*, pages 162–167, 2004.
- [44] S. Mertler, S. Müller, and C. Thole. Predictive principal component analysis as a data compression core in a simulation data management system. In *2015 Data Compression Conference*, pages 173–182, April 2015.
- [45] C. Nagaiah, K. Kunisch, and G. Plank. Numerical solution for optimal control of the reaction-diffusion equations in cardiac electrophysiology. *Comput. Optim. Appl.*, 49:149–178, 2011. 10.1007/s10589-009-9280-3.
- [46] J. T. Oden and S. Prudhomme. Goal-oriented error estimation and adaptivity for the finite element method. *Computers & Mathematics with Applications*, 41(5-6):735–756, 2001.
- [47] E. Otero, R. Vinuesa, O. Marin, E. Laure, and P. Schlatter. Lossy data compression effects on wall-bounded turbulence: bounds on data reduction. *Flow, Turbulence and Combustion*, 101(2):365–387, 2018.
- [48] R. Rannacher. On the adaptive discretization of PDE-based optimization problems. In M. Heinkenschloss and et al., editors, *PDE Constrained Optimization*. Springer, 2006.
- [49] D. Reed and J. Dongarra. Exascale computing and big data. *Comm. ACM*, 58(7):56–68, 2015.
- [50] A. Said and W. Pearlman. A new, fast, and efficient image codec based on set partitioning in hierarchical trees. *IEEE Trans. Circ. Syst. Video Tech.*, 6(3):243–250, 1996.
- [51] M. Schäfer, S. Turek, F. Durst, E. Krause, and R. Rannacher. Benchmark computations of laminar flow around a cylinder. In *Flow simulation with high-performance computers II*, pages 547–566. Vieweg+Teubner, 1996.

- [52] J. Schneck, M. Weiser, and F. Wende. Impact of mixed precision and storage layout on additive schwarz smoothers. Report 18-62, Zuse Institute Berlin, 2018.
- [53] P. Solin and D. Andrs. On scientific data and image compression based on adaptive higher-order FEM. *Adv. Appl. Math. Mech.*, 1(1):56–68, 2009.
- [54] R. Stevenson. Locally supported, piecewise polynomial biorthogonal wavelets on nonuniform meshes. *Constr. Approx.*, 19(4):477–508, 2003.
- [55] W. Sweldens. The lifting scheme: A construction of second generation wavelets. *SIAM J Math. Anal.*, 29(2):511–546, 1998.
- [56] D. Tao, S. Di, Z. Chen, and F. Cappello. Significantly improving lossy compression for scientific data sets based on multidimensional prediction and error-controlled quantization. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1129–1139, May 2017.
- [57] D. Tao, S. Di, X. Liang, Z. Chen, and F. Cappello. Improving performance of iterative methods by lossy checkpointing. In *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '18, pages 52–65, New York, NY, USA, 2018. ACM.
- [58] R. I. Teran, C.-A. Thole, and R. Lorentz. New developments in the compression of LS-DYNA simulation results using FEMZIP. 6th European LS-DYNA Users' Conference, 2007.
- [59] C.-A. Thole. Compression of LS-DYNA3D™ simulation results using FEMZIP©. 3. LS-DYNA Anwenderforum, 2004.
- [60] A. Toselli and O. Widlund. *Domain Decomposition Methods – Algorithms and Theory*, volume 34 of *Computational Mathematics*. Springer, 2005.
- [61] Y. M. Volin and G. M. Ostrovskii. Automatic computation of derivatives with the use of the multilevel differentiating techniques—1. algorithmic basis. *Comput. Math. Appl.*, 11(11):1099–1114, 1985.
- [62] C. von Tycowicz, F. Kälberer, and K. Polthier. Context-based coding of adaptive multiresolution meshes. *Computer Graphics Forum*, 30(8):2231–2245, 2011.
- [63] M. Weiser. On goal-oriented adaptivity for elliptic optimal control problems. *Optim. Meth. Softw.*, 28(13):969–992, 2013.
- [64] M. Weiser and S. Götschel. State trajectory compression for optimal control with parabolic PDEs. *SIAM J. Sci. Comp.*, 34(1):A161–A184, 2012.
- [65] S. Williams, A. Waterman, and D. Patterson. Roofline: an insightful visual performance model for multicore architectures. *Comm. ACM*, 52(4):65–76, 2009.
- [66] O. Zienkiewicz, R. Taylor, J. Zhu, and N. P. *The finite element method*. Elsevier Butterworth-Heinemann, 2005.