

LARGE-SCALE LOSSY DATA COMPRESSION BASED ON AN A PRIORI ERROR ESTIMATOR IN A SPECTRAL ELEMENT CODE

OANA MARIN, MICHEL SCHANEN, AND PAUL FISCHER

Abstract.

I/O is becoming an increasing bottleneck on large-scale computer systems. We present a highly scalable lossy data compression algorithm with controllable committed error based on an a priori estimator. The scientific data field is compacted via the discrete Chebyshev transform (DCT), truncated at a level within a user-specified tolerance, and subsequently compressed before writing to disk by using a version of Huffman encoding. The flexibility of the DCT transform and computational efficiency doubled by the a priori error estimator allows a dynamic compression ratio which can achieve as high as 97% compression for data visualization even in cases as complex as fully developed turbulent flow. In resilience problems however, the compression ratio is problem dependent. The algorithm is implemented in the spectral-element code Nek5000 and tested on highly turbulent large-scale simulation data of up to 10 billion degrees of freedom. The implementation via tensor products is highly efficient, leading to a decrease of flops in matrix-vector multiplications in three dimensions from $\mathcal{O}(N^6)$ down to $\mathcal{O}(N^4)$.

Key words. Large-scale, I/O, discrete cosine transform, discrete Chebyshev transform, Huffman encoding, lossy, computational fluid dynamics, error estimator

1. Introduction. Although the main challenge of high performance computing is to develop algorithms that scale well on increasingly many processors, the issue of handling and storing the generated large data sets is becoming a significant part of the computation. The performance in the various levels of the memory hierarchy is steadily diverging, making computations more memory bound and potentially I/O bound. Whereas cache access speeds have increased significantly, I/O speeds have not been able to keep up the pace. This work focuses on reducing the amount of data that has to be written to disk by applying lossy data compression. The compression ratio is selected optimally through the implementation of an a priori error estimator.

Data compression is a popular concept in image processing, where resolution and high frame rate have an immediate impact on the viewer, whereas the audience of scientific data compression is far less wide. One of the ideas imported from image processing was the discrete cosine transform, which has been the ground algorithm for JPEG image compression since the early 1970s [1]. More recently, same ideas were used in computational fluid dynamics simulations [16], but initially limited to Cartesian grids, extending later to nonconformal meshes.

Compression algorithms can be roughly classified as lossy, where data loss is allowed within given bounds, and lossless, where no error is incurred but the compression ratio is lower. The Discrete Chebyshev Transform, also known as Discrete Cosine Transform, and to which we shall refer here to as DCT, falls into the category of lossy compression algorithms, by which the energy of the signal is compacted in a convenient way that allows for optimal truncation. Other strategies in the same category rely on the discrete wavelet transform (DWT) [5, 18, 17] or floating point compression [9, 14]. Another approach for data compression of time varying sequences relies on building reduced order models, via proper orthogonal decomposition (POD) [13], [19]. This approach exploits the dynamics of the simulation. However, it may require a time series and the dynamics of the system may not always be properly captured as noted in [13]. These strategies can be complemented by a fast lossless compression algorithm [3] that instead of truncating, maps the data in such a way that the representation of probable sequences are shorter than the one of improbable

49 sequences.

50 In the present work DCT was chosen for its exceptional energy compactness prop-
51 erties as well as being computationally amenable. Although both DCT and DWT
52 address the nature of the data from a physical/mathematical perspective, DCT is far
53 easier to implement than DWT which tends to be problem dependent. Essential in
54 any lossy compression algorithm is to have control over the error, and most works
55 focus on the error in *max* or *rms* norm, as in [10]. Here, however, we focus on the L_2
56 norm on curvilinear grids which under the DCT transform leads to an easy to handle
57 expression for the truncation tolerance.

58 Gains in data representation due to compression become significant at large scales
59 and high dimensions and affect not only disk storage but also I/O speed. When used
60 solely for visualization purposes we can achieve a gain of 97% in compression ratio;
61 however, we intend to use this technique for a wider range of applications, such as
62 adjoint-based optimization [15], and to this end we have developed an a priori error
63 estimator that gives a clear account of how much data we can afford to truncate
64 in order to restore the solution with a given accuracy. Once the data is truncated,
65 the compression is performed on the actual bits of data. This is achieved through a
66 Huffman encoding that specifically targets the truncation symbol, here 0, as the one
67 with highest probability. The approach of the algorithm requires a scalable algorithm
68 that under optimal conditions increases I/O speeds in addition to decreasing file sizes.

69 This paper starts with a description of the data representation (section 2), fol-
70 lowed by an analysis of the properties of DCT (section 3). These properties of the
71 DCT lead to the derivation of the a priori error estimator. Concluding the theoretical
72 aspects of this work we outline the bitwise encoding strategy (section 4). The results
73 on three cases of the highest degree of difficulty are presented in section 5, together
74 with studies of the increase in I/O performance. In section 6 we briefly summarize
75 our conclusions.

76 **2. Data Representation of the Spectral-Element Method.** The spectral-
77 element method has a two-layered data representation, at the element level and inside
78 an element, which are populated with a set of points, roots of orthogonal polynomi-
79 als. Spectral-element methods may differ in their choice of element representation:
80 hexahedral, tetrahedral, and even hexagonal, or hybrids thereof. In terms of the spec-
81 tral representation inside each element, the most popular choices are Gauss Legendre
82 Lobatto points (GLL) [4] or Chebyshev points [2]. The code Nek5000 [6], which we
83 used to showcase our implementation, is based on GLL spectral grids and hexahedral
84 elements; however, for completeness we also tested the method on Chebyshev grids.

85 The entire domain Ω is thus split into elements Ω_e . The solution is required to be
86 continuous across each element boundary, class C^0 , and admits continuous derivatives
87 within an element, class C^1 ; for details see [4].

88 Consider an element Ω_e with inner data points $\mathbf{x} = (x_1, x_2, x_3)$. The solution can
89 be represented as a linear combination of orthogonal polynomials

90 (1)
$$\mathbf{u}(\mathbf{x})|_{\Omega_e} = \sum_{i,j,k} u_{ijk}^e \phi_i(x_1) \phi_j(x_2) \phi_k(x_3) ,$$

where ϕ are Lagrange polynomials and the data points \mathbf{x} are GLL points. Since
polynomials ϕ are orthogonal and also evaluate to unity at each grid point, we may
write via tensor products

$$\mathbf{u}|_{\Omega_e} \equiv (I_x \otimes I_y \otimes I_z) \mathbf{u}^e ,$$

2

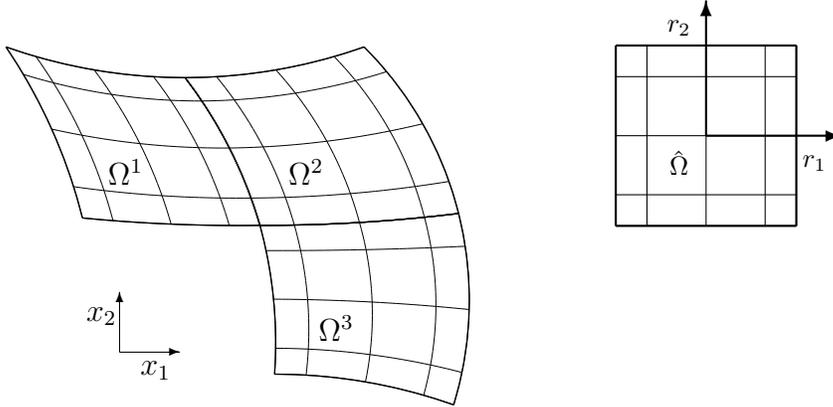


Figure 1: Spectral element mesh, 3 elements with 5 Gauss-Legendre-Lobatto points per element.

which is an identity. Although redundant in many ways, it gives an insight into the tensor product representation. Furthermore the solution over the entire mesh is represented as

$$\mathbf{u}(\mathbf{x})|_{\Omega_e} = \sum_{j=1}^M \mathbf{u}^j \delta_{je} ,$$

91 where δ_{je} is the Kronecker delta function

$$92 \quad (2) \quad \delta_{je} = \begin{cases} 1, & j = e \\ 0, & j \neq e \end{cases} .$$

93 In [Figure 1](#) we can distinguish between these two levels of representation, element-
 94 wise and GLL inner points. The parallelism of Nek5000 is always performed at the
 95 element level; in other words, GLL points interior to an element can never be split
 96 between two processors.

Without detailing on the numerical methods based on a variational formulation, we note that the spectral-element method, just like the finite-element method, does not seek a point-wise discrete solution \mathbf{u} that approximates the analytical solution $\bar{\mathbf{u}}$, but instead seeks a solution that minimizes the scalar product

$$\int_{\Omega} (\mathbf{u} - \bar{\mathbf{u}}) \mathbf{v}$$

97 for all test functions \mathbf{v} in a given polynomial space tailored to the problem at hand.
 98 With this in mind, a curvilinear element in Nek5000 (be it a part of either a structured
 99 or unstructured mesh) is mapped to a reference element where the spectral integration
 100 quadrature is defined. If we assume that there exists a mapping from an element Ω_e
 101 to a reference element $\hat{\Omega}_e$,

$$\mathbf{x}(\mathbf{r}) : \Omega_e \rightarrow \hat{\Omega}_e ,$$

and that any isoparametric deformation \mathbf{x} is

$$\mathbf{x}(\mathbf{r}) = \sum_{ijk} x_{ijk} \phi_i(r_1) \phi_j(r_2) \phi_k(r_3) ,$$

then a data field over the curvilinear mesh of discrete points \mathbf{x} takes the form

$$\mathbf{u}(\mathbf{x}) = \sum_{ijk} u_{ijk} \phi_i(r_1) \phi_j(r_2) \phi_k(r_3) ,$$

102 with r_1, r_2, r_3 coordinates of the reference element, as illustrated in [Figure 1](#).

103 However, if we are to compute derivatives and integrals of fields \mathbf{u} , we need to
 104 take into account the mapping $\mathbf{x}(\mathbf{r})$. For this work we are interested only in mappings
 105 and the L_2 norm, namely,

106 (3)
$$\int_{\Omega_e} \mathbf{u}^2 \, d\Omega_e = \int_{\hat{\Omega}_e} \mathbf{u}^2 J(\mathbf{r}) \, d\hat{\Omega}_e ,$$

with J being the Jacobian of the mapping $\mathbf{x}(\mathbf{r})$:

$$J(\mathbf{r}) = \det(X^r), \quad (X^r)_{ij} = \frac{\partial x_i}{\partial r_j}, \quad i, j = 1, \dots, d .$$

107 **3. Data Truncation Algorithm.** The truncation of the solution fields is per-
 108 formed on any type of data (e.g., velocity, pressure, temperature) in the same fashion
 109 since from the compression point of view the single most important issue is how over-
 110 resolved or underresolved the data is. For vector fields, such as velocity, we perform
 111 the truncation in each dimension independently.

112 **3.1. Discrete Cosine Transform.** The discrete cosine transform represents
 113 the amplitude of a signal in spectral space, by being the real part of a Fourier trans-
 114 form (the remainder, the discrete sine transform, representing the phase). In the
 115 current context, however, we do not make use of the connection between DCT and
 116 the fast Fourier transform, but rather by the correspondence of the DCT to the dis-
 117 crete Chebyshev transform, or the Karhunen-Loève (KL). In [12], it was shown that
 118 DCT is the optimal KL transform, having the best energy compaction efficiency for
 119 strongly correlated Markov processes.

120 Fast-forwarding to the core ideas of this work, we show that the DCT as applied
 121 to a signal $u(x)$ in a single dimension yields

122 (4)
$$w(x_j) = c_j \sum_{i=1}^N u(x_i) \cos \left[\frac{\pi}{2N} (2i-1)(j-1) \right], \quad j = 1, \dots, N$$

123 and

124 (5)
$$c_j = \begin{cases} \sqrt{1/N}, & j = 1, \\ \sqrt{2/N}, & 2 \leq j \leq N. \end{cases}$$

125 The DCT transform can also be represented in matrix form as

126 (6)
$$T_{ij} = c_j \cos \left[\frac{\pi}{2N} (2i-1)(j-1) \right] ,$$

127 leading to $w = Tu$, where T is an $N \times N$ matrix.

128
129
130
131
132
133
134
135

Properties of the DCT transform.

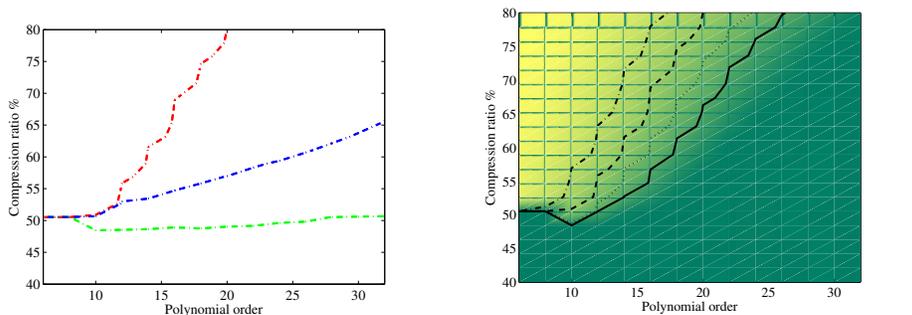
1. Orthogonality: $T^T T = T T^T = I$, where I is the identity matrix.
2. $T^{-1} = T^T$: thus the inverse DCT transform IDCT is the transpose of the DCT.
3. Energy preservation: $w^T w = (Tu)^T (Tu) = u^T T^T Tu = u^T u$.
4. Energy compactness.
5. Separability: the three-dimensional DCT (let us define it $T^3(\mathbf{u})$) is a sequence of one-dimensional DCT transforms; that is, $T^3(\mathbf{u}) = (T \otimes T \otimes T)\mathbf{u}$.

To restore a data field from DCT space to the real space, we can apply the inverse

$$u = T^{-1}w .$$

136 Given that $T^{-1} = T^T$ is sufficient to compute only the direct transform, the inversion
137 process follows naturally, via transposition.

138 **3.2. Prerequisites.** To assess whether a DCT-based implementation may yield
139 satisfactory results, and to compare with the JPEG approach, we first performed
140 a short study of a synthetically generated signal of high frequency on the reference
141 element $[-1, 1]$: here $u(x, y) = \sin(4x) \cos(2y) + \cos(5x) \sin(5y)$. To this end we
142 applied the DCT transform to the signal represented on a Chebyshev grid, on a
143 Gauss-Legendre-Lobatto grid, and on an equidistant grid. We sought to determine
144 what compression ratio can be achieved such that we can restore the solution with
145 a fixed error. Here we considered $\epsilon = 10^{-6}$, for increasing grid sizes. Note that a
146 grid of N equidistant discrete points corresponds to polynomial order $N - 1$ on GLL
147 and Chebyshev grid representations. In Figure 2a one can observe that in regions
148 where the signal is underresolved (up to polynomial order ≈ 10) the behavior between
149 grids does not differ; however, with increasing resolution one can truncate more on
150 Chebyshev grids than on any others, followed by GLL grids and lastly by equidistant
151 grids. We note that different signals may yield different results: in some scenarios
152 the Chebyshev grids and GLL grids have a similar trend whereas equidistant grids
preserve a very slow decay of the error.



(a) For a fixed error (10^{-6}) comparing compression ratio vs grid size/polynomial order (N): green: equidistant grid, blue: GLL grid, red: Chebyshev grid. (b) Color plot of the error on a Chebyshev grid, contour plot of various magnitudes, in direction of arrow - increasing accuracy (error from 10^{-4} to 10^{-10}).

Figure 2: Two-dimensional error assessment of a synthetically produced signal.

153

154 For more insight into the qualitative behavior of DCT, now only on Chebyshev
 155 grids, we generated a set of compression ratios and polynomial orders to find the
 156 errors in each regime as illustrated in Figure 2b. Since our code is a GLL based code,
 157 we had to map the data to Chebyshev grids, prior to applying the DCT (although
 158 this can be done in a single step, as will be shown). However, comparative studies
 159 on real three-dimensional cases of fully turbulent flow did not yield great differences
 160 between GLL and Chebyshev grids, since the signal is already packed on the mesh by
 161 design.

162 **3.3. Tensor Products.** The separability property of the DCT transform makes
 163 it an efficient tool from a computational viewpoint. Because of separability it can be
 164 applied on a higher-dimensional field as a tensor product of one-dimensional ones,
 165 thus lowering the computational costs of any evaluation.

Consider $A : \mathbb{R}^M \rightarrow \mathbb{R}^M$, $B : \mathbb{R}^N \rightarrow \mathbb{R}^N$, $C : \mathbb{R}^P \rightarrow \mathbb{R}^P$.

$$\mathbf{w} = (A \otimes B)\mathbf{u} = \mathbf{A}\mathbf{u}B^T .$$

In a three dimensional setup this translates into

$$\mathbf{w} = (A \otimes B \otimes C)\mathbf{u} = (A \otimes B)\mathbf{u}C^T = \mathbf{A}\mathbf{u}C^T B^T .$$

In index notation the two dimensional product $\mathbf{w} = (A \otimes B)\mathbf{u} = \mathbf{A}\mathbf{u}B^T$ can be written
 as

$$w_{ij} = \sum_{k=1}^N \sum_{l=1}^N a_{ik} u_{kl} b_{lj}, \quad B = b_{jl} .$$

166 This amounts to $\mathcal{O}(N^3)$ operations as $\mathcal{O}(N^4)$ if the matrix $A \otimes B$ is set up explicitly.
 167 For the three-dimensional case the advantage is even bigger, yielding

$$168 \quad (7) \quad w_{ijk} = \sum_{l=1}^N \sum_{m=1}^N \sum_{p=1}^N a_{il} u_{lmp} c_{pk} b_{mj}, \quad B = (b)_{jm,j,m=1,\dots,N}, \quad C = c_{kp,k,p=1,\dots,N} .$$

169 Analyzing the expression in (7), we note that each matrix-matrix product neces-
 170 sitates N^3 flops. We have two such products, and the last one has to be performed N
 171 times, leading to a total count of $N^3 + N^3 + N^4$. This reduces the evaluation costs from
 172 $\mathcal{O}(N^6)$ operations (if the entire matrix is explicitly constructed) to $\mathcal{O}(N^4)$ operations.
 173 This aspect of separability, which allows for tensor product representations, becomes
 174 important with increasing dimension since the evaluation times go from $\mathcal{O}(N^{2d})$ to
 175 $\mathcal{O}(N^{d+1})$, which is clearly more advantageous for higher dimensions d . Now we apply
 176 the DCT transform T in all three directions and denote them, respectively as $T_x, T_y,$
 177 T_z , although $T_x = T_y = T_z = T$, we have

$$178 \quad (8) \quad \mathbf{w} = (T_x \otimes T_y \otimes T_z)\mathbf{u} = T_x \mathbf{u} T_z^T T_y^T$$

and in reverse

$$\mathbf{u} = (T_x \otimes T_y \otimes T_z)\mathbf{w} = T_x^T \mathbf{w} T_z T_y .$$

Furthermore if we want to map the signal to a Chebyshev grid, it also suffices to
 have one forward interpolation operator, denoted M^{gc} , from the GLL grid to the
 Chebyshev grid. Now the transform T is applied not to u but to $M^{gc}u$ and in one
 dimension results in

$$w = TM^{gc}u ,$$

179 while in three dimensions the rule in (8) remains unchanged since $T \rightarrow TM^{gc}$.

180 **3.4. A Priori Error Estimator.** The truncation of the signal (e.g. velocity
 181 field) occurs in DCT space after the signal has been decorrelated. However, in order
 182 to maintain control over the error incurred through the truncation of the flow field,
 183 it is necessary to devise an a priori error estimator that guarantees that the data can
 184 be retrieved with a desired accuracy. As will be shown shortly the L_2 norm in DCT
 185 space is equivalent to the L_2 norm in real space; although the norm is imposed locally,
 186 it is also satisfied globally.

187 *Local to global error.* The entire domain Ω consists of elements Ω_e , such that
 188 $\Omega = \cup_e \Omega_e$. The global velocity field is therefore split into its local restrictions $\mathbf{u}_e =$
 189 $\mathbf{u}|_{\Omega_e}$. Because of the nature of our computational domains, which are curvilinear
 190 with nonuniform elements, we need to consider the norm weighted by $V = \int_{\Omega} d\Omega$, the
 191 volume of the domain, or element, namely,

$$192 \quad (9) \quad \|\mathbf{u}\| = \sqrt{\frac{\int_{\Omega} \mathbf{u}^2 d\Omega}{\int_{\Omega} d\Omega}} = \sqrt{\frac{\|\mathbf{u}\|_{L_2}^2}{V}} .$$

By virtue of the spectral element domain decomposition and using $\mathbf{u}|_{\Omega_e} = \mathbf{u}_e$ we can write the global L_2 norm as a sum of local norms:

$$\|\mathbf{u}\|_{L_2}^2 = \sum_e \|\mathbf{u}_e\|_{L_2}^2 .$$

This translates in terms of the norm weighted by the volume as

$$\frac{\|\mathbf{u}\|_{L_2}^2}{V} = \frac{1}{V} \sum_e \frac{\|\mathbf{u}_e\|_{L_2}^2}{V_e} V_e ,$$

193 which once again with the current definition of the norm, (9), is

$$194 \quad (10) \quad \|\mathbf{u}\|^2 = \frac{1}{V} \sum_e \|\mathbf{u}_e\|^2 V_e .$$

195 Regarding the truncation error incurred by $\tilde{\mathbf{u}}$, if we impose a threshold on the global
 196 solution $\|\mathbf{u} - \tilde{\mathbf{u}}\| < \epsilon$, then from (10) it is sufficient to impose a similar local norm
 197 $\|\mathbf{u}_e - \tilde{\mathbf{u}}_e\| < \epsilon$.

Truncation in DCT space. Assuming a spectral discretization in Gauss-Legendre-Lobatto points, with weights gathered in the mass matrix B , the L_2 norm in one dimension becomes

$$\|u\|_{L_2} = \sqrt{\int_{\Omega} u^2 d\Omega} \approx \bar{u}^T B \bar{u} .$$

For clarity we denote here as \bar{u} the discrete correspondent of the velocity field u . Now if we transfer the field \bar{u} into spectral space via DCT, we have $\bar{w} = T\bar{u}$:

$$\|w\|_{L_2} = \sqrt{\int_{\Omega} w^2 d\Omega} \approx \bar{w}^T B \bar{w} = \bar{w}^T T^T B T \bar{w} .$$

198 From the properties of DCT we have that $T^T T = I$; and since the mass matrix is
 199 a diagonal matrix, we also have $T^T B T = B$. This reasoning is performed in one
 200 dimension; however, because of the separability of DCT, it can be easily transported
 201 to a higher dimension via tensor products.

202 In conclusion, although the error should be cautiously measured locally in DCT
 203 space, the result is identical to that in real space.

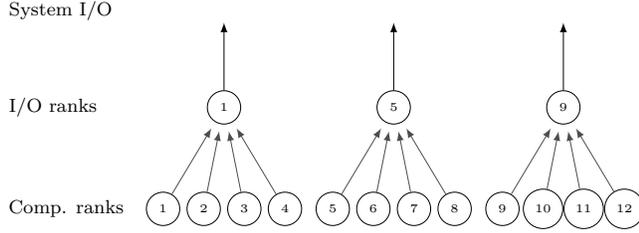


Figure 3: I/O write operation using 12 processes with 3 I/O ranks.

4. Compression Algorithm. The number of entries within one spectral element is $M = P^d$, where d is the spatial dimension and P the polynomial order. Based on DCT and the a priori error estimator, we are able to truncate K entries of \mathbf{u} by setting each one to zero. This leaves us with $N = M - K$ nonzero entries. We assume that each entry takes m bits in its binary encoding $e : \mathbb{R} \rightarrow \mathbb{B}^m$, where

$$x \mapsto \underbrace{[b_0, b_1, \dots, b_{m-2}, b_{m-1}]_2}_{m \text{ bits}},$$

with \mathbb{B} being the space of binary numbers. One example of such an encoding is the binary numbers representation where $x = \sum_{i=0}^{m-1} b_i \cdot 2^i$. With double precision floating-point number encoding as defined by IEEE have, we $m = 64$ bits. We want a compression algorithm that is efficient in terms of both performance and compression ratio and, flexible with regard to the encoding e . Without loss of generality we assume that $e(0) = [0, \dots, 0]_2$. We propose the compression encoding h :

$$h([b_0, b_1, \dots, b_{m-2}, b_{m-1}]_2) = \begin{cases} \underbrace{[0]_2}_{1 \text{ bit}}, & \text{if } b_0 = \dots = b_{m-1} = [0]_2, \\ \underbrace{[1, b_0, b_1, \dots, b_{m-2}, b_{m-1}]_2}_{m+1 \text{ bits}}, & \text{else.} \end{cases}$$

The entries of value zero are all compressed by using the single bit 0 for their encoding. Every other value is being prepended with the single bit 1. It is a simplified Huffman encoding where we distinguish only two types of symbols: the zeros that are being compressed and the nonzeros. Of course, to make this encoding achieve high compression ratios, we assume that zero is by far the most prevalent number.

We define the optimal zero compression ratio as $\frac{K}{M}$, where all the zeros have zero size. Using our encoding, we end up with $K + M \cdot (m + 1)$ and thus a compression ratio of

$$\frac{K + (M - K) \cdot (m + 1)}{M \cdot m}.$$

For $K = 0$, we end up with an increase of M bits, whereas with $K = M$ our highest compression ratio is $\frac{1}{m}$. It follows that this encoding is useful only if $K > M/m$. The complexity of this encoding is linear in the number of total grid points n totalling to $\mathcal{O}(n \cdot d)$ with d being the dimension.

4.1. Compressed I/O in Nek5000. I/O has a wide range of use cases, including visualization, resilience, and adjoint checkpointing. It represents the lowest layer

222 of the memory hierarchy where the data is written on permanent storage devices (e.g.,
 223 disks). Because it is the last memory layer, it generally has the slowest bandwidth and
 224 the highest latency of all the memories involved in the computation. This bottleneck
 225 poses great challenges for large-scale computer systems, and the amount of output
 226 data should be selected wisely.

227 One way of addressing this bottleneck is to adapt the hardware to the access
 228 patterns. Burst buffers, for example, assume short peaks of high disk access. Another
 229 alternative is to compress and reduce the amount of data. In Nek5000, the data
 230 consists in its simplest form of the velocity vector field and the pressure scalar field.
 231 Because of the continuity we assume that these two fields are an excellent candidate
 232 for the DCT truncation described in [section 3](#), while ignoring the effects of shocks.

233 The I/O implementation of Nek5000 is shown in [Figure 3](#). It distinguishes I/O
 234 *ranks* as a subset of *compute ranks*. For brevity reasons we focus only on the write
 235 operation, which is fundamentally similar to reads and more frequent. After a write
 236 operation is initiated, every compute rank sends its data to the I/O ranks, which then
 237 dispatch the data to the system I/O API (i.e., `fwrite`).

238 Our compression algorithm is made up of two stages. One is the truncation
 239 relying on DCT (see [section 3](#)), and the other is the actual compression of the zeros
 240 created by the truncation (see [section 4](#)). To achieve maximum scalability, we apply
 241 the truncation on the compute nodes. Consequently, it would also be desirable to
 242 apply the compression on the compute nodes. However, this design raises flexibility
 243 concerns. One data segment i is compressed from full-length n_i to the compressed-
 244 length c_i and written out serially segment after segment. To know the length c_i of the
 245 compressed segments, we also store the length c_i at the beginning of each compressed
 246 data segment. If the data is compressed on the compute nodes, the I/O nodes collect
 247 the various compressed data segments together with the prepended c_i 's and write them
 248 to disk. Nek5000 is able to be run with different numbers of nodes using the same data.
 249 Let us assume that data has been compressed and written using n compute nodes.
 250 If this data is read and then distributed, for example, among $2n$ compute nodes, this
 251 distribution becomes nontrivial, because the compressed fragments cannot easily be
 252 split up in a scalable way. Hence, we decided to do the data compression only on
 253 the I/O nodes. In that case, and for the same reason, the compressed data must be
 254 read with the same number of I/O nodes as it has been written. With this process of
 255 course, the compression scales only with the number of I/O nodes, which, as we will
 256 see later, may lack scalability, depending on the architecture.

257 **5. Implementation and Results.** The current work is independent of the na-
 258 ture of the data fields. However, compression is applied here to flow fields from
 259 turbulent simulations. This choice is justified by our numerical experiments which
 260 showed that fully turbulent flow fields are the hardest to compress since the signal
 261 usually has high frequencies and is fully resolved. On the other hand signals that
 262 are overresolved can be highly compressed, and this implies fields stemming from
 263 Reynolds-averaged Navier-Stokes, laminar flows, and so forth.

264 Both cases selected here are results of simulations using the incompressible Navier-
 265 Stokes, which in nondimensional formulation are given by

$$266 \quad (11) \quad \frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\nabla p + \frac{1}{Re} \nabla^2 \mathbf{u} + \mathbf{f}$$

$$267 \quad \nabla \cdot \mathbf{u} = 0 ,$$

268 where \mathbf{u} is the flow velocity, p the pressure, and Re the Reynolds number given as

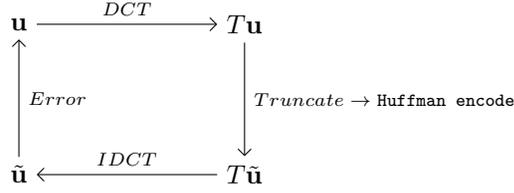


Figure 4: Workflow for compression and restoring the solution.

269 $Re = UD/\mu$ (μ being the viscosity).

Algorithm 1 Parallel compression on an already partitioned mesh with `nel` elements each.

```

procedure SETUP
  T ← DCT_setup_1d(p)
end procedure
procedure TRUNCATE
5:   for 0 ≤ k < nel do
      uDCT,k = T · uk · TT · TT
      sort(uDCT,k)
      utrunc,k = uDCT,k > ε
    end for
10: end procedure
procedure COMPRESS
  if IOnode then
    for q ∈ IOchildren do
      Recv(utrunc, q)
15:   ucompress = huff_encode(utrunc)
      output(ucompress)
    end for
  else
    Send(utrunc, IOparent)
20: end if
end procedure

```

270 **5.1. Algorithm.** Combining the DCT truncation (see [section 3](#)), the Huffman-
 271 based compression (see [section 4](#)), the error estimator (see [subsection 3.4](#)), and the I/O
 272 layout in Nek5000 (see [subsection 4.1](#)), we derive the parallel algorithm in [Algorithm 1](#)
 273 and its implementation. The parallel algorithm is described on an already partitioned
 274 data \mathbf{u} , each process executing it locally.

275 The `Setup` procedure is called once at the startup of Nek5000. It creates the DCT
 276 operator \mathbf{T} , which depends on the element grid points (line 2) and ultimately only on
 277 the polynomial order $\mathbf{p}=\mathbf{N}-1$. The operator \mathbf{T} is of size $\mathbf{N}\cdot\mathbf{N}$, and thus its implementation
 278 via tensor products renders its computation extremely fast.

279 The `Truncate` and `Compress` procedures are called one after another each time an
 280 output is initiated. Every process iterates over every element \mathbf{k} and applies the three-
 281 dimensional DCT using the tensor product implementation of \mathbf{T} (line 6). Subsequently,
 282 the resulting DCT transformed array \mathbf{u}_{DCT} is sorted and then truncated based on the

283 error estimator (line 8). No communication is involved in this process, making it
284 embarrassingly parallel.

285 In `Compress` the Huffman encoding is applied at the I/O node level. If the process
286 is an I/O node (see [Figure 3](#)), it iterates over all its children and receives the truncated
287 data of every child `q` (line 14). It then applies the Huffman encoding, as seen in
288 [section 4](#), and writes the output of child `q` to disk. If the process is not an I/O node it
289 just sends the data to its I/O parent `IOparent` (line 19). The number of messages and
290 thus its latency depends on the number of children in `IOchildren` (line 13). Assuming
291 this ratio is constant for a given system and architecture, the compression step is also
292 embarrassingly parallel at the I/O node level.

293 The uncompressed I/O uses the same communication structure for collection the
294 data as in the routine `COMPRESS`. To achieve a speedup in the I/O using the com-
295 pression, the `for` loop in lines 13-17 has to become faster than in the uncompressed
296 case. All the operations in that loop, `Recv`, `huff_encode`, and `output` are supposed to
297 scale linearly with the data size. Thus, if the performance gain of the compression in
298 `output` is high enough to account for the additional time needed for the encoding, we
299 experience a speedup. These parameters are highly system dependent. `huff_encode`
300 is a highly memory bandwidth bound routine, whereas `output` is I/O bound. The
301 current development in diverging memory access and I/O speeds gives us confidence
302 that future architectures will increase the benefits of our approach.

303 **5.2. Results.** The first case is a three-dimensional direct numerical simulation
304 of flow past an airplane wing [7] at the high Reynolds number of 400.000 based on
305 freestream velocity and cord length. To fully resolve the turbulent scales requires 3.2
306 billion grid points (1.847.664 elements), making this a typical large-scale computing
307 application. This flow case, as seen in [Figure 5a](#), is not fully turbulent in the en-
308 tire domain, and regions with less fluctuations can be compressed more than their
309 counterparts.

310 The second case, involving turbulent flow, in a rod bundle of 37 pins [11], is
311 the result of a large eddy simulation, illustrated in [Figure 5b](#). This simulation was
312 performed on 10 billion grid points on an unstructured mesh, and the flow is turbulent
313 in the entire domain, leading to lower compression rates. The data file in this case
314 was available in single precision, which is common in such types of simulations when
315 compression is not available. Data stored in single precision is widely used since most
316 postprocessing is done at runtime, with little need for full velocity fields. The few
317 applications that may need the data for postprocessing are not as sensitive to the lack
318 of accuracy induced by single-precision data.

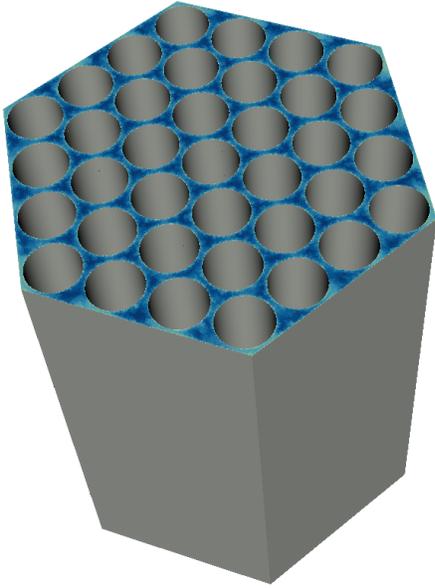
319 The third case is the canonical example of turbulent flow in a straight pipe at a
320 friction Reynolds number of $Re_\tau = 180$, studied in detail in [8]. This is the smallest
321 case, 36.480 elements with approximately 18 million gridpoints.

322 The implementation follows [Algorithm 1](#), or the schematic view [Figure 4](#), by which
323 we apply the DCT transform, then truncate, and compress. Note the distinction we
324 make here between truncation, which is the mathematical procedure of setting to
325 zero the information that is deemed redundant in the signal (i.e. above a certain
326 tolerance), and compression, which is the actual bitwise encoding that removes these
327 zeros from the data set.

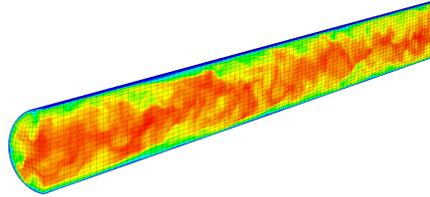
328 We defined and tested a set of tolerances in the range of 10^{-i} with $i = 1, \dots, 13$
329 and compressed optimally based on the a priori estimator in [subsection 3.4](#). Assessing
330 the compression ratio versus loss of accuracy as in [Figure 6a](#) and [Figure 7a](#) we note
331 that even in the most challenging case of fully developed turbulence, one can compress



(a) Flow past an airplane wing (≈ 3.2 billion gridpoints).



(b) Flow in a rod bundle (≈ 10 billion gridpoints).



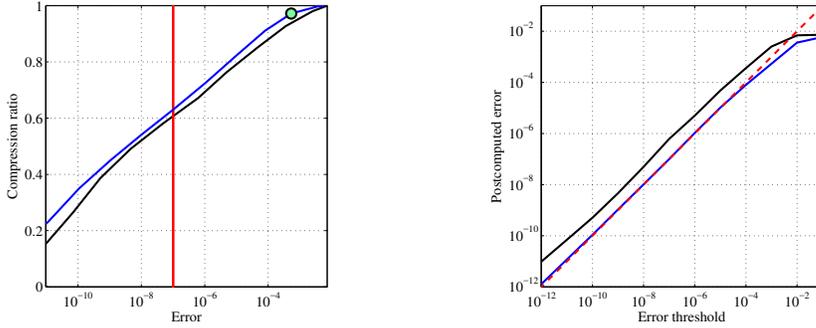
(c) Flow in a pipe at $Re_\tau = 180$.

Figure 5: Velocity magnitude of different flow configurations.

332 more at higher or equal accuracy than by storing data in single precision. The case of
 333 the flow in a rod bundle for which the available data was in single precision allows for
 334 less compression (see [Figure 7b](#)) than its fully turbulent counterpart in (see [Figure 7a](#)).
 335 We attribute this difference to the noise induced in double-precision operations by the
 336 single-precision format.

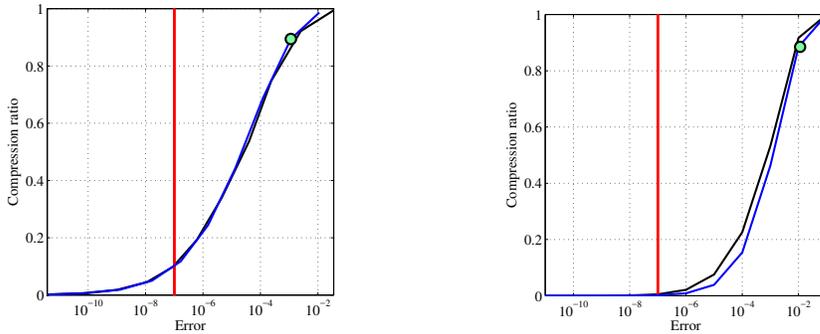
337 The use of compression is envisioned both for file restarts at a minimally necessary
 338 accuracy and visualizations. Nonetheless, the requirements for visualization seem to
 339 be low. An accuracy of 10^{-2} , corresponding to 97% compression, is consistently suffi-
 340 cient for satisfactory visualizations, as illustrated in [Figure 8a](#), [Figure 8b](#), [Figure 9a](#),
 341 and [Figure 9b](#).

342 **5.3. Data Compression for Resilience.** Data compression can be exploited
 343 beyond visualizations, because it is being used as a restart file for subsequent time



(a) Compression ratio (C_r) vs error: (blue) GLL grid, (black) Chebyshev grid, (green marker) corresponds to visualization in Figure 9b. (b) A priori vs a posteriori error: (blue) GLL grid, (black) Chebyshev grid.

Figure 6: Flow past an airplane wing



(a) Flow in a pipe: (blue) GLL grid, (black) Chebyshev grid, (green marker) corresponds to visualization threshold. (b) Flow in a rod bundle: (blue) GLL grid, (black) Chebyshev grid, (green marker) corresponds to visualization in Figure 8b.

Figure 7: Compression ratio (C_r) vs error.

344 -dependent computations, keeping in mind that the impact of initial conditions may
 345 be overarching. Lossy data compression induces a truncation error, which depending
 346 on the sensitivity to initial conditions may be of greater or lesser relevance. For
 347 example, transition to turbulence can trigger a sudden decay in kinetic energy at
 348 even small errors, where fully developed turbulence leads to the same statistical state
 349 even for larger errors. At high errors we also expect an impact on code efficiency,
 350 since truncations in data may lead to a higher iteration count in the algebraic solver
 351 for the first timesteps after restart. A full study of the impact of data loss for file
 352 restarts is therefore highly problem dependent.

353 The a priori error estimator is the stepping stone for any restart since it is universal
 354 and relies only on the signal quality. To assess how well the a priori error estimator
 355 agrees with the a posteriori one, we have compared it in Figure 6b. Both DCT
 356 transforms on GLL and Chebyshev grids follow well the imposed tolerance, with a

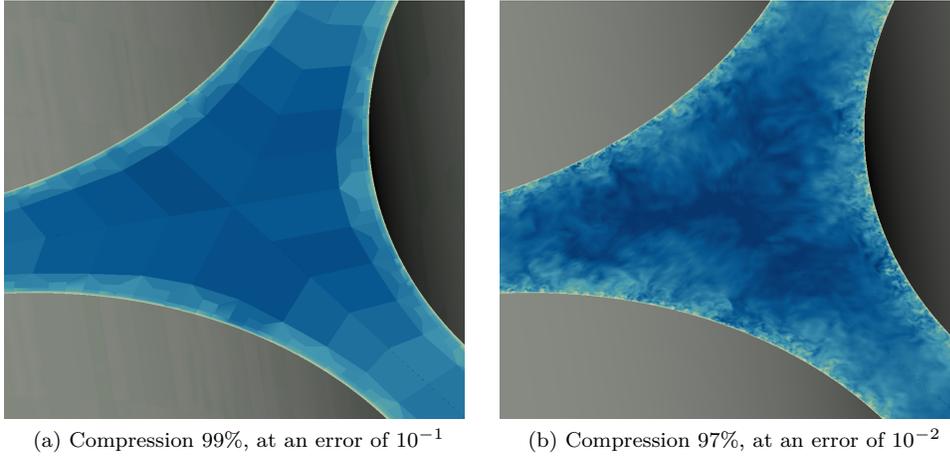


Figure 8: Visualization retrieved at different compression ratios for flow in a rod bundle.

357 smaller deviation on the Chebyshev grid. For noisy data fields, however, the estimator
 358 deviates a bit more; see [Figure 10b](#), where the data was available in single-precision
 359 compared with double-precision fields in [Figure 10a](#).

360 If any loss in data may be detrimental to the final solution, the alternative is to
 361 use the compression only for visualizations, where it performs exceptionally well, and
 362 overwrite the checkpoints as uncompressed data at the end of every new successful
 363 computation.

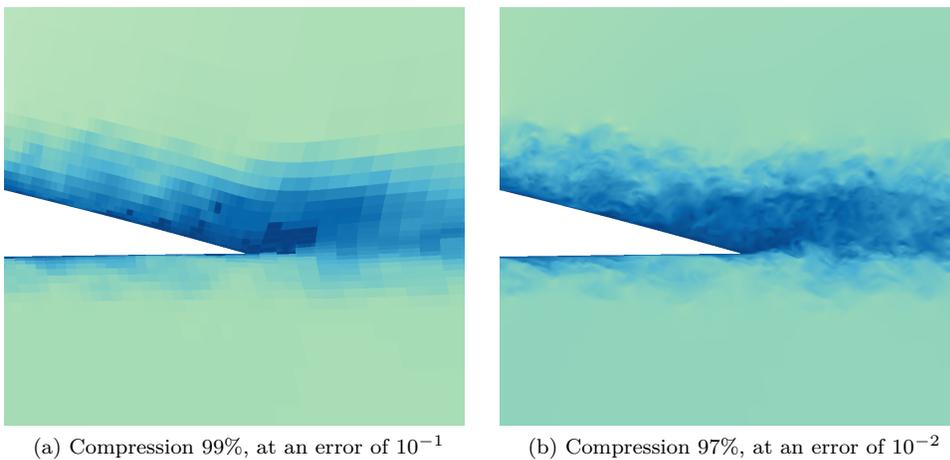
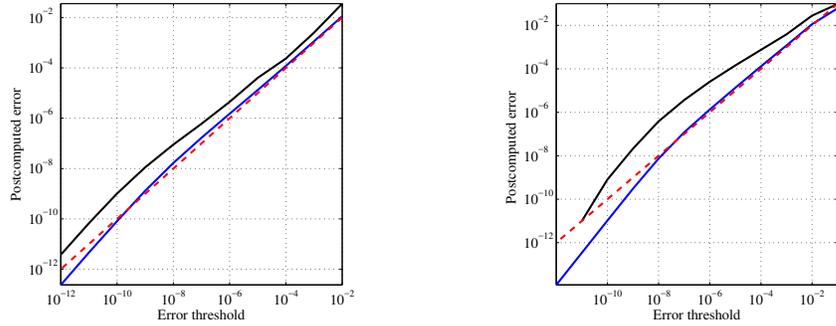
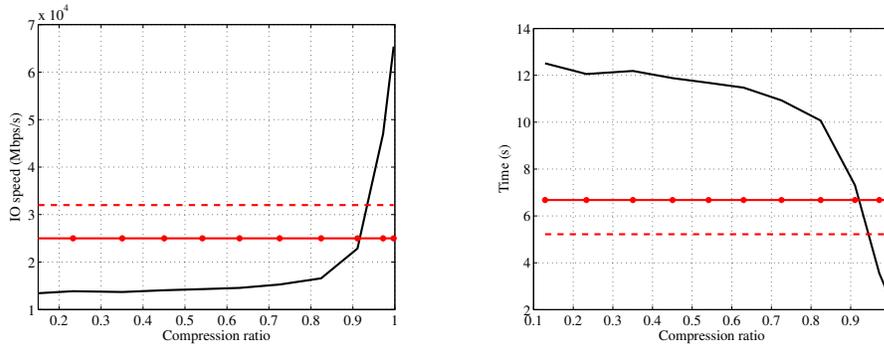


Figure 9: Visualization retrieved at different compression ratios for flow around a wing.



(a) Flow in a pipe: (blue) GLL grid, (black) Chebyshev grid, (red dotted) imposed tolerance. (b) Flow in a rod bundle case: (blue) GLL grid, (black) Chebyshev grid, (red dotted) imposed tolerance.

Figure 10: A posteriori error vs imposed tolerance on a priori error estimate.



(a) I/O speed in Mbps/s vs compression ratio using 65,536 processes with 1,024 I/O nodes for the uncompressed and 2,048 I/O nodes for the compressed write, (red dotted) theoretical bandwidth limit, (red continuous) speed for uncompressed data. (b) Time to write in seconds versus compression ratio, (red dotted) theoretical bandwidth limit, (red continuous) speed for uncompressed data.

Figure 11: I/O performance on the flow past an airplane wing case.

364 **5.4. I/O Performance.** The implementation of the I/O has been described
 365 in [subsection 4.1](#). We benchmarked the I/O speeds on 1,024 Mira BG/Q nodes at
 366 Argonne using 65,536 processes, called compute nodes in this paper. Each Mira node
 367 is composed of 16 cores. Nek5000 was run with 32 processes per such node, with two
 368 processes per core. Every 128 Mira nodes are physically connected to one I/O node at
 369 4 GB/s. That 4 GB/s connection is attached to a network switch and the backbone,
 370 which eventually leads to the disks. System noise due to other running applications
 371 may prevent full usage of the 4 GB/s at the I/O node level. Thus, at 1,024 nodes we
 372 may expect a maximum of 32 GB/s assuming no interference with other jobs.

373 The number of abstract I/O nodes in Nek5000 does not necessarily correspond to

Table 1: Overview of runtime parameters

Data	Mira Nodes	Nek I/O Nodes	Nek Nodes
Uncompressed	2,048	1,024	65,536
Compressed	2,048	2,048	65,536

374 the number of physical I/O nodes on Mira. To determine the maximum write speed,
 375 we wrote data at various numbers of I/O nodes ranging from 128 to 2,048 with 10
 376 measurements each. The peak performance was achieved by using 1,024 I/O nodes
 377 for the uncompressed data and 2,048 nodes for the compressed data. The parameters
 378 of the runs are summarized in Table 1, and the performance is plotted in Figure 11a.

379
 380 The cost of writing compressed data has been at most 2, with a compression
 381 ratio below 0.2 compared with an uncompressed write. At a compression ratio above
 382 0.9 an I/O speed advantage can be observed. This would be the case mainly with
 383 visualization data. We expect that further optimizations in the implementation should
 384 lead to an I/O speed increase at lower compression ratios.

385 **6. Conclusions.** We have implemented a scalable lossy data compression in a
 386 spectral-element code and controlled the data compression via an a priori error esti-
 387 mator. The lossy data algorithm has been inspired by the DCT transform used by the
 388 JPEG image compression algorithm. However, we have identified that the compres-
 389 sion ratio may achieve higher ratios on Chebyshev and GLL grids than on equidistant
 390 grids, as in image compression. Although our method has been demonstrated on fluid
 391 dynamics data, it is applicable to other types of data. The a priori error estimator is
 392 based on the L_2 norm, which is a far more suitable norm for continuous scientific data
 393 than is the *rms* norm used in image compression. The algorithm has the outstanding
 394 performance for visualization, where it is possible to retain the same visualization
 395 quality using only 3% of the initial data. The compression of the data on disk is
 396 achieved via Huffman encoding, which leads not only to reduced sizes in data but
 397 also to faster I/O speeds by saturating the bandwidth at a higher speed. The imple-
 398 mentation of the DCT not only is flexible but also, because of the separability of the
 399 DCT, leads to an efficient implementation via tensor products, thereby lowering the
 400 computational cost for three-dimensional data from $\mathcal{O}(N^6)$ to $\mathcal{O}(N^4)$.

401 **Acknowledgments.** We acknowledge the support of Philipp Schlatter (KTH),
 402 Ricardo Vinuesa (KTH), Elia Merzari (ANL), and Nicolas Offermans (KTH). This
 403 research used resources of the Argonne Leadership Computing Facility, which is a
 404 DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.
 405 This material was based upon work funded by the U.S. Department of Energy, Office
 406 of Science, under contract DE-AC02-06CH11357

407 The submitted manuscript has been created by the University of Chicago as Operator of Argonne National Laboratory ("Argonne") under Contract No. DE-AC02-06CH11357 with the U.S. Department of Energy. The U.S. Government retains for itself, and others acting on its behalf, a paid-up, nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government.

- 409 [1] N. AHMED, T. NATARAJAN, AND K. R. RAO, *Discrete cosine transform*, IEEE Transactions on
410 Computers, C-23 (1974), pp. 90–93.
- 411 [2] J. P. BOYD, *Chebyshev and Fourier spectral methods*, Courier Corporation, 2001.
- 412 [3] Y. COLLET, *Lz4: Extremely fast compression algorithm*, code. google. com, (2013), [http://](http://cyan4973.github.io/lz4/)
413 cyan4973.github.io/lz4/.
- 414 [4] M. O. DEVILLE, P. F. FISCHER, AND E. H. MUND, *High-Order Methods for Incompressible Fluid*
415 *Flow*, Cambridge University Press, 2002, <http://dx.doi.org/10.1017/CBO9780511546792>.
416 Cambridge Books Online.
- 417 [5] G. ERLEBACHER, M. Y. H. DIRECTOR, ET AL., *Wavelets: Theory and Applications: Theory*
418 *and Applications*, Oxford University Press, 1995.
- 419 [6] P. FISCHER, J. LOTTES, S. KERKEMEIER, O. MARIN, K. HEISEY, A. OBABKO, E. MERZARI, AND
420 Y. PEET, *Nek5000: User's manual*, Tech. Report ANL/MCS-TM-351, Argonne National
421 Laboratory, 2015.
- 422 [7] S. M. HOSSEINI, R. VINUESA, P. SCHLATTER, A. HANIFI, AND D. S. HENNINGSON, *Direct nu-*
423 *merical simulation of the flow around a wing section at moderate reynolds number*, Inter-
424 national Journal of Heat and Fluid Flow, (2016).
- 425 [8] G. K. KHOURY, P. SCHLATTER, A. NOORANI, P. F. FISCHER, G. BRETHOUWER, AND A. V.
426 JOHANSSON, *Direct numerical simulation of turbulent pipe flow at moderately high Reynolds*
427 *numbers*, Flow, Turbulence and Combustion, 91 (2013), pp. 475–495, [doi:10.1007/s10494-](https://doi.org/10.1007/s10494-013-9482-8)
428 [013-9482-8](https://doi.org/10.1007/s10494-013-9482-8), <http://dx.doi.org/10.1007/s10494-013-9482-8>.
- 429 [9] P. LINDSTROM AND M. ISENBURG, *Fast and efficient compression of floating-point data*, IEEE
430 Transactions on Visualization and Computer Graphics, 12 (2006), pp. 1245–1250.
- 431 [10] A. LODDOCH AND J. SCHMALZL, *Variable quality compression of fluid dynamical data sets using*
432 *a 3-d dct technique*, Geochemistry, Geophysics, Geosystems, 7 (2006).
- 433 [11] E. MERZARI, P. FISCHER, AND J. WALKER, *Large-scale simulation of rod bundles: Coherent*
434 *structure recognition and stability analysis*, in ASME/JSME/KSME 2015 Joint Fluids
435 Engineering Conference, American Society of Mechanical Engineers, 2015.
- 436 [12] K. R. RAO AND P. YIP, *Discrete Cosine Transform: Algorithms, Advantages, Applications*,
437 Academic Press Professional, Inc., San Diego, CA, USA, 1990.
- 438 [13] M. RATHINAM, LINDA, AND R. PETZOLD, *A new look at proper orthogonal decomposition*, SIAM
439 J. Numer. Anal, 41 (2003), pp. 1893–1925.
- 440 [14] N. SASAKI, K. SATO, T. ENDO, AND S. MATSUOKA, *Exploration of lossy compression for*
441 *application-level checkpoint/restart*, in Parallel and Distributed Processing Symposium
442 (IPDPS), 2015 IEEE International, IEEE, 2015, pp. 914–922.
- 443 [15] M. SCHANEN, O. MARIN, H. ZHANG, AND M. ANITescu, *Asynchronous two-level checkpointing*
444 *scheme for large-scale adjoints in the spectral-element solver Nek5000*, Procedia Computer
445 Science, 80 (2016), pp. 1147 – 1158, [doi:http://dx.doi.org/10.1016/j.procs.2016.05.444](https://doi.org/10.1016/j.procs.2016.05.444). In-
446 ternational Conference on Computational Science 2016, ICCS 2016, 6-8 June 2016, San
447 Diego, California, USA.
- 448 [16] J. SCHMALZL, *Using standard image compression algorithms to store data from com-*
449 *putational fluid dynamics*, Computers & Geosciences, 29 (2003), pp. 1021–1031,
450 [doi:http://dx.doi.org/10.1016/S0098-3004\(03\)00098-0](https://doi.org/10.1016/S0098-3004(03)00098-0).
- 451 [17] K. SCHNEIDER AND O. V. VASILYEV, *Wavelet methods in computational fluid dynamics**, Annual
452 Review of Fluid Mechanics, 42 (2010), pp. 473–503.
- 453 [18] A. TROTT, R. MOORHEAD, AND J. MCGINLEY, *Wavelets applied to lossless compression and*
454 *progressive transmission of floating point data in 3-d curvilinear grids*, in Visualization'96.
455 Proceedings., IEEE, 1996, pp. 385–388.
- 456 [19] F. VAN BELZEN AND S. WEILAND, *A tensor decomposition approach to data compression and*
457 *approximation of nd systems*, Multidimensional Systems and Signal Processing, 23 (2012),
458 pp. 209–236, [doi:10.1007/s11045-010-0144-x](https://doi.org/10.1007/s11045-010-0144-x).