Low-precision climate computing: Preserving information despite fewer bits

Milan Klöwer
Atmospheric, Oceanic and Planetary Physics
Jesus College
University of Oxford

supervised by

Prof. Tim Palmer, University of Oxford
Dr. Peter Düben, European Centre for Medium-Range Weather Forecasts

A thesis submitted for the degree Doctor of Philosophy

Oxford, September 2021

DON’T PANIC

Abstract

Who needs those bits anyway?

Contents

1 Introduction

2 General methods

2.1 Binary number formats

Integers

The simplest way to represent a real number in bits is the integer format. An n-bit signed integer starts with a sign bit followed by a sequence of integer bits, that are decoded as a sum of powers of two with exponents $0, 1, \ldots, n-2$. An positive integer x with signbit $b_0 = 0$ is therefore decoded in bits $b_1, \ldots, b_{n-2}$ as

$$x = \sum_{i=1}^{n-2} 2^{i-1} b_i \tag{2.1}$$

To avoid multiple representations of zero and to simplify hardware implementations, negative integers, with a sign bit (red) being 1, are decoded with two's complement interpretation (denoted with an underscore) by flipping all other bits and adding 1 [**?**]. For example in the 4-bit signed integer format (Int4), $1110_{\text{Int4}} = 1010\_ = -2$. The largest representable integer for a format with n bits is therefore $2^{n-1} - 1$ and the spacing between representable integers is always 1.

Fixed-point numbers

Fixed-point numbers extend the integer format with n_f fraction bits to n_i signed integer bits to decode an additional sum of powers of two with negative exponents $-1, -2, \ldots, -n_f$. A positive fixed-point number is

$$x = \sum_{i=1}^{n_i-2} 2^{i-1} b_i + \sum_{i=1}^{n_f} 2^{-i} b_{n_i-2+i} \tag{2.2}$$

Every additional fraction bit reduces the number of integer bits, for example Q6.10 is the 16-bit fixed-point format with 6 signed integer bits and 10 fraction bits.

Flexibility regarding the dynamic range can therefore be achieved with integer arithmetic if fixed-point numbers are used [**?**]. Unfortunately, we did not achieve convincing results with integer arithmetic for the applications in this study, as rescaling of the equations is desired to place many arithmetic calculations near the largest representable

number [**?**]. However, any result beyond will lead to disastrous results, as integer overflow usually returns a negative value following a wrap-around behaviour.

Floating-point numbers

The IEEE standard on floating-point arithmetic defines how floats encode a real number x in terms of a sign, and several exponent and significant bits

$$x = (-1)^{sign\ bit} \cdot 2^{e-bias} \cdot (1+f) \tag{2.3}$$

The exponent bits e are interpreted as unsigned integers, such that $e - bias$ converts them effectively to signed integers. The significant bits f_i define the significand as $f = \sum_{i=1}^{n_f} f_i 2^{-i}$ such that $(1+f)$ is in the bounds $[1, 2)$. An 8-bit float encodes a real number with a sign bit (red), $n_e = 3$ exponent bits (blue) and $n_f = 4$ fraction bits (black) as illustrated in the following example

$$3.14 \approx 0\,1001001_{\text{Float8}} = (-1)^{0} \cdot 2^{4-bias} \cdot (1 + 2^{-1} + 2^{-4}) = 3.125 \tag{2.4}$$

with $bias = 2^{n_e-1} - 1 = 3$. Exceptions to Eq. 2.3 occur for subnormal numbers, infinity (Inf) and Not-a-Number (NaN) when all exponent bits are either zero (subnormals) or one (Inf when f=0, or NaN else). 16-bit half-precision floatin-point numbers (Float16) have 5 exponent bits and 10 significant bits. A truncated version of the Float32 format (8 exponent bits, 23 significant bits) is BFloat16 with 8 exponent bits and 7 significant bits. Characteristics of various formats are summarised in Table 2.1. A format with more exponent bits has a wider dynamic range of representable numbers but lower precision, as fewer bits are available for the significant. All floating-point formats have a fixed number of significant bits. Consequently, they have a constant number of significant digits throughout their range of representable numbers (subnormals excluded), which is in contrast to posit numbers, which are introduced in the next section.

Logarithmic fixed-point numbers

Fixed-point numbers have a limited range and for the applications in this study an unsuitable distribution of decimal precision. However, logarithmic fixed-point numbers are similar to floating-point numbers. A n-bit logarithmic fixed-point number is defined as

$$x = (-1)^{sign\ bit} \cdot 2^{e} \tag{2.5}$$

Format	bits	exp bits	*minpos*	*maxpos*	ϵ	% NaR
Float64	64	11	$5.0 \cdot 10^{-324}$	$1.8 \cdot 10^{308}$	16.3	0.0
Float32	32	8	$1.0 \cdot 10^{-45}$	$3.4 \cdot 10^{38}$	7.6	0.4
Float16	16	5	$6.0 \cdot 10^{-8}$	65504	3.7	3.1
BFloat16	16	8	$9.2 \cdot 10^{-41}$	$3.4 \cdot 10^{38}$	2.8	0.4
Float8	8	3	$1.5 \cdot 10^{-2}$	15.5	1.9	12.5
Posit32	32	2	$7.5 \cdot 10^{-37}$	$7.5 \cdot 10^{37}$	8.8	0.0
Posit(16,1)	16	1	$3.7 \cdot 10^{-9}$	$3.7 \cdot 10^{9}$	4.3	0.0
Posit(16,2)	16	2	$1.4 \cdot 10^{-17}$	$1.4 \cdot 10^{17}$	4.0	0.0
Posit(8,0)	8	0	$1.5 \cdot 10^{-2}$	64	2.2	0.4
Int16	16	0	1	32767	0.8	0
Q6.10	16	0	$9.8 \cdot 10^{-4}$	32.0	3.7	0
LogFixPoint16	16	15	$5.4 \cdot 10^{-20}$	$1.8 \cdot 10^{19}$	3.2	0.0
Approx14	14	13	$5.4 \cdot 10^{-20}$	$9.1 \cdot 10^{18}$	2.6	0.8

Table 2.1 | Some characteristics of various number formats. *minpos* is the smallest representable positive number, *maxpos* the largest. The machine precision ϵ, is the decimal precision at 1. % NaR denotes the percentage of bit patterns that represent not a number (NaN), infinity or not a real (NaR).

where e is encoded as an $(n-1)$-bit fixed-point number (Eq. 2.2). Consequently, logarithmic fixed-point numbers are equally spaced in log-space and have a perfectly flat decimal precision throughout the dynamic range of representable numbers (Fig. **??**). We call the 16-bit logarithmic fixed-point numbers with 7 signed integer bits and 8 fraction bits LogFixPoint16. Approx14 is a proprietary number format developed by Singular Computing, which is essentially a 14-bit logarithmic fixed-point numbers with 7 signed integer bits and 6 fraction bits.

Logarithmic number formats have the advantage that no rounding error is applied for multiplication, as the addition of the exponents is exact with fixed-point numbers (as long as no under or overflow occurs). Hence, multiplication with LogFixPoint16 and Approx14 is not just exact but also fast, due to implementation as integer addition. Conversely, addition with logarithmic numbers is difficult. Adding two logarithmic numbers involves the computation of a logarithm, which, however, for low precision numbers can be implemented as a table look-up.

Both LogFixPoint16 and Approx14 come with a round-to-nearest rounding mode in log2-space. We consider $x_1 = 2^0 = 1$ and $x_2 = 2^1 = 2$ as two representable numbers as an example with x in between. With round-to-nearest (and tie to even) in linear-space all numbers x larger equal 1.5 are round up and others round down. With round-to-nearest in log2-space $2^{\frac{1}{2}} = \sqrt{2} = 1.414...$ is the log-midpoint as $\log_2(\sqrt{2}) = 0.5$. Consequently,

the numbers between $\sqrt{2}$ (inclusive) and 2 will be round up and only numbers between 1 and less than $\sqrt{2}$ will round down. Hence, the linear range of numbers that will be round up is larger than those that will round down. This rounding is biased as the expectation of rounded uniformly distributed values between 1 and 2 is not equal to the expectation without rounding. Let $\text{round}_{\log_2}(x)$ be the round-to-nearest function in log2-space and x be drawn N-times from a random uniform distribution $U(1,2)$, then

$$\frac{1}{N}\sum_i^N x_i = 1.5 \neq \frac{1}{N}\sum_i^N \text{round}_{\log_2}(x_i) = 3 - \sqrt{2} = 1.586... \tag{2.6}$$

We will investigate the effect of this round-to-nearest in log2-space in section **??**.

Posit numbers

Posit numbers arise from a projection of the real axis onto a circle (Fig. **??**), with only one bitpattern for zero and one for Not-a-Real (NaR, or complex infinity), which serves as a replacement for Not-a-Number (NaN). The circle is split into *regimes*, determined by a constant *useed*, which always marks the north-west on the posit circle (Fig. **??**b). Regimes are defined by $useed^{\pm 1}$, $useed^{\pm 2}$, $useed^{\pm 3}$, etc. To encode these regimes into bits, posit numbers extend floating-point arithmetic by introducing regime bits that are responsible for the dynamic range of representable numbers. Instead of having a fixed length, regime bits are defined as the sequence of identical bits after the sign bit, which are eventually terminated by an opposite bit. The flexible length allows the significand (or mantissa) to occupy more bits when less regime bits are needed, which is the case for numbers around one. A resulting higher precision around one is traded against a gradually lower precision for large or small numbers. A positive posit number p is decoded as [**???**] (negative posit numbers are converted first to their two's complement, see Eq. 2.9)

$$p = (-1)^{sign\ bit} \cdot useed^k \cdot 2^e \cdot (1+f) \tag{2.7}$$

where k is the number of regime bits. e is the integer represented by the exponent bits and f is the fraction which is encoded in the fraction (or significant) bits. The base $useed = 2^{2^{e_s}}$ is determined by the number of exponent bits e_s. More exponent bits increase - by increasing *useed* - the dynamic range of representable numbers for the cost of precision. The exponent bits themselves do not affect the dynamic range by changing the value of 2^e in Eq. 2.7. They fill gaps of powers of 2 spanned by $useed = 4, 16, 256, \ldots$

for $e_s = 1, 2, 3, ...$, and every posit number can be written as $p = \pm 2^n \cdot (1+f)$ with a given integer n [**??**]. We will use a notation where Posit(n,e_s) defines the posit numbers with n bits including e_s exponent bits. A posit example is provided in the Posit(8,1)-system (i.e. $useed = 4$)

$$57 \approx 01110111_{\text{Posit(8,1)}} = (-1)^0 \cdot 4^2 \cdot 2^1 \cdot (1 + 2^{-1} + 2^{-2}) = 56 \tag{2.8}$$

The sign bit is given in red, regime bits in orange, the terminating regime bit in brown, the exponent bit in blue and the fraction bits in black. The k-value is inferred from the number of regime bits, that are counted as negative for the bits being 0, and positive, but subtract 1, for the bits being 1. The exponent bits are interpreted as unsigned integer and the fraction bits follow the IEEE floating-point standard for significant bits. For negative numbers, i.e. the sign bit being 1, all other bits are first converted to their two's complement (**?**, denoted with an underscore subscript) by flipping all bits and adding 1,

$$\begin{aligned} -0.28 \approx 11011110_{\text{Posit(8,1)}} &= 10100010\_ \\ &= (-1)^1 \cdot 4^{-1} \cdot 2^0 \cdot (1 + 2^{-3}) = -0.28125. \end{aligned} \tag{2.9}$$

After the conversion to the two's complement, the bits are interpreted in the same way as in Eq. 2.8.

Posits also come with a no overflow/no underflow-rounding mode: Where floats overflow and return infinity when the exact result of an arithmetic operation is larger than the largest representable number (*maxpos*), posit arithmetic returns *maxpos* instead, and similarly for underflow where the smallest representable positive number (*minpos*) is returned. This is motivated as rounding to infinity returns a result that is infinitely less correct than *maxpos*, although often desired to indicate that an overflow occurred in the simulation. Instead, it is proposed to perform overflow-like checks on the software level to simplify exception handling on hardware [**?**]. Many functions are simplified for posits, as only two exceptions cases have to be handled, zero and NaR. Conversely, Float64 has more than 10^{15} bitpatterns reserved for NaN, but these only make up $< 0.05\%$ of all available bit patterns. The percentage of redundant bitpatterns for NaN increases for floats with fewer exponent bits (Table 2.1), and only poses a noticeable issue for Float16 and Float8.

The posit number framework also highly recommends *quires*, an additional register on hardware to store intermediate results. Dot-product operations are fused with quire arithmetic and can therefore be executed with a single rounding error, which is only

applied when converting back to posits. The quire concept could also be applied to floating-point arithmetic (fused multiply-add is available on some processors), but is technically difficult to implement on hardware for a general dot-product as the required registers would need to be much larger in size. For fair comparison we do not take quires into account in this study. The posit number format is explained in more detail in **?**. In order to use posits on a conventional CPU we developed for the Julia programming language [**?**] the posit emulator *SoftPosit.jl* [**?**], which is a wrapper for the C-based library SoftPosit [**?**]. The type-flexible programming paradigm, facilitated by the Julia language, is outlined in **??**.

Self-organizing numbers

The design of floating-point numbers in the 1970s and 1980s was motivated to meet several criteria: (i) hardware-friendly, i.e. the number format was designed to map easily from existing arithmetic circuits into bits; (ii) multi-purpose, i.e. initially declared as format for *scientific computations* it was supposed to be able to represent very large numbers $O(10^{-100})$ to $O(10^{-300})$ as well as very tiny numbers $O(10^{-100})$ to $O(10^{-300})$ with the same precision, to allow the use in many different fields of science (iii) error analysis-friendly, i.e. especially Float64 was designed to allow for very precise calculations, such that most scientists would not need to perform a numerical error analysis.

In the following we will relax these criteria and seek to find a number format that has the lowest rounding errors for a given application. We thereby ignore any hardware limitations, and create this number format purely on the basis of a software emulator. In fact, we end up designing it based on look-up tables. Arithmetic operations are with look-up tables not functions that calculate the result based on the inputs, but return the result from an array where the inputs determine the indices. This is in general only feasible for a small set of different inputs, as the underlying arrays have to be stored in memory. Look-up tables for 16-bit arithmetic, as considered here, are of a size of several GB which is unfeasible for current computing hardware. Look-up tables therefore are stored in RAM whereas a storage in much smaller low-level caches is necessary for speed. Every additional bit quadruples the size, such that look-up tables are only attractive for very low precision number formats or can be used for software emulators for up to 16 bit. We also do not aim to create a multi-purpose number format, but conversely a number format that is flexible enough to accommodate the precision requirements of a given algorithm as good as possible.

Motivated by the decimal precision analysed in the previous sections for floats and

posits, the new number format is supposed to represent numbers in a given application with most precision for the numbers that occur most frequently and with no bitpatterns for numbers that never occur. Consequently, this number format is supposed have bitpatterns that occur equally frequent. This concept aligns with maximising entropy, which will be discussed in the next section. In analogy to the unsupervised learning of self-organizing maps, which are mostly used in two or more dimensions, we call the new number format self-organizing numbers, or *sonums* in short. Note that the self-organization is here carried out on the real number axis, i.e. in one dimension.

We will make use of ideas introduced by the posit framework as introduced in section 2.1 as most redundant bitpatterns that occur in floats (± 0,$\pm\infty$, and a very large share of bitpatterns for NaNs, see Table 2.1) are removed for posits and only 2 bitpatterns for zero and NaR are retained as exceptions. The sonum circle is therefore designed in analogy to the posit circle (Fig. **??**), but will be populated differently except for zero and NaR, which are mapped to identical bitpatterns for both formats. As illustrated in Fig. **??** sonums retain the symmetry with respect to zero, such that there is a reversible map (the two's complement, **?**) between a number and its negation. However, sonums do not have a symmetry with respect to the multiplicative inverse as posits or floats have (note that for posits or floats this symmetry is only perfect when the significand is zero, otherwise rounding is applied, and excluding subnormal numbers). In the illustrated sonum circle it is therefore the idea to keep the real number value for s_1 to s_7 flexible and subject to training. In fact, sonums can be trained to replicate exactly the behaviour of posits with the same number of bits, but for any number of exponent bits. An n-bit sonum format has $m = 2^{n-1} - 1$ real number values that have to be defined. For 4-bit sonums $m = 7$, for 8-bit $m = 127$ and for 16-bit $m = 32767$. The size of the look-up tables scales with m^2 and is therefore quartic with the number of bits. Making use of commutativity for addition and multiplication as well as anti-commutativity for subtraction, the size reduces by a factor of two for those operations. The required size is therefore about 8KB per table for 8 bit and about 1GB per table for 16 bit. Division tables are twice that size. Sonums bear some similarity with type II unums, the predecessor of posits [**?**].

In the following we will describe the maximum entropy training (also called quantile quantization) for sonums and present ways to train sonums to minimize the decimal rounding error.

Given a data set D, regarded as j-element array of real numbers or a high precision approximation, we wish to find the sonum values s_i for $i \in 1, ..., 2^{n-1} - 1$ to maximise the entropy for an n-bit sonum format when representing the numbers in D. The infor-

mation entropy H (or Shannon entropy, **?**) is defined as

$$H = -\sum_i p_i \log_2(p_i) \tag{2.10}$$

where i is one possible state (here: bitpattern) with probability p_i, such that $\sum_i p_i = 1$ (Note that we define $p_i \log_2(p_i) = 0$ for $p_i = 0$). As we use the logarithm with base 2, the information entropy has units of bits. For a uniformly distributed probability, i.e. $p_i = \frac{1}{m}$ with m possible states the entropy is maximised to $n = \log_2(m)$ bits. In other words, the entropy is maximised when all states are equally likely and is zero for a discrete Dirac delta distribution.

We apply the concept of information entropy to the encoding of the standard uniform distribution $U(0, 1)$ between 0 and 1 with Float16, as an example (Fig. **??**). Analysing the bitpattern histogram, we observe no bitpatterns in Float16 occur that are associated with negative numbers or numbers larger than 1. Converting the frequency of occurrence of every bitpattern into a probability p_i, we calculate the entropy as 12 bit of theoretically 16 bit that are available in Float16. This can be roughly interpreted as follows: The sign bit is unused as only positive numbers occur. One bit is redundant as only values in $(0, 1)$ occur and none in $(1, \infty)$. Another two bits are unused due to the uneven bitpattern distribution between $(0, 1)$.

Maximising the entropy for the standard uniform distribution $U(0, 1)$ with sonums means that the values $s_i, i \in \{1, ..., m\}$ will be associated with numbers that are equidistantly distributed between 0 and 1. In theory therefore, $s_i = \frac{i}{m}$, which corresponds to the fixed-point numbers with a range from 0 to 1. In practice, one bitpattern is reserved for 0 and one for NaR, such that the entropy is not perfectly maximised. Furthermore, due to the symmetry with respect to zero, sonums have only 15 bit entropy as half the bitpatterns are reserved for all $-s_i$, which are not actually used. This poses only an issue in this artificial example, as many applications produce numbers that are symmetric with zero.

The generalization to arbitrary distributions, i.e. for any data set D, is therefore proposed as follows. In short, the array D is first sorted then split into m chunks of equal size. For each chunk the midpoint is found which is identified as the corresponding value for s_i. This can be written as an algorithm as shown in Fig. **??**. We use the arithmetic average between the minimum and maximum value (i.e. midpoint) in each chunk to satisfy the round-to-nearest rounding mode.

Once sonums are trained (i.e. the values s_i are set) the decimal precision can be calculated. An example is given in Fig. **??**, which shows how decimal precision of sonums

follow the distribution of data from the Lorenz 1996 model, which will be introduced and discussed in section **??**. After training the look-up tables have to be filled, which means that every arithmetic operation between all possible unique pairs of sonums is precomputed. This is for 8-bit sonums (Sonum8) fast, and even for 16-bit sonums (Sonum16) completed within a few minutes. Subsequently, sonums can be used as a number format like floats and posits, however, sonums will presumably only yield reliable results for the application they were trained on. We will investigate in section **??** how sonums compare to floats and posits in the Lorenz 1996 system.

In the previous section we discussed a maximum entropy approach for training sonums, however, there are other training approaches possible that we want to investigate. Given a data set $D_j, j \in \{1, ..., N\}$ of length N, and a maximum entropy-trained set of sonum values $s_i, i \in \{1, ..., m\}$ we may want to know whether the s_i actually minimize the average rounding error ARE

$$ARE = \frac{1}{N} \sum_{j}^{N} |D_j - \text{round}_s(D_j)| \tag{2.11}$$

with

$$\text{round}_s(x) = \underset{s_i \in s}{\arg\min} |x - s_i| \tag{2.12}$$

being the round-to-nearest rounding function for a given set of sonums s. Alternatively, one could require the average decimal error ADE to be minimized

$$ADE = \frac{1}{N} \sum_{j}^{N} |\log_{10}(\frac{D_j}{\text{round}_s(D_j)})| \tag{2.13}$$

which is equivalent to the (linear) average rounding error ARE when the logarithm with base 10 is applied to D_j beforehand and the rounding function is changed accordingly. Based on the framework around decimal precision presented in the previous section one may argue that it is more important to minimize ADE than ARE, but further analysis is needed to assess this with respect to a statistic like forecast error.

How to find s given D to minimize either ARE or ADE? We are therefore seeking a one-dimensional classification method that sorts all values in D into classes s_i. A classification is therefore a clustering and the two terms can be used interchangeably. Using the Jenks Natural Breaks Classification [**?**] is proposed and presented in the following in a modified version, that was found to be better suited in first tests for our applications.

The Jenks classification is usually applied on multi-modal distributions with a few classes. Here, we are attempting to find up to 32767 (for 16-bit sonums) classes from millions of data points or more, which complicates the convergence of this iterative algorithm. The original Jenks algorithm is a method to minimize in-class variance while maximizing the variance between classes.

The modified Jenks classification algorithm is presented in a simplified version.

(0) Convert all D_j to their absolute value $|D_j|$.

(1) Define m (arbitrary) initial classes, each with an upper break point b_i, such that all D_j within the previous break point b_{i-1} and the i-th break point b_i belong to class i. The m-th class break point is the maximum in D_j. We choose the maximum entropy method of the previous section as a initial classification.

Then loop over

(2) For each class i, calculate an (unnormalized) error norm E_i of values in that class with respect to the class midpoint. The error can be the total rounding error or the total decimal error for example.

(3) Calculate the sum of the error norms of all classes $\sum_i E_i$, which is important to assess convergence. Dividing by N yields the average rounding or decimal error, depending on which error norm was used.

(4) For each class i except the last one, compare the error E_i to the next class error E_{i+1}.

(4.1) If $E_i < E_{i+1}$: Increase b_i by r, which will be defined shortly. That means, shift the break point to the right on the real axis to make the i-th class bigger and the $(i+1)$-th class smaller. This will increase E_i and decrease E_{i+1}.

(4.2) Else: Decrease b_i by r.

Choosing an appropriate value for r, which is a flux of data point from one class to a neighbouring class, is difficult. We found that r should scale with the size of the donating class, such that a certain share of points should be passed on. Additionally, we decrease the flux r if the previous flux direction was opposite (4.2 was evaluated instead of 4.1 or vice versa), which is helpful to aid convergence. However, we increase the flux r if the previous flux direction was the same, which accelerates convergence.

In the next section we will test sonums against floats and posits. At the moment we will restrict this analysis to sonums which were trained with the maximum entropy

classification. Using sonums with minimised rounding error or decimal error is subject to further analysis to satisfy convergence.

Sonum16 will be tested against floats and posits in the one-level Lorenz 1996 model **??**, which is a simple chaotic weather model, described by the following equations

$$\frac{dX_k}{dt} = X_{k-1}(X_{k+1} - X_{k-2}) - X_k + F \tag{2.14}$$

with $k \in \{1, 2, ..., 36\}$. Periodic boundary conditions are applied, such that $X_0 = X_{36}$ and $X_{37} = X_1$. The first term on the right-hand side represents advection and the second is a relaxation term. The forcing is set to $F = 8$ and independent of space and time. Although the Lorenz 1996 model can be extended to a two or three-level version, such that levels can be interpreted as large, medium and small-scale variables, the model used here is the simple one-level version. The model is spun-up from rest $X_k = 0$ with a small perturbation in one variable. Scaling can be applied by multiplication with a constant s, such that $\hat{X}_k = sX_k$

$$\frac{d\hat{X}_k}{dt} = s^{-1}\hat{X}_{k-1}(\hat{X}_{k+1} - \hat{X}_{k-2}) - \hat{X}_k + F \tag{2.15}$$

which controls the range of numbers, occuring in the simulation. As similarly suggested for the Lorenz 63 model [**?**], we use $s = 10^{-1}$ (which is precomputed) for the simulation of Eq. 2.15 with posits to center the arithmetic operations around 1. This is beneficial for posit arithmetic as otherwise the prognostic variables X_k are $O(10)$.

A Hovmoeller diagram illustrates the chaotic dynamics simulated by the Lorenz 1996 model (Fig. **??**). The initial perturbation in X_1 is advected throughout the domain within the first time steps. After this first wake, the model's state becomes chaotic. Posit(16,1) represents well the dynamic, as shown in the comparison with Float64 for reference.

To quantify the forecast error, we run a set of 1000 forecasts per number format, starting from a random time step of a long control simulation. The simulation with Float64 is taken as reference truth. Float16 has an exponential error growth that starts much earlier than the error growth for Posit(16,1) (Fig. **??**). Both formats have on average an identical error growth rate. Posits clearly cause a reduced rounding error compared to floats at all lead times, making posits a better suited number format for the simulation of the Lorenz 1996 model.

We now use the long control simulation with Float64 to produce a dataset that contains all prognostic variables as well as the arithmetic results of all intermediate terms calculated for the tendencies. 16-bit sonums are trained with this dataset, such that

they can self-organize around the numbers that occur most frequently within the Lorenz 1996 model (Fig. **??**b). Consequently, Sonum16 has a slightly higher decimal precision compared to posits for the mode of the data distribution. A second mode is created for the tendencies, for which numbers of the order of 10^{-1} frequently occur. The decimal precision of Sonum16 drastically drops beyond the largest numbers $O(100)$ in the Lorenz 1996 model, as no bitpatterns have to be used to encode these real numbers.

After training, the sonum circle (Fig. **??**) is defined. All arithmetic operations are precomputed creating look-up tables for multiplication, addition and subtraction. No look-up table is created for division as the Lorenz 1996 equations (Eq. 2.15) are written division-free (s^{-1} is precomputed). We can now quantify the forecast error as for floats and posits, running a set of 1000 forecasts from the same initial conditions as used for floats and posits.

Sonum16 has a smaller forecast error compared to posits for the important lead times where the normalised RMSE exceeds 1% (Fig. **??**). Interestingly, although the error growth is much faster for the first time steps, it levels off afterwards and approaches the same error growth rate as for floats and posits once the normalised RMSE exceeds about 1%. This points towards a higher potential of Sonum16, when the cause of the initial rapid error growth is understood and circumvented with adjustments in the training method. As discussed in the previous section, sonums can be trained to minimize the average decimal rounding error, an aspect that requires further analysis to understand the optimal distribution of decimal precision for a given application. Nevertheless, sonums already provide perspectives towards an optimal number format for a given application. The main characteristics presumably are: (i) high precision for the most frequently occurring numbers, (ii) a tapered precision towards the smallest numbers and (iii) no redundant bitpatterns for very large and very small numbers that do not occur. Posits fulfill these criteria better than floats, which is likely the reason why they outperform floats in the applications presented here.

2.2 Rounding modes

Round to nearest

Stochastic rounding

The default rounding mode for floats and posits is round-to-nearest tie-to-even. In this rounding mode an exact result x is rounded to the nearest representable number x_i.

In case x is half-way between two representable numbers, the result will be tied to the even. A floating-point number x_i is considered to be even, if its significand ends in a zero bit. These special cases are therefore alternately round up or down, which removes a bias that otherwise persists (see Eq. 2.6 for an example of biased rounding). Let x_1 and x_2 be the closest two representable numbers to x and $x_1 \leq x < x_2$ then

$$\text{round}_{\text{nearest}}(x) = \begin{cases} x_1 & \text{if } x - x_1 < x_2 - x, \\ x_1 & \text{if } x - x_1 = x_2 - x \text{ and } x1 \text{ even}, \\ x_2 & \text{else}. \end{cases} \tag{2.16}$$

For stochastic rounding, rounding of x down to a representable number x_1 or up to x_2 occurs at probabilities that are proportional to the distance between x and x_1, x_2, respectively. Let δ be the distance between x_1, x_2, then

$$\text{round}_{\text{stoch}}(x) = \begin{cases} x_1 & \text{with probability} \quad 1 - \delta^{-1}(x - x_1) \\ x_2 & \text{with probability} \quad \delta^{-1}(x - x_1). \end{cases} \tag{2.17}$$

This behaviour is illustrated in Fig. **??**. In case that x is already identical with a representable number no rounding is applied and the chance to obtain another representable number is zero. For x being half way between two representable numbers, the chance of round up or round down is 50%. The introduced absolute rounding error for stochastic rounding is always at least as big as for round-to-nearest, and when low-probability round away from nearest occurs, it can be up to $\pm\delta$, whereas for round-to-nearest the error is bound by $\pm\frac{\delta}{2}$. Although the average absolute rounding error is therefore larger for stochastic rounding, the expected rounding error decreases towards zero for repeated roundings

$$\lim_{N\to\infty} \frac{1}{N} \sum_{i}^{N} \text{round}_{\text{stoch}}(x) = x \tag{2.18}$$

as follows by inserting Eq. 2.17. Stochastic rounding is therefore exact in expectation.

The stochastic rounding mode is implemented for Float16 and BFloat16. Software emulations of both number formats rely on conversion to Float32, such that the exact result (to the precision provided by Float32) is known before conversion back to 16 bit. Instead of calculating the probabilities given in Eq. 2.17, we add a stochastic perturbation $\xi \in [-\frac{\delta}{2}, \frac{\delta}{2}]$ to x before round-to-nearest. Let r be uniformly distributed in $[0, 1]$ then

Eq. 2.17 can then be rewritten as

$$\text{round}_{\text{stoch}}(x) = \begin{cases} \text{round}_{\text{nearest}}(x + \frac{\delta}{2}(r - \frac{x - x_1}{\delta})) & \text{if } x_1 = 2^n \text{ and } x - x_1 < \frac{\delta}{4} \\ \text{round}_{\text{nearest}}(x + \delta(r - \frac{1}{2})) & \text{else}\,. \end{cases} \tag{2.19}$$

The special case only occurs for x being within $\frac{\delta}{4}$ larger than a floating-point number $x_1 = 2^n$, that means with zero significand. In this case the distance from x_1 to the previous float is only $\frac{\delta}{2}$, which has to be accounted for.

Efficient bitwise implementations

2.3 Error norms

Mean, absolute and relative error

Decimal error and precision

The decimal precision is defined as [**??**]

$$\text{decimal precision} = -\log_{10} |\log_{10}(\frac{x_{\text{repr}}}{x_{\text{exact}}})| \tag{2.20}$$

where x_{exact} is the exact result of an arithmetic operation and x_{repr} is the representable number that x_{exact} is rounded to, given a specified rounding mode. For the common round-to-nearest rounding mode, the decimal precision approaches infinity when the exact result approaches the representable number and has a minimum in between two representable numbers. This minimum defines the *worst-case* decimal precision, i.e. the decimal precision when the rounding error is maximised. The worst-case decimal precision is the number of decimal places that are at least correct after rounding.

Fig. **??** compares the worst-case decimal precision for various 16 and 8-bit floats and posits, as well as 16-bit integers, the fixed-point format Q6.10 (6 integer bits, 10 fraction bits) and logarithmic fixed-point numbers LogFixPoint16 and Approx14. Float16 has a nearly constant decimal precision of almost 4 decimal places, which decreases for the subnormal numbers towards the smallest representable number *minpos*. 16-bit posits, on the other hand, show an increased decimal precision for numbers around 1 and a wider dynamic range, in exchange for less precision for numbers around 10^4 as well as 10^{-4}. The machine precision ϵ (in analogy to the machine error, also known as machine

epsilon), defined as half the distance between 1 and the next representable number, is given in terms of decimal precision and is summarised in Table 2.1 for the various formats. Due to the no overflow/no underflow-rounding mode, the decimal precision is slightly above zero outside the dynamic range.

The decimal precision of 16-bit integers is negative infinity for any number below 0.5 (round to 0) and maximised for the largest representable integer $2^{15} - 1 = 32767$. Similar conclusions hold for the fixed-point format Q6.10, as the decimal precision is shifted towards smaller numbers by a factor of $\frac{1}{2}$ for each additional fraction bit.

2.4 Type-flexibility through code composability

A type-flexible programming paradigm

Julia's programming paradigms of *multiple-dispatch* and *type-stability* facilitate the use of arbitrary number formats without the need to rewrite an algorithm, while compiling functions for specific types [**?**]. As this is an essential feature of Julia and extensively made use of here, we briefly outline the benefits of Julia by computing the harmonic sum $\sum_{i=1}^{\infty} \frac{1}{i}$ with various number types as an example. Analytically the harmonic sum diverges, but with finite precision arithmetic several issues arise. With an increasing sum the precision is eventually lower than required to represent the increment of the next summand. The integer i as well as its inverse $\frac{1}{i}$ have to be representable in a given number format, and are also subject to rounding errors.

Executing the function `harmonic_sum` for the first time with a type `T` as the first argument, triggers Julia's *just-in-time* compiler (Fig. **??**). The function is type-stable, as the types of all variables are declared and therefore known to the compiler. At the same time Julia allows for type-flexibility, as its *multiple-dispatch* means that calling `harmonic_sum` with another type `T2` will result in a separately compiled function for `T2`. We can therefore compute the harmonic sum with arbitrary number types, as long as the zero-element `zero(T)`; the one-element `one(T)`; addition; division; conversion from integer and conversion to float are defined for `T`.

The harmonic sum converges after 513 elements when using Float16 (Fig. **??**). The precision of BFloat16 is so low that the sum already converges after 65 elements, as the addition of the next term $1/66$ is rounded back to 5.0625. We identify the addition of small terms to prognostic variables of size $\mathcal{O}(1)$ as one of the major challenges with low precision arithmetic, which is discussed in more detail in section **??**. Using Posit(16,1), the sum only converges after 1024 terms, due to the higher decimal precision of posits

between 1 and 10.

Analysis number formats

2.5 Information theory

Entropy

Mutual information

Preserved information

3 Information-preserving compression

Contributions This chapter is largely based on the following publication[1]

M Klöwer, M Razinger, JJ Dominguez, PD Düben and TN Palmer, 2021. *Compressing atmospheric data into its real information content*, **Nature Computational Science**, in review.

3.1 Introduction

Many supercomputing centres in the world perform operational weather and climate simulations several times per day1. The European Centre for Medium-Range Weather Forecasts (ECMWF) produces 230TB of data on a typical day and most of the data is stored on magnetic tapes in its archive. The data production is predicted to quadruple within the next decade due to an increased spatial resolution of the forecast model2–4. Initiatives towards operational predictions with global storm-resolving simulations, such as Destination Earth5 or DYAMOND6, with a grid spacing of a couple of kilometres will further increase data volume. This data describes physical and chemical variables of atmosphere, ocean and land in up to 6 dimensions: three in space, time, forecast lead time, and the ensemble dimension. The latter results from calculating an ensemble of forecasts to estimate the uncertainty of predictions7,8. Most geophysical and geochemical variables are highly correlated in all of those dimensions, a property that is rarely exploited for climate data compression, although multidimensional compressors are being developed9–12.

Floating-point numbers are the standard to represent real numbers in binary form. 64-bit double precision floating-point numbers (Float64) consist of a sign bit, 11 exponent bits representing a power of two, and 52 mantissa bits allowing for 16 decimal places of precision across more than 600 orders of magnitude13. Most weather and climate models are based on Float64 arithmetics, which has been questioned as the transition to 32-bit single precision floats (Float32) does not necessarily decrease the quality of forecasts14,15. Many bits in Float32 only contain a limited amount of information as even 16-bit arithmetic has been shown to be sufficient for parts of weather and climate applications16–19. The information, as defined by Shannon information

[1] with the following author contributions. Conceptualization: MK, MR, JJD. Data curation: MR, JJD, MK. Formal Analysis: MK. Methodology: MK. Visualization: MK. Writing – original draft: MK. Writing – review & editing: MK, PDD, MR, JJD, TNP. The contributions of Miha, Juanjo, Peter, and Tim are highly appreciated.

theory20,21, for simple chaotic dynamical systems is often zero for many of the 32 bits in Float3222. This supports the general concept of low-precision climate modelling for calculations and data storage, as, at least in theory, many rounding errors are entirely masked by other uncertainties in the chaotic climate system23,24.

The bitwise information content has been formulated for predictability in dynamical systems22. It quantifies how much individual bits in the floating-point representation contribute to the information necessary to predict the state of a chaotic system at a later point in time. This technique has been used to optimise the simulation of simple chaotic systems on inexact hardware to reduce the precision as much as possible. Here, we extend the bitwise information content to distinguish between bits with real and false information in data and to quantify the preserved real information in data compression.

Data compression for floating-point numbers often poses a trade-off in size, precision and speed25–27: Higher compression factors for smaller file sizes can be achieved with lossy compression, which reduces the precision and introduces rounding errors. Additionally, higher compression requires more sophisticated compression algorithms, which can decrease compression and/or decompression speed. A reduction in precision is not necessarily a loss of real information, as occurring rounding errors are relative to a reference that itself comes with uncertainty. Here, we calculate the bitwise real information content20–22 of atmospheric data to discard bits that contain no information28,29 and only compress the real information content. Combined with modern compression algorithms10,30–32 the multi-dimensional correlation of climate data is exploited for higher compression efficiency33,34.

3.2 Methods

Data

CAMS data is analysed for one time step on 01/12/2019 12:00 UTC, bilinearly regridded onto a regular 0.4°x0.4° longitude-latitude grid using climate data operators (cdo) v1.9. All 137 vertical model levels are included. Furthermore, global fields of temperature from ECMWF's ensemble prediction system with 91 vertical levels are used from the first 25 members of a 50-member 15-day ensemble forecast starting on 24 Sept 2020 00:00 UTC. Bilinear regridding onto a regular 0.2°x0.2° longitude-latitude grid, similar as for the CAMS data, is applied. All compression methods here include the conversion from Float64 to Float32.

Only longitude-latitude grids are considered in this paper. However, the methodol-

ogy can be applied to other grids too. For example, ECMWF's octahedral grid collapses the two horizontal dimensions into a single horizontal dimension which circles on latitude bands around the globe starting at the South Pole till reaching the North Pole55. Fewer grid points of the octahedral grid reduce the size, but the correlation in latitudinal direction cannot be exploited.

Bitpattern entropy

An n-bit number format has 2^n bitpatterns available to encode a real number. For most data arrays, not all bitpatterns are used at uniform probability. The bitpattern entropy is the Shannon information entropy H, in units of bits, calculated from the probability of each bitpattern

$$H = -\sum_{i=1}^{2^n} p_i \log_2(p_i) \tag{3.1}$$

The bitpattern entropy is $H \leq n$ and maximised to n bits for a uniform distribution. The free entropy is the difference $n - H$.

Grid definitions

The compression methods described here are applied to gridded binary data.Data on structured grids can be represented as a tensor, such that for two dimensions that data can be arranged in a matrix A with elements a_{ij} and i, j indices. Adjacent elements in A, e.g. a_{ij} and $a_{i+1,j}$, are also adjacent grid points. Every element a_{ij} is a floating-point number, or in general a number represented in any binary format. The n bits in a_{ij} are described as bit positions, including sign, exponent and mantissa bits. In the following we will consider sequences of bits that arise from incrementing the indices i or j while holding the bit position fixed. For example, the sequence of bits consisting of the first mantissa bit in a_{ij}, then the first mantissa bit in $a_{i+1,j}$, and so on. We may refer to these bits as bits from adjacent grid points. Every bit position in elements of A is itself a matrix, e.g. the matrix of sign bits across all grid points.

Real information content

The Shannon information entropy20 H in units of bits takes for a bitstream $b = b_1 b_2 ... b_k ... b_l$, i.e. a sequence of bits of length l, the form

$$H = -p_0 \log_2(p_0) - p_1 \log_2(p_1) \tag{3.2}$$

with p_0, p_1 being the probability of a bit b_k in b being 0 or 1. The entropy is maximised to 1 bit for equal probabilities $p_0 = p_1 = \frac{1}{2}$ in b. We derive the mutual information40–42 of two bitstreams $r = r_1 r_2 ... r_k ... r_l$ and $s = s_1 s_2 ... s_k ... s_l$. The mutual information is defined via the joint probability mass function p_{rs}, which here takes the form of a 2x2 matrix

$$p_{rs} = \begin{pmatrix} p_{00} & p_{01} \\ p_{10} & p_{11} \end{pmatrix} \tag{3.3}$$

with p_{ij} being the probability that the bits are in the state $r_k = i$ and $s_k = j$ simultaneously and $p_{00} + p_{01} + p_{10} + p_{11} = 1$. The marginal probabilities follow as column or row-wise additions in p_{rs}, e.g. the probability that $r_k = 0$ is $p_{r=0} = p_{00} + p_{01}$. The mutual information $M(r, s)$ of the two bitstreams r, s is then

$$M(r, s) = \sum_{r=0}^{1} \sum_{s=0}^{1} p_{rs} \log_2 \left(\frac{p_{rs}}{p_{r=r} p_{s=s}} \right) \tag{3.4}$$

We now consider the two bitstreams r, s being the preceding and succeeding bits (for example in space or time) in a single bitstream b, i.e. $r = b_1 b_2 ... b_{l-1}$ and $s = b_2 b_3 ... b_l$. As explained above, this can for example be the bitstream of all first mantissa bits in the gridded data. Considering r, s as the preceding and succeeding bits is equivalent to the bitwise mutual information in adjacent grid points. The (unconditional) entropy is then effectively $H = H(r) = H(s)$ as in Eq. 3.2 and for l being very large. The conditional entropies H_0, H_1 are conditioned on the state of the preceding bit b_{k-1} being 0 or 1, respectively

$$H_0 = -p_{00} \log_2(p_{00}) - p_{01} \log_2(p_0 1) \tag{3.5}$$

$$H_1 = -p_{10} \log_2(p_{10}) - p_{11} \log_2(p_1 1) \tag{3.6}$$

The conditional entropy is maximised to 1 bit for bitstreams where the probability of a bit being 0 or 1 does not depend on the state of the preceding bit, which is here defined as *false information*. With the conditional and unconditional entropies and p_0, p_1 as in

Eq. 3.2 the mutual information M of succeeding bits can be written as

$$I = H - p_0 H_0 - p_1 H_1 \tag{3.7}$$

which is the real information content I. This definition is similar to **?** but avoids an additional assumption of an uncertainty measure. Their formulation similarly uses the state of bits as predictors but assesses the conditional probability density function (pdf) of a dynamical system as predictands. The binwidth of the pdf is chosen to represent the uncertainty in the system, which the bitwise real information strongly depends on. The formulation here avoids such an additional assumption of uncertainty as bits are used as both predictors and predictands in the conditional entropy. Consequently, the uncertainty is obtained from the data itself solely based on the mutual information between bits in adjacent grid points.

Eq. 3.7 defines the real information as the entropy minus the false information. For bitstreams with either $p_0 = 1$ or $p_1 = 1$, i.e. all bits are either 0 or 1, the entropies are zero $H = H_0 = H_1 = 0$ and we may refer to the bits in the bitstream as being unused. In the case where $H > p_0 H_0 + p_1 H_1$, the preceding bit is a predictor for the succeeding bit which means that the bitstream contains real information ($I > 0$).

The multidimensional real information content

The real information content I_m for an m-dimensional array A is the sum of the real information along the m dimensions. Let b_j be a bitstream obtained by unravelling a given bitposition in A along its j-th dimension. Although the unconditional entropy H is unchanged along the m-dimensions, the conditional entropies H_0, H_1 change as the preceding and succeeding bit is found in another dimension, e.g. b_2 is obtained by reordering b_1. $H_0(b_j)$ and $H_1(b_j)$ are the respective conditional entropies calculated from bitstream b_j. Normalization by $\frac{1}{m}$ is applied to I_m such that the maximum information is 1 bit in I_m^*

$$I_m^* = H - \frac{p_0}{m} \sum_{j=1}^{m} H_0(b_j) - \frac{p_1}{m} \sum_{j=1}^{m} H_1(b_j) \tag{3.8}$$

Due to the presence of periodic boundary conditions for longitude a succeeding bit might be found across the bounds of A. This simplifies the calculation as the bitstreams are obtained from permuting the dimensions of and subsequent unravelling into a vector.

Preserved information

We define the preserved information $P(r, s)$ in a bitstream s when approximating r (e.g. after a lossy compression) via the symmetric normalised mutual information

$$R(r, s) = \frac{2M(r, s)}{H(r) + H(s)} \tag{3.9}$$

R is the redundancy of information of r in s. The preserved information P in units of bits is then the redundancy-weighted real information I in r

$$P(r, s) = R(r, s)I(r) \tag{3.10}$$

The information loss L is $1 - P$ and represents the unpreserved information of r in s. In most cases we are interested in the preserved information of an array $X = (x_1, x_2, ...x_q, ...x_n)$ of bitstreams x when approximated by a previously compressed array $Y = (y_1, y_2, ...y_q, ...y_n)$. For an array A of floats with $n = 32$ bit, for example, x_1 is the bitstream of all sign bits unravelled along a given dimension (e.g. longitudes) and x_{32} is the bitstream of the last mantissa bits. The redundancy $R(X, Y)$ and the real information $I(X)$ are then calculated for each bit position q individually, and the fraction of preserved information P is the redundancy-weighted mean of the real information in X

$$P(X, Y) = \frac{\sum_{q=1}^{n} R(x_q, y_q)I(x_q)}{\sum_{q=1}^{n} I(x_q)} \tag{3.11}$$

The quantity $\sum_{q=1}^{n} I(x_q)$ is the total information in X and therefore also in A. The redundancy is $R = 1$ for bits that are unchanged during rounding and $R = 0$ for bits that are rounded to zero. The preserved information with bitshave or halfshave28,29 (i.e. replacing mantissa bits without real information with either 00...00 or 10...00, respectively) is therefore equivalent to truncating the bitwise real information for the (half)shaved bits. For round-to-nearest, however, the carry bit depends on the state of bits across several bit positions. To account for interdependency of bit positions the mutual information has to be extended to include more bit positions in the joint probability p_{rs}, which will then be a $m \times 2$ matrix. For computational simplicity, we truncate the real information as the rounding errors of round-to-nearest and halfshave are equivalent.

Significance of real information

In the analysis of real information it is important to distinguish between bits with very little but significant information and those with information that is insignificantly different from zero. While the former have to be retained, the latter should be discarded to increase compressibility. A significance test for real information is therefore presented.

For an entirely independent and approximately equal occurrence of bits in a bitstream of length l, the probabilities p_0, p_1 of a bit being 0 or 1 approach $p_0 \approx p_1 \approx \frac{1}{2}$, but they are in practice not equal for $l < \infty$. Consequently, the entropy is smaller than 1, but only insignificantly. The probability p_1 of successes in the binomial distribution (with parameter $\frac{1}{2}$) with l trials (using the normal approximation for large l) is

$$p_1 = \frac{1}{2} + \frac{z}{2\sqrt{2}} \tag{3.12}$$

where z is the $1 - \frac{1}{2}(1 - c)$ quantile at confidence level c of the standard normal distribution. For $c = 0.99$ corresponding to a 99%-confidence level which is used as default here, $z = 2.58$ and for $l = 5.5 \cdot 10^7$ (the size of a 3D array from CAMS) a probability $\frac{1}{2} \leq p \leq p_1 = 0.5002$ is considered insignificantly different from equal occurrence $p_0 = p_1$. The associated free entropy H_f in units of bits follows as

$$H_f = 1 - p_1 \log_2(p_1) - (1 - p_1) \log_2(1 - p_1) \tag{3.13}$$

We consider real information below H_f as insignificantly different from 0 and set the real information $I = 0$.

Dependency of the bitwise real information on correlation

The real information as defined here depends on the mutual information of bits in adjacent grid points. Higher autocorrelation in data (meaning a higher correlation between adjacent grid points) increases the mutual information in the mantissa bits. With higher correlation the adjacent grid values are closer, increasing the statistical dependence of mantissa bits that would otherwise be independent at lower correlation. Consequently, the real bitwise information content is increased and more mantissa bits have to be retained to preserve 99% of real information (Fig. 3.2a and b).

The increasing number of retained mantissa bits with higher autocorrelation in data will decrease compression factors, as it is easier to compress bits that are rounded to zero. However, a higher correlation also increases the redundancy in bits of adjacent

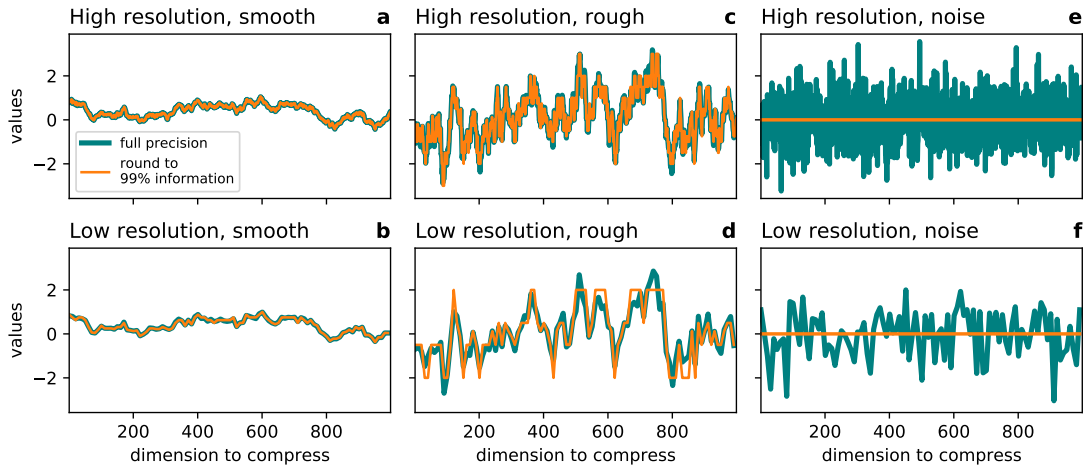

Figure 3.1 | Resolution and smoothness dependence of the information-preserving compression. **a,b** Highly autocorrelated data (1st order auto-regressive process with correlation $r = 0.999$) will have many mantissa bits preserved, at high and low resolution. **c,d** Many mantissa bits in data with less autocorrelation ($r = 0.95$) will be independent at low resolution and therefore rounded to zero. **e,f** All bits in random data ($r = 0$) drawn from a standard normal distribution are fully independent so that removing the false information rounds this data to zero. Low resolution data (**b,d,f**) is obtained from high resolution (**a,c,e**) by subsampling every 10th data point.

grid points, which favours a more efficient lossless compression. These two effects counteract and compression factors only increase piecewise over a small range of correlations while the retained mantissa bits are constant (Fig. 3.2c and d). Once an additional mantissa bit has to be retained to preserve 99% of real information, compression factors jump back down again, resulting in a sawtooth wave. Over a wide range of typical correlation coefficients (0.5 - 0.9999) the compression factors are otherwise constant and no higher compressibility is found with increased correlation.

The compression factors can, however, depend on the range of values represented in binary: A shift in the mean to have positive or negative values only means that the sign bit is unused, which increases compression factors (compare Fig. 3.2a with b), despite identical correlation coefficients. Although the correlation is invariant under multiplicative scaling and addition, the bitwise information changes under addition: When the range of values in data fits into a power of two its real information is shifted across bit positions into the mantissa bits, such that the exponent bits are unused. This can be observed for atmospheric temperatures stored in Kelvin (within 200-330K) where only the last exponent bit and mantissa bits contain information (Fig. **??**). Using Celsius instead shifts information from the mantissa bits into the exponent and sign bits.

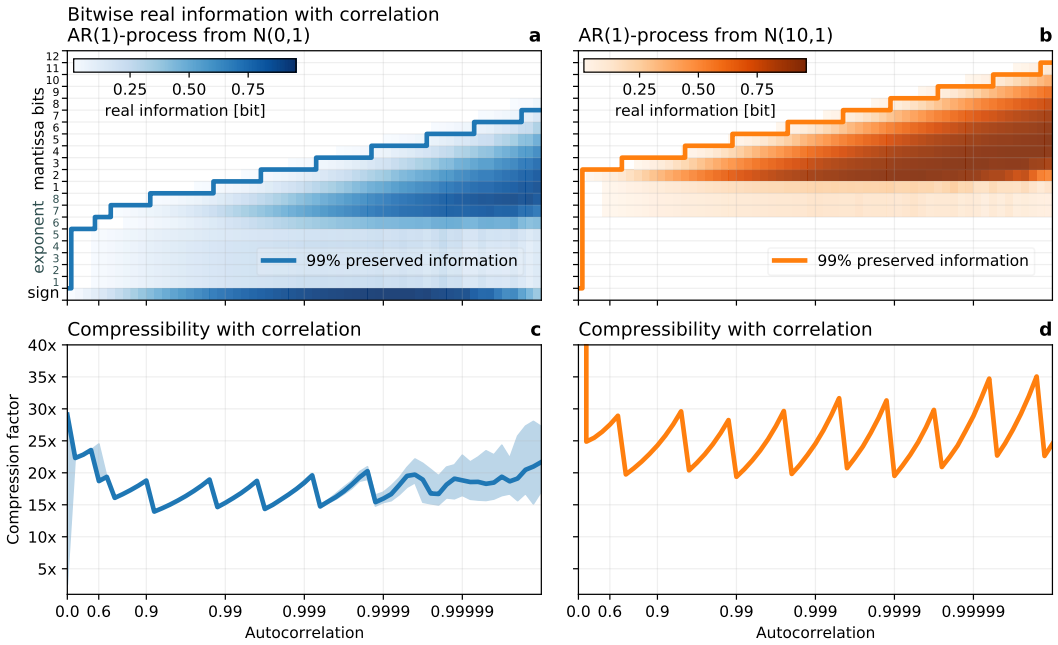

Figure 3.2 | Dependency of the bitwise real information and compressibility on correlation. a The bitwise real information content of a first-order autoregressive process (AR(1) with Gaussian distribution N(0,1), i.e. with zero mean and unit variance) with varying lag-1 autocorrelation. The bits that have to be retained to preserve 99% of information are enclosed with a solid line. b as **a** but the AR(1) process follows a Gaussian distribution with a mean of 10. **c,d** Compression factors for **a,b** when preserving 99% of information. Shading denotes the interdecile range.

Limitations of the information-preserving compression

The definition of real information presented here is based on the mutual information in adjacent grid points. We therefore assume a spatial and temporal coherence of data that will come with some autocorrelation. For vanishing autocorrelation in the data the real information content will drop to zero, as the mutual information between bits in adjacent but independent grid points approaches zero. In this case the entire dataset is identified as false information and consequently rounded to zero. In practice, this only occurs with data having autocorrelation coefficients of less than 0.2 (Fig. 3.2). If there is valuable scientific information in such seemingly random data, then the underlying assumption that real information is identified by the mutual information in adjacent grid points does not hold.

Issues with the bitwise real information content can arise in data that was previously subject to lossy compression: Linear or logarithmic quantization, for example, rounds data in linear or logarithmic space, respectively, which is not equivalent to binary rounding in the floating-point format. Consequently, such a quantization will generally introduce non-zero bits in the mantissa of floats when decompressed. These bits can have some statistical dependence, appearing as artificial information induced by the quantization. Such artificial information can be observed as a small background information (i.e. the bitwise real information is always significantly different from 0) or a reemerging information in the last mantissa bits. In this case, the information distribution across bit positions deviates clearly from the typical, where the information drops monotonically in the mantissa bits and becomes insignificantly different from 0 (see the examples in Fig. 3.10, 3.2, 3.4 or **??**). A solution to this quantization-induced artificial information is to apply the bitwise real information analysis not on the decompressed floats but on the rounded integers representing the compressed data in linear or logarithmic quantization. The bitwise real information content as defined here is independent of the binary format and therefore can be applied to floats as well as the unsigned integers from quantization. Note that this issue does not arise when rounding is applied in the same encoding which is also used to analyse the real information content. In our case, using the floating-point representation for the rounding (i.e. removing the false information) the rounded mantissa bits are always 0 which also means zero entropy when analysing the information. Applying the rounding for floats repeatedly will have no effect beyond the first application (idempotence).

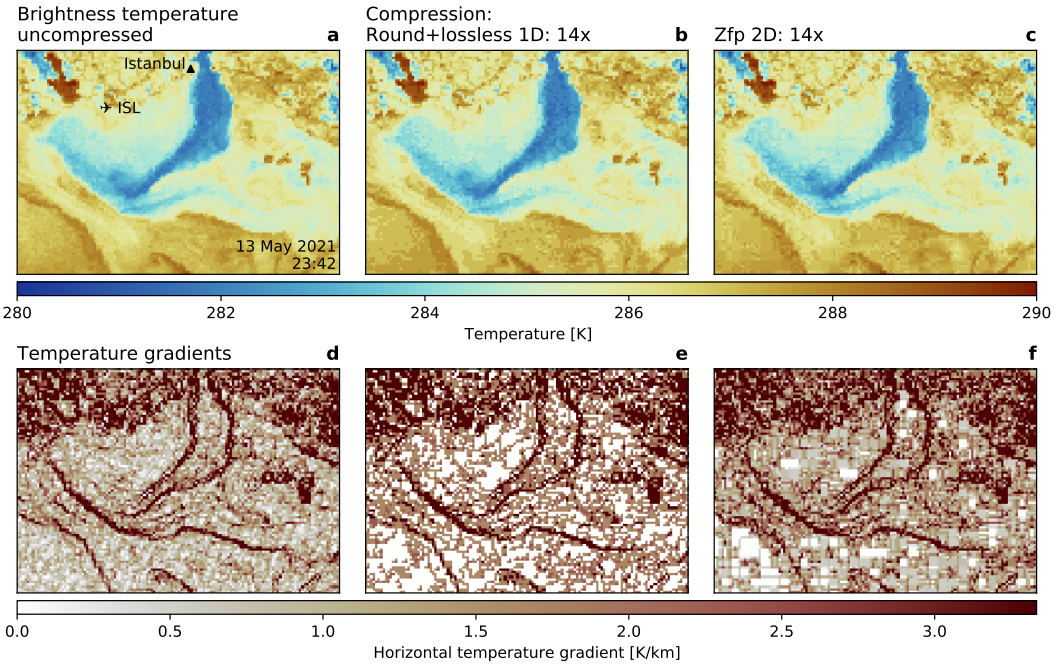

Figure 3.3 | Preservation of gradients during compression. Compressing the brightness temperature of Fig. 3.14 (VIIRS sensor aboard the satellite Suomi NPP) south of Istanbul where the Black Sea outflows into the Marmara Sea. Oceanic fronts with strong horizontal gradients in sea surface temperature are visible. **a** Brightness temperature uncompressed. **b** as **a** but compressed using round+lossless preserving 99% of real information. **c** as **a** but using Zfp compression in the two horizontal dimensions. **d** Horizontal temperature gradient uncompressed highlighting the oceanic fronts from **a**. **e** as **a** but the horizontal gradient is calculated from the round+lossless compressed dataset as shown in **b**. **f** as **e** but using Zfp compression as shown in **c**. The coarseness of the visualisation represents the resolution of the data. Istanbul (Hagia Sophia) and Atatürk Airport (ISL) are marked for orientation.

Preservation of gradients

The preservation of gradients and other higher-order derivatives in data is a challenging aspect of compression. Removing false information in data via rounding can result in identical values in adjacent grid points. Even if these values were not identical before rounding, they may not be significantly different from each other in the sense of real and false information. In this case, a previously weak but non-zero gradient will be rounded to zero.

This can be illustrated in the example of analysing oceanic fronts obtained from satellite measurements of sea surface temperatures (Fig. **??**). Identified by large horizontal gradients in temperature, the location and strength of oceanic fronts is well preserved using compressed data. However, areas of very weak gradients can largely vanish with round+lossless (Fig. **??**e). In this case the temperature in adjacent grid points is insignificantly different from each other and therefore the gradient zero after the removal of false information. Weak gradients are better preserved with Zfp compression at similar compression factors, but its block structure becomes visible (Fig. **??**f).

Linear and logarithmic quantization

The n-bit linear quantization compression for each element a in an array A is

$$\tilde{a} = \text{round}\left(2^{n-1}\frac{a - \min(A)}{\max(A) - \min(A)}\right) \tag{3.14}$$

with round a function that rounds to the nearest integer in $0, \ldots, 2^{n-1}$. Consequently, every compressed element $\tilde{a}$ can be stored with n bits. The n-bit logarithmic quantization compression for every element $a \geq 0$ in A is

$$\tilde{a} = \begin{cases} 0 & \text{if } a = 0, \\ \text{round}\left(c + \Delta^{-1}\log(a)\right) + 1 & \text{else.} \end{cases} \tag{3.15}$$

which reserves the bit pattern zero to encode 0. The logarithmic spacing is

$$\Delta = \frac{\log(\max(A)) - \log(\min^{+}(A))}{2^n - 2} \tag{3.16}$$

The constant $c = 1/2 - \Delta^{-1}\log(\min^{+}(A)(\exp(\Delta)+1)/2)$ is chosen to implement round-to-nearest in linear space instead of in logarithmic space, for which $c = -\Delta^{-1}\log(\min^{+}(A))$. The function $\min^{+}(A)$ is the minimum of all positive elements in A.

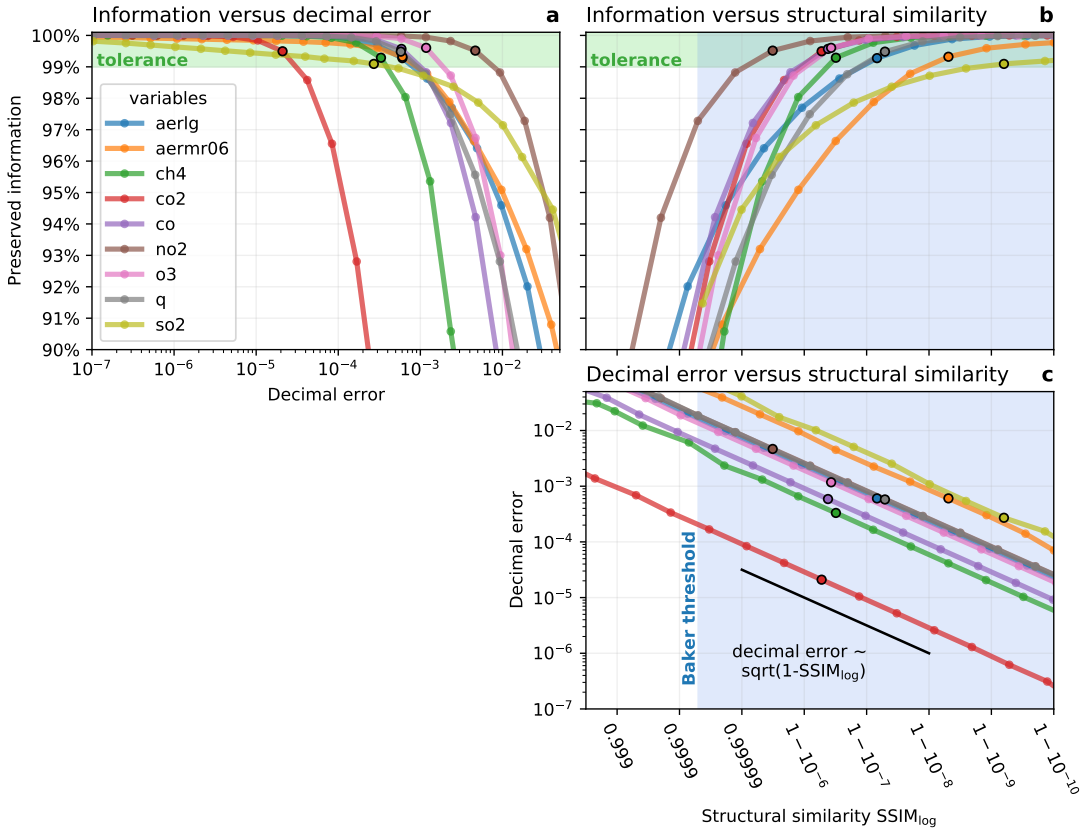

Figure 3.4 | The relationships between preserved information, decimal error and structural similarity for rounding within the information-preserving compression. **a** The last 1% of information tends to be distributed across many mantissa bits such that a trade-off arises where a large increase in compressibility is achieved for a small tolerance in information loss. The preserved information is presented as a function of the decimal error, which itself increases exponentially for every additional bit (small circles) that is discarded due to rounding. Denoted circles present the number of mantissa bits that have to be retained during compression to preserve at least 99% of information. **b** The preserved information increases as a function of the structural similarity (SSIM)56. The proposed threshold for climate data of SSIM=0.99995 by Baker et al. 2019 is shaded52. All variables are very close or above the Baker threshold when preserving 99% of information. **c** The decimal error is proportional to the square root of the structural dissimilarity 1-SSIM for binary rounding within the information-preserving compression

Structural similarity

A metric to assess the quality of lossy compression in image processing is the structural similarity index measure (SSIM)57. For images it is based on comparisons of luminance, contrast and structure. For floating-point arrays the luminance contributions to SSIM can be interpreted as the preservation of the mean; the contrast compares the variances and the structure compares the correlation. The SSIM of two arrays A, B of same size is defined as

$$\mathrm{SSIM}(A, B) = \frac{(2\mu_A\mu_B + c_1)(2\sigma_{AB} + c_2)}{(\mu_A^2 + \mu_B^2 + c_1)(\sigma_A^2 + \sigma_B^2 + c_2)} \tag{3.17}$$

With μ_A, μ_B the respective means, σ_A^2, σ_B^2 the respective variances and σ_{AB} the covariance. $c_1 = (k_1 L)^2$ and $c_2 = (k_2 L)^2$ are introduced to increase stability with a small denominator and $k_1 = 0.01$, $k_2 = 0.03$. The dynamic range is $L = \max(\max(A), \max(B)) - \min(\min(A), \min(B))$. The SSIM is a value in $[0, 1]$ where the best possible similarity $\mathrm{SSIM} = 1$ is only achieved for identical arrays.

For rounded floating-point arrays the decimal error is proportional to the square root of the dissimilarity $1 - \mathrm{SSIM}$ (Fig. 3.4c). The SSIM in this case is approximately equal to the correlation, as round-to-nearest is bias-free (i.e. $\mu_A \approx \mu_B$) and as the rounding error is typically much smaller than the standard deviation of the data (i.e. $\sigma_A \approx \sigma_B$). Here, we use the logarithmic SSIM, $\mathrm{SSIM}_{\log}(A, B) = \mathrm{SSIM}(\log(A), \log(B))$, which is the SSIM applied to log-preprocessed data (the logarithm is applied element-wise). The usage of $\mathrm{SSIM}_{\log}$ is motivated due to the rather logarithmic data distribution for most variables (Fig. 3.7), but similar results are obtained for SSIM. The proportionality to the decimal error is unchanged when using $\mathrm{SSIM}_{\log}$.

? propose the SSIM as a quality metric for lossy compression of climate data52. While for image processing SSIM > 0.98 is considered good quality, **?** suggest a higher threshold of SSIM = 0.99995 for climate data compression. The preserved information as defined here can be used as a compression quality metric similar to the SSIM. When preserving 99% of real information the $\mathrm{SSIM}_{\log}$ is also above the Baker threshold (Fig. 3.4b), reassuring that our threshold of 99% preserved real information is reasonable. In general, the preserved information is a monotonic function of the structural similarity SSIM (or $\mathrm{SSIM}_{\log}$) for rounded floating-point arrays, further supporting the usage of preserved information as a metric for data compression.

Lossless compression

We use Zstandard as a default lossless algorithm for the round+lossless method. Zstandard is a modern compression algorithm that combines many techniques to form a single compressor with tunable 22 compression levels that allow large trade-offs between compression speed and factors47,49. Here, we use compression level 10, as it presents a reasonable compromise between speed and size. Zstandard outperforms other tested algorithms (deflate, LZ4, LZ4HC and Blosc) in our applications and is also found to be among the best in the lzbench compression benchmark47 and other studies have focused on comparisons44. Lossless compressors are often combined with reversible transformations that preprocess the data. The so-called bitshuffle44 transposes an array on the bit-level, such that bit positions (e.g. the sign bit) of floating-point numbers are stored next to each other in memory. Another example is the bitwise XOR-operation58 with the preceding floating-point value, which sets subsequent bits that are identical to 0. Neither bitshuffle nor XOR significantly increased the compression factors in our applications.

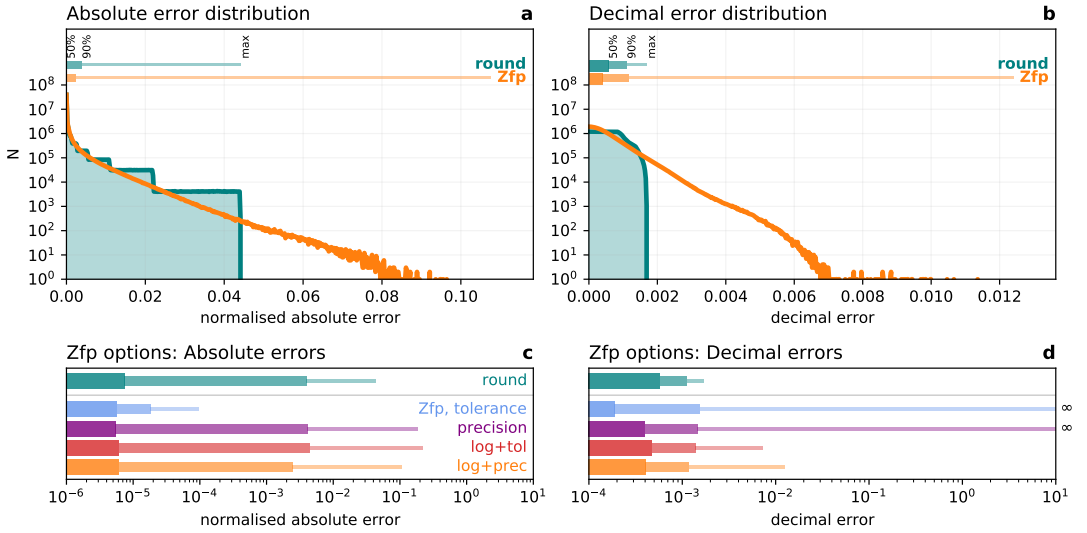

Figure 3.5 | Error distribution of binary rounding compared to Zfp compression. IEEE round-to-nearest and Zfp compression of water vapour (specific humidity) in the three spatial dimensions. **a, c** normalised absolute errors **b, d** decimal errors. 7 mantissa bits are retained for rounding corresponding to 99% preserved information. The precision parameter of Zfp is chosen to yield median errors that are at least as small as those obtained by rounding. **c, d** Zfp via specifying tolerance (tol) or precision (prec) with and without log-preprocessing. Maximum decimal errors that reached infinity in **d** due to sign changes are marked.

Matching retained bits to Zfp's precision

The Zfp compression algorithm divides a d-dimensional array into blocks of size 4^d to exploit correlation in every dimension of the data. Within each block a transformation of the data is applied with specified absolute error tolerance or precision, which bounds a local relative error. We use Zfp in its precision mode, which offers discrete levels to manually adjust the retained precision. Due to the rather logarithmic distribution of CAMS data (Fig. S1), a log-preprocessing of the data is applied to prevent sign changes (including a flushing to zero) within the compression. The error introduced by Zfp is approximately normally distributed and therefore usually yields higher maximum errors compared to round-to-nearest in float arithmetic, although median errors are comparable. In order to find an equivalent error level between the two methods, we therefore choose the precision level of Zfp to yield median absolute and decimal errors that are at least as small as those from rounding.

This method is illustrated in Fig. 3.5 in more detail: Errors introduced from round-to-nearest for floats have very rigid error bounds, the majority of errors from Zfp compression are within these bounds when matching median errors. However, given the normal distribution of errors with Zfp, there will be a small share of errors that is beyond the bounds from round-to-nearest. Using the precision mode of Zfp and log-preprocessed data bounds these maximum errors well (Fig. 3.5c and d).

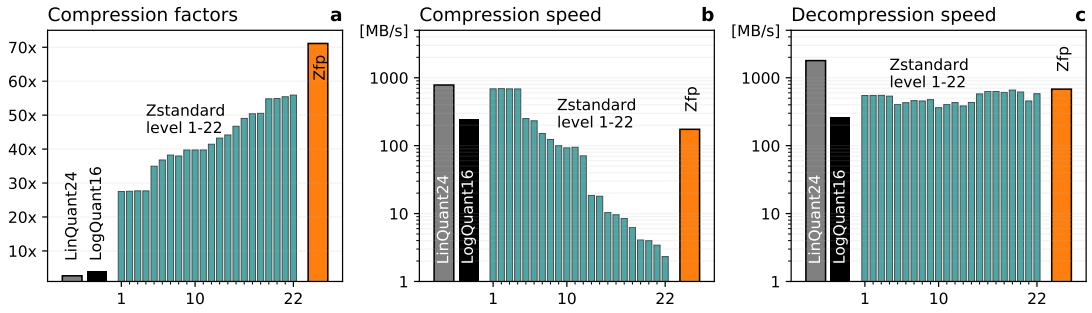

Figure 3.6 | Compressor performances. Compressing water vapour (specific humidity, variable code q) (3 mantissa bits retained, as in Fig. 3.11) with 24-bit linear quantization (LinQuant24), 16-bit logarithmic quantization (LogQuant16), round+lossless (Zstandard, compression level 1-22) and Zfp (precision-mode, including log-preprocessing): **a** Compression factors, **b** compression speed, **c** decompression speed. Timings are single-threaded on an Intel Core™ i7 (Kaby Lake) and do not include the writing to disk.

Compressor performances

Although different compressors and their performance are not within the central focus of this study, we analyse the compression and decompression speeds as a sanity check (Fig. 3.6). In order to find a data compression method that can be used operationally, a certain minimum data throughput should be achieved. The current 24-bit linear quantization method reaches compression speeds of almost 800 MB/s single-threaded on an Intel Core™ i7 (Kaby Lake) CPU in our application, excluding writing to disk. For the logarithmic quantization, this decreases to about 200 MB/s due to the additional evaluation of a logarithm for every value. For Zstandard the user can choose between 22 compression levels, providing a trade-off between the compression speed (highest for level 1) and the compression factor (highest for level 22). Compression speed reduces from about 700 MB/s at compression level 1 to 2 MB/s at level 22 (Fig. 3.6b), such that for high compression factors about a thousand cores would be required in parallel to compress in real time the 2GB/s data production at ECMWF. For Zstandard at compression level 10 speeds of at least 100MB/s are achieved, but at the cost of about 50% larger file sizes. We use compression level 10 throughout this study as a compromise. The decompression speed is independent of the level (Fig. S7c). The additional performance cost of binary rounding is with 2 GB/s negligible. Zfp reaches compression speeds of about 200 MB/s (single-threaded, including the log-preprocessing) in our application, enough to compress ECMWF's data production in real time with a small number of processors in parallel.

3.3 Drawbacks of current compression methods

The Copernicus Atmospheric Monitoring Service35 (CAMS) is performing operational predictions with an extended version of the Integrated Forecasting System IFS, the global atmospheric forecast model implemented by ECMWF. CAMS includes various atmospheric composition variables, like aerosols, trace and greenhouse gases that are important to monitor global air quality. The system monitors for example the spread of volcanic eruptions or emissions from wildfires. Most variables in CAMS have a multi-modal statistical distribution, spanning many orders of magnitude (Fig. 3.7).

The current compression technique for CAMS is the linear quantization, widely used in the weather and climate community through the data format GRIB236. CAMS uses the 24-bit version, which encodes values in a data array with integers from 0 to $2^{24}-1$. These 24-bit unsigned integers represent values linearly distributed in the min-max range. Un-

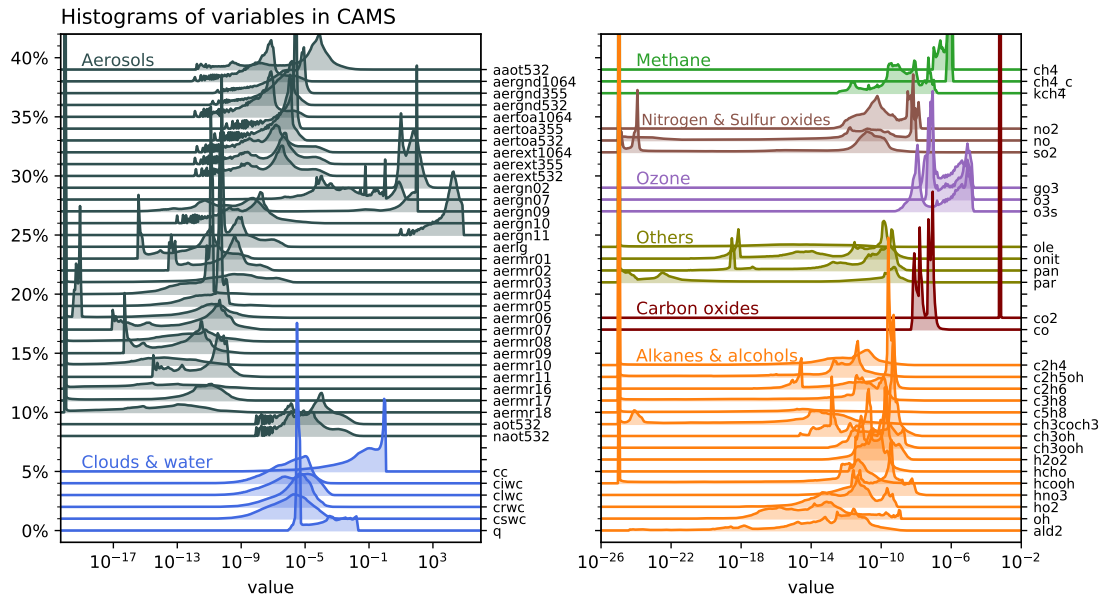

Figure 3.7 | Statistical distributions of all variables in CAMS. Histograms use a logarithmic binning and are staggered vertically for clarity. The variable abbreviations are explained in Table **??**.

used sign or exponent bits from the floating-point representation are therefore avoided and some of the trailing mantissa bits are discarded in quantization. Choosing the number of bits for quantization determines the file size, but the precision follows implicitly, leaving the required precision or amount of preserved information unassessed.

Although linear quantization bounds the absolute error, its linear distribution is unsuited for most variables in CAMS: Many of the available 24 bits are effectively unused as the distribution of the data and the quantized values match poorly (Fig. 3.8). Alternatively, placing the quantized values logarithmically in the min-max range better resolves the data distribution. As floating-point numbers are already approximately logarithmically distributed, this motivates compression directly within the floating-point format, which is also used for calculations in a weather or climate model and post-processing.

3.4 Bitwise real information content

Many of the trailing mantissa bits in floating-point numbers occur independently and at similar probability, i.e. with high information entropy21,22. These seemingly random bits are incompressible37–39, reducing the efficiency of compression algorithms. How-

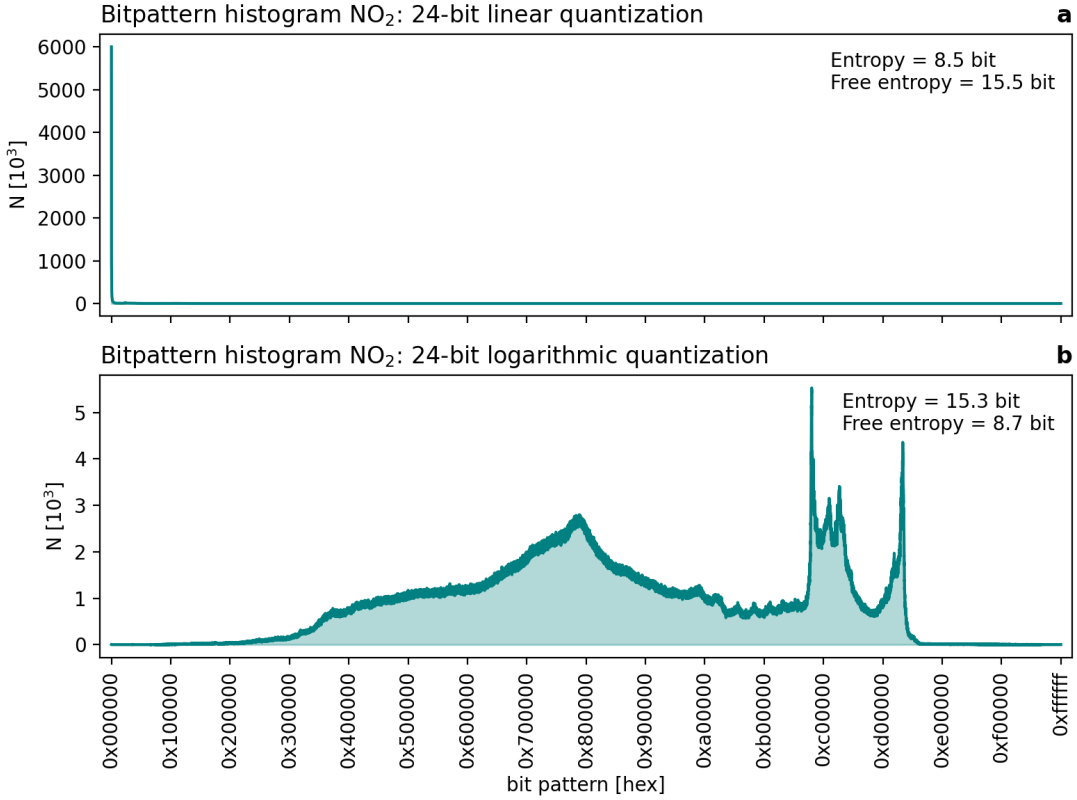

Figure 3.8 | Bitpattern histogram for linear and logarithmic quantization. **a** Linear 24-bit quantization and **b** 24-bit logarithmic quantization of nitrogen dioxide NO_2 mixing ratio [kg/kg]. All grid points and all vertical levels are used, consisting of $5.6 \cdot 10^7$ values with a range of $2 \cdot 10^{-14}$ to $2 \cdot 10^{-7}$ kg/kg. Bitpatterns are denoted in 24-bit hexadecimal. The free entropy is the difference between the available 24 bit and the bitpattern entropy and quantifies the number of effectively unused bits.

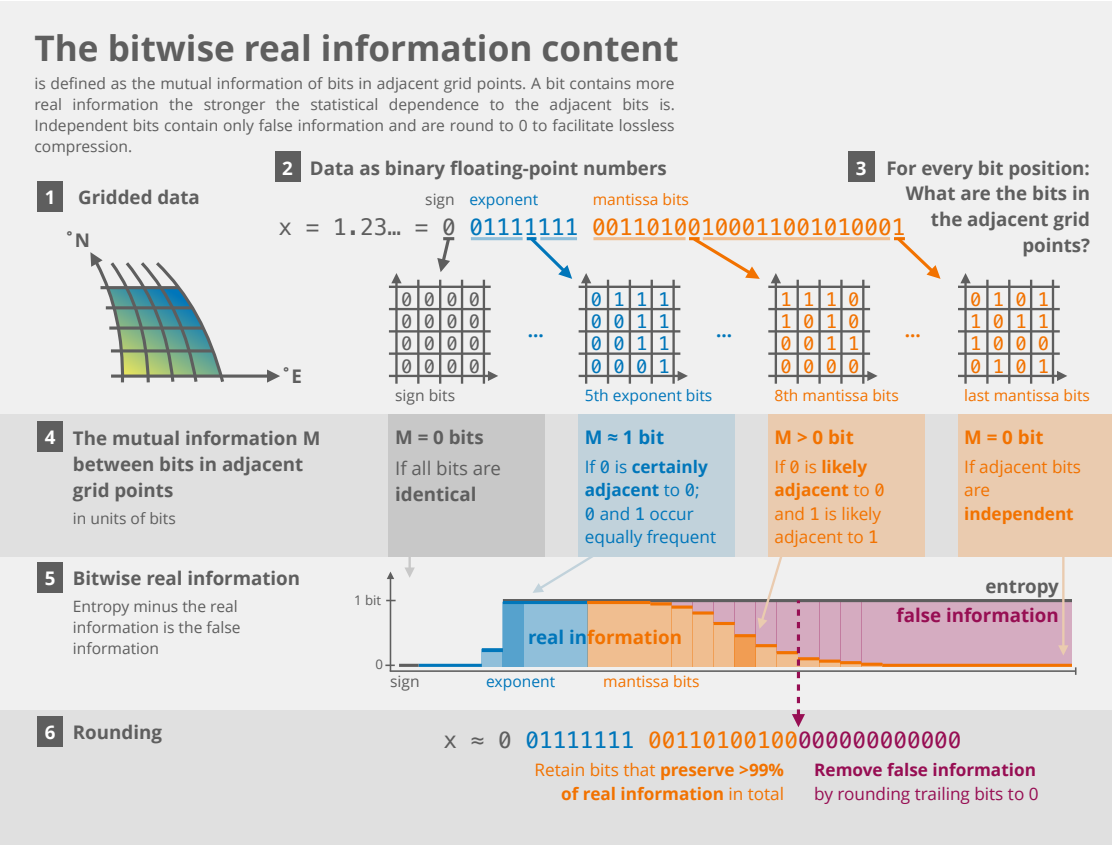

Figure 3.9 | The bitwise real information content explained schematically.

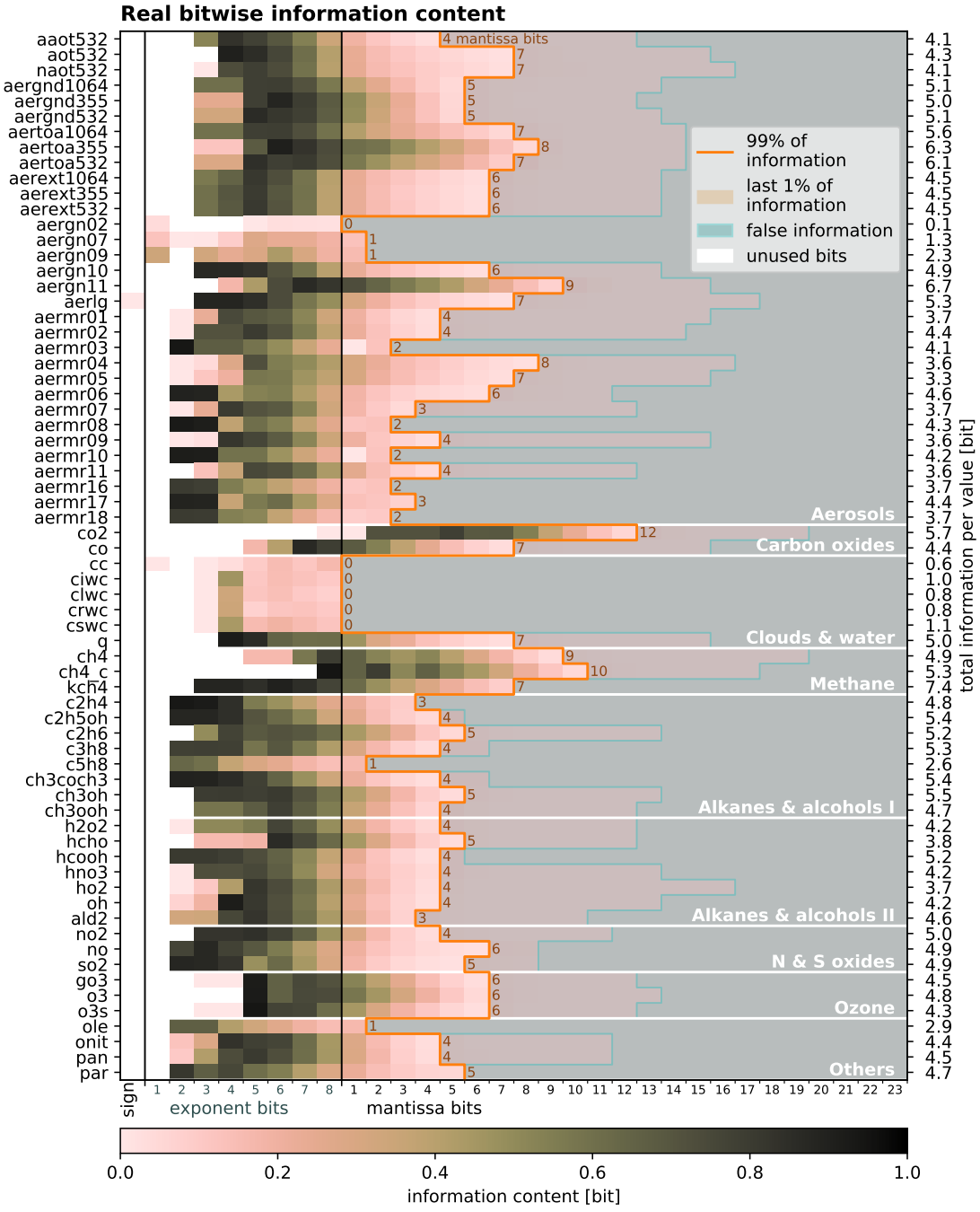

Figure 3.10 | Bitwise real information content for all variables in CAMS. The real information is calculated in all three spatial dimensions, revealing false information and unused bits, using the 32-bit encoding of single-precision floats. The bits that should be retained to preserve 99% of real information are enclosed in orange. Bits without any real information are shaded in grey-blue. The sum of the real information across bit positions per variable is the total information per value. Variable abbreviations are explained in Table **??**.

ever, they probably also contain a vanishing amount of real information, which has to be analysed to identify bits with and without real information. The former should be conserved while the latter should be discarded to increase compression efficiency.

We define the bitwise real information content as the mutual information20,37,40–43 of bits in adjacent grid points (Fig. **??**). A bit contains more real information the stronger the statistical dependence to the adjacent bits is. Bits without real information are identified when this dependence is insignificantly different from zero and we regard the remaining entropy in these bits as false information. The adjacent bit can be found in any of the dimensions of the data, e.g. in longitude, time or in the ensemble dimension. However, always the same bit position is analysed, e.g. the dependence of the first mantissa bit with other first mantissa bits in adjacent grid points.

In general, this analysis can be applied to any n-dimensional gridded data array when its adjacent elements are also adjacent in physical space, including structured and unstructured grids. However, data without spatial or temporal correlation at the provided resolution will be largely identified as false information due to the independence of adjacent grid points (Fig. 3.1 and 3.2). If valuable scientific information is present in such seemingly random data, then the bitwise real information content as defined here is unsuited.

? formulate the bitwise information content for simple chaotic systems, assuming an inherent natural uncertainty which had to be defined22. Their approach aims to enable reduced precision simulations on inexact hardware. Here, we reformulate the bitwise real information as the mutual information in adjacent grid points for the application in climate data compression. The quantization in the floating-point representation is used as an uncertainty, such that no additional assumption on the uncertainty of the underlying data has to be made. While most data compression techniques leave the choice of the retained precision to the user, the analysis here automatically determines a precision from the data itself based on the separation of real and false information bits.

Many exponent bits of the variables in CAMS have a high information content (Fig. 3.10), but information content decreases to zero within the first mantissa bits for most variables. Exceptions occur for variables like carbon dioxide (CO_2) with mixing ratios varying in a very limited range of 0.5-1.5 mg/kg (equivalent to about 330-990ppmv) globally. Due to the limited range, most exponent bits are unused and the majority of the real information is in mantissa bits 2 to 12.

The sum of real information across all bit positions is the total information per value, which is less than 7 bits for most variables. Importantly, the last few percent of total

information is often distributed across many mantissa bits. This presents a trade-off where for a small tolerance in information loss many mantissa bits can be discarded, resulting in a large increase in compressibility (Fig. 3.4a). Aiming for 99% preserved information is found to be a reasonable compromise.

3.5 Compressing only the real information

Based on the bitwise real information content, we suggest a new strategy for data compression of climate variables: First, we diagnose the real information for each bit position. Afterwards, we round bits with no significant real information to zero, before applying lossless data compression. This allows us to minimise information loss but to maximise the efficiency of compression algorithms.

Bits with no or only little real information (but high entropy) are discarded via binary round-to-nearest as defined in the IEEE-754 standard13 (see ??). This rounding mode is bias-free and therefore will ensure global conservation of quantities important in climate model data. Rounding removes the incompressible false information and therefore increases compressibility. While rounding is irreversible for the bits with false information, the bits with real information remain unchanged and are bitwise reproducible after decompression. Both the real information analysis and the rounding mode are deterministic, also satisfying reproducibility.

Lossless compression algorithms can be applied efficiently to rounded floating-point arrays (the round+lossless method). Many general-purpose lossless compression algorithms are available38,39,44–49, which are based on dictionaries and other statistical techniques to remove redundancies. Most algorithms operate on bitstreams and exploit the correlation of data in a single dimension only, we therefore describe this method as 1-dimensional (1D) compression. Here, we use Zstandard for lossless compression, which has emerged as a widely available default in recent years (see Methods).

The compression of water vapour at 100% preserved information (16 mantissa bits are retained) yields a compression factor of 7x relative to 64-bit floats (Fig. 3.11a). At 99% of preserved information (7 mantissa bits are retained) the compression factor increases to 39x. As the last 1% of real information in water vapour is distributed across 9 mantissa bits, we recommend this compromise to increase compressibility. With this compression a 15-fold storage efficiency increase is achieved compared to the current method at 2.67x. Effectively only 1.6 bits are therefore stored per value.

Compressing all variables in CAMS and comparing error norms reveals the advan-

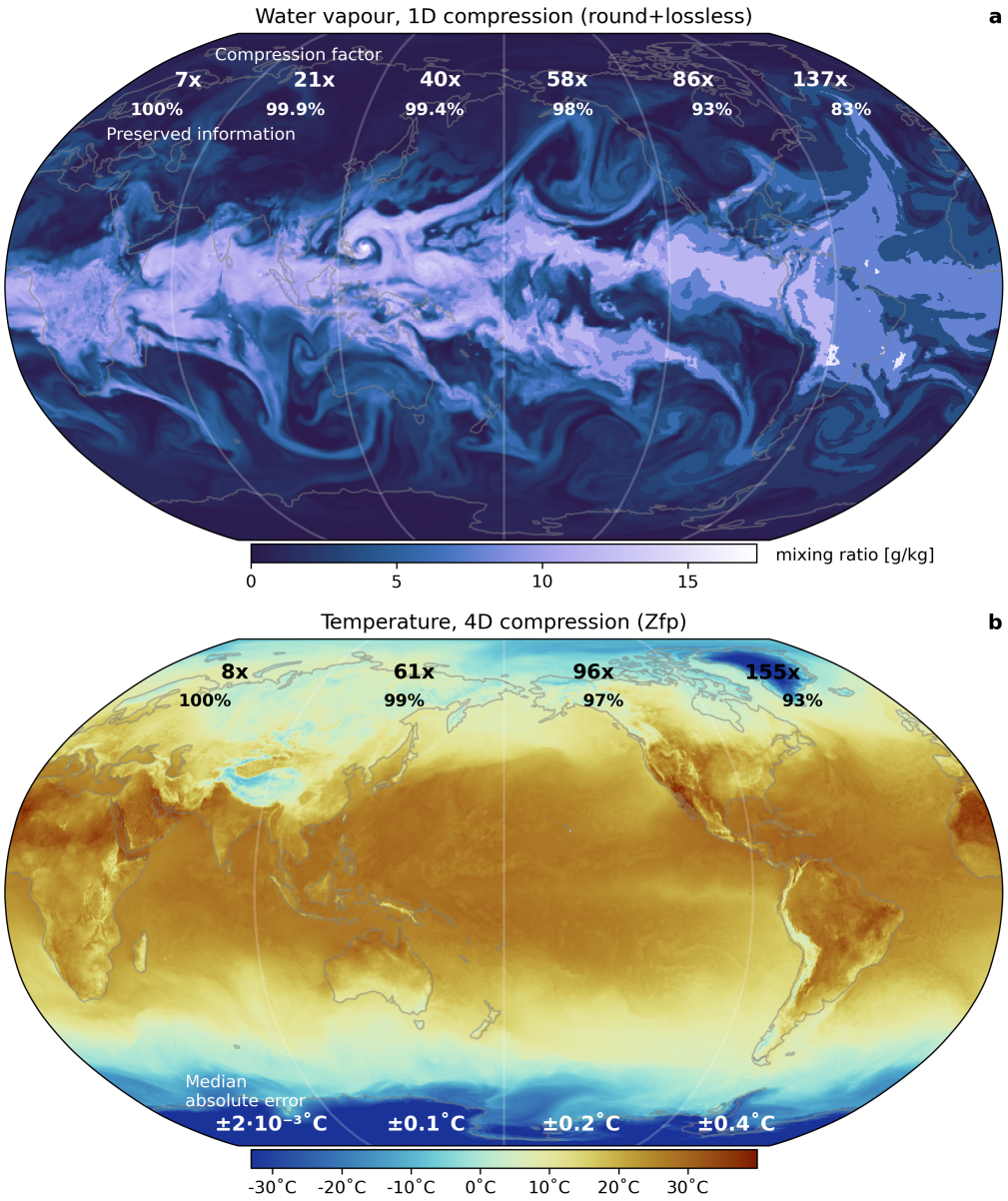

Figure 3.11 | Compression at various levels of preserved information. a Water vapour (specific humidity) compressed in the longitudinal dimension. The vertical level shown is at about 2 km geopotential altitude, but compression factors include all vertical levels. **b** Surface temperature compressed in the four space-time dimensions at various levels of preserved information with compression algorithm Zfp. Compression factors are relative to 64-bit floats.

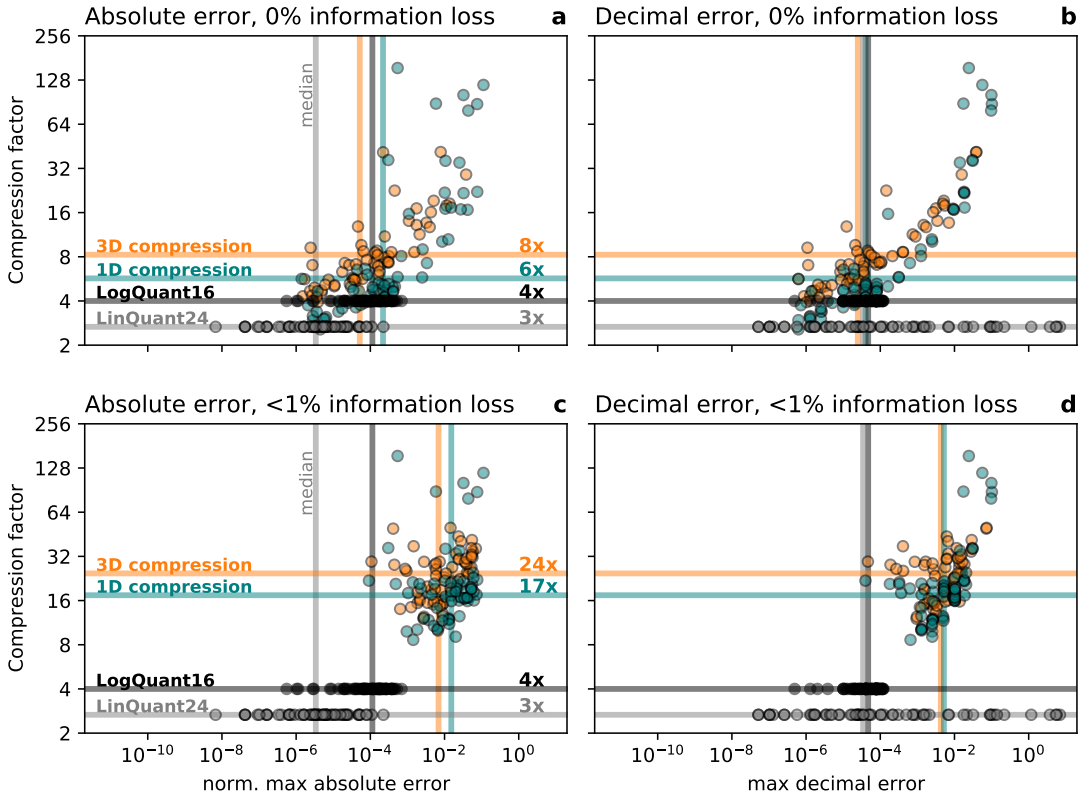

Figure 3.12 | Compression factors versus compression errors. The maximum absolute and decimal error for 24-bit linear and 16-bit logarithmic quantization (LinQuant24, LogQuant16) with 1-dimensional round+lossless and 3-dimensional Zfp compression. Every marker represents for one variable the global maximum of the **a, c** normalised absolute error, **b, d** decimal error for **a, b** 100% preserved information, and **c, d** 99% preserved information. The geometric mean of compression factors over all variables is given as horizontal lines. The median of the errors across all variables is given as vertical lines.

tages of the 1D round+lossless method compared to the 24-bit linear quantization technique currently in use (Fig. 3.12). The maximum decimal errors (see ??) are smaller for many variables due to the logarithmic distribution of floating-point numbers. Some variables are very compressible (>60x) due to many zeros in the data, which is automatically made use of in the lossless compression. Compression factors are between 3x and 60x for most variables, with a geometric mean of 6x when preserving 100% of information. Accepting a 1% information loss the geometric mean reaches 17x, which is the overall compression factor for the entire CAMS data set with this method when compared to data storage with 64 bits per value.

Furthermore, the 24-bit linear quantization could be replaced by a 16-bit logarithmic quantization, as the mean and absolute errors are comparable. The decimal errors are often even lower and naturally bound in a logarithmic quantization despite fewer available bits.

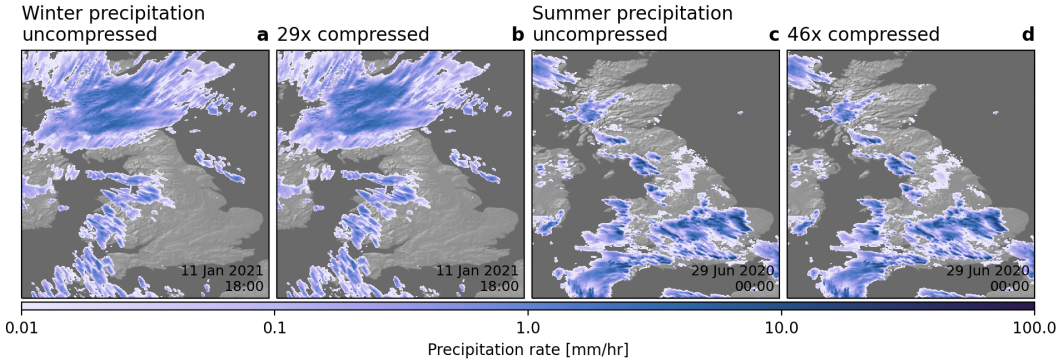

Figure 3.13 | Compression of radar-based observations of precipitation over Great Britain. **a** Precipitation for the hour preceding 18:00 UTC on 11 Jan 2021 from the UK MetOffice NIMROD data at about 1km horizontal resolution. **b** as a but the data was compressed preserving 99% of real information achieving compression factors of 29x relative to 64 bit. **c** and **d** as **a** and **b** but for 00:00 UTC on 29 Jun 2021 and achieving compression factors of 46x.

A broad applicability of the bitwise real information content analysis for compression is tested with further data sets: Radar-based observations of precipitation over Great Britain are similarly compressible using the same method (Fig. 3.13) and so are satellite measurements of brightness temperature with a very high resolution of about 300m horizontally (Fig. 3.14). Even for anthropogenic emissions of methane or nitrogen dioxide similar compression results are obtained, despite limited spatial correlation of point sources (Fig. 3.15). The bitwise real information content in this case is largely determined by the smooth background concentrations and therefore still sufficiently high

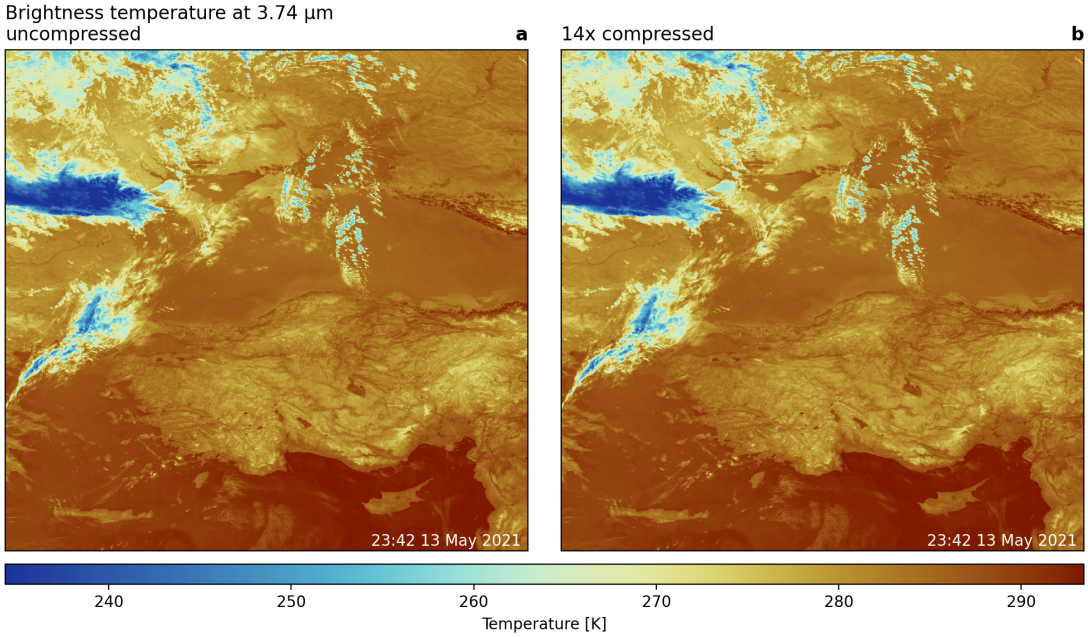

Figure 3.14 | Compression of satellite-based observations of brightness temperature over the Black Sea and Turkey. a Brightness temperature measured by the 3.74µm (I4) channel of the VIIRS sensor on board the Suomi-NPP satellite at about 300m horizontal resolution on the 13 May 2021. **b** as **a** but the data was compressed preserving 99% of real information with the round+lossless method achieving compression factors of 14x relative to 64 bit.

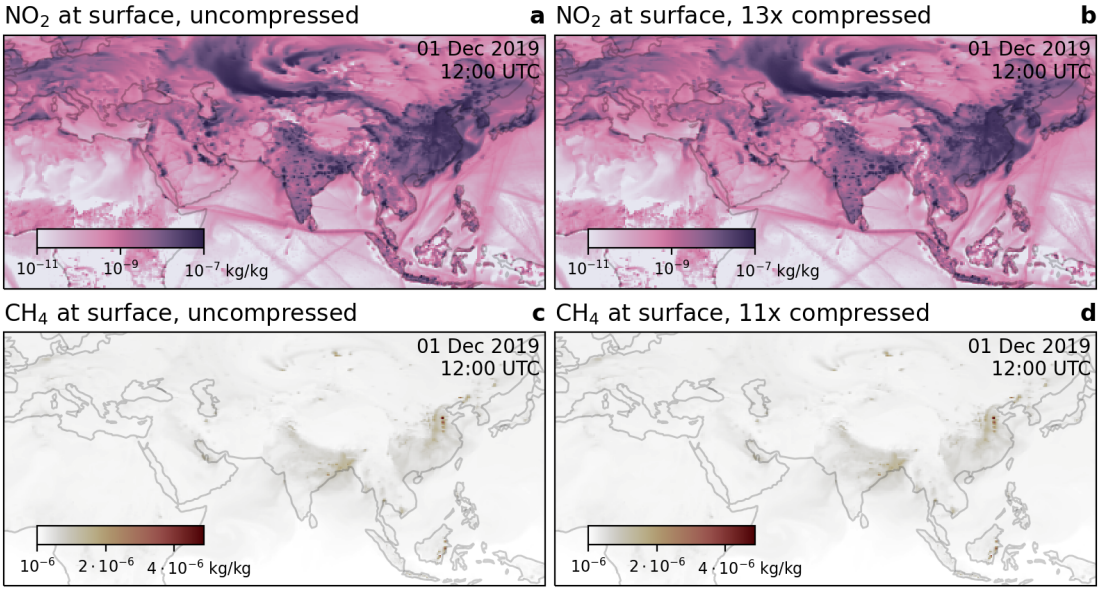

Figure 3.15 | Compression of nitrogen dioxide (NO_2) and methane (CH_4) at the surface. a Surface NO_2 concentrations preliminary result from fossil fuel combustion. **b** Surface CH_4 concentrations often include point sources, such as here in East China, East India and East Borneo. **b,d** as **a,c** but compressed preserving 99% of information achieving a compression factor of 13x, 11x, respectively.

to preserve the point sources.

In an operational setting we recommend the following workflow: First, for each variable the bitwise real information content is analysed from a representative subset of the data. Representative is, for example, a single time step for subsequent time steps if the statistics of the data distribution are not expected to change. From the bitwise real information the number of mantissa bits to preserve 99% of information is determined (the keepbits). Second, during the simulation the arrays that will be archived are rounded to the number of keepbits (which are held fixed) and compressed. The first step should be done offline, meaning once in advance of a data-producing simulation. Only the second step has to be performed online, meaning every time data is archived.

The presented round+lossless compression technique separates the lossy removal of false information and the actual lossless compression. This provides additional flexibilities as any lossless compressor can be used and application-specific choices can be made regarding availability, speed and resulting file sizes. However, most general-purpose lossless compression algorithms operate on bitstreams and require multidimensional data to be unravelled into a single dimension. Multidimensional correlation is therefore not fully exploited in this approach.

We extend the ideas of information-preserving compression to modern multidimensional compressors. The analysis of the bitwise real information content leads naturally to the removal of false information via rounding in the round+lossless method. For other lossy compressors, however, the separation of real and false information has to be translated to the precision options of such compressors. While such a translation is challenging in general, we present results from combining the bitwise real information analysis with one modern multidimensional compressor in the next section.

3.6 Multidimensional climate data compression

Modern compressors have been developed for multidimensional floating-point arrays10,30,31, which compress in several dimensions simultaneously. We will compare the 1D round+lossless compression to Zfp, a modern compression algorithm for two to four dimensions10. Zfp divides a d-dimensional array into blocks of 4^d values (i.e. the edge length is 4), which allows to exploit the correlation of climate data in up to 4 dimensions. To extend the concept of information-preserving compression to modern compressors like Zfp, the bitwise real information is translated to the precision options of Zfp (more details in the Methods).

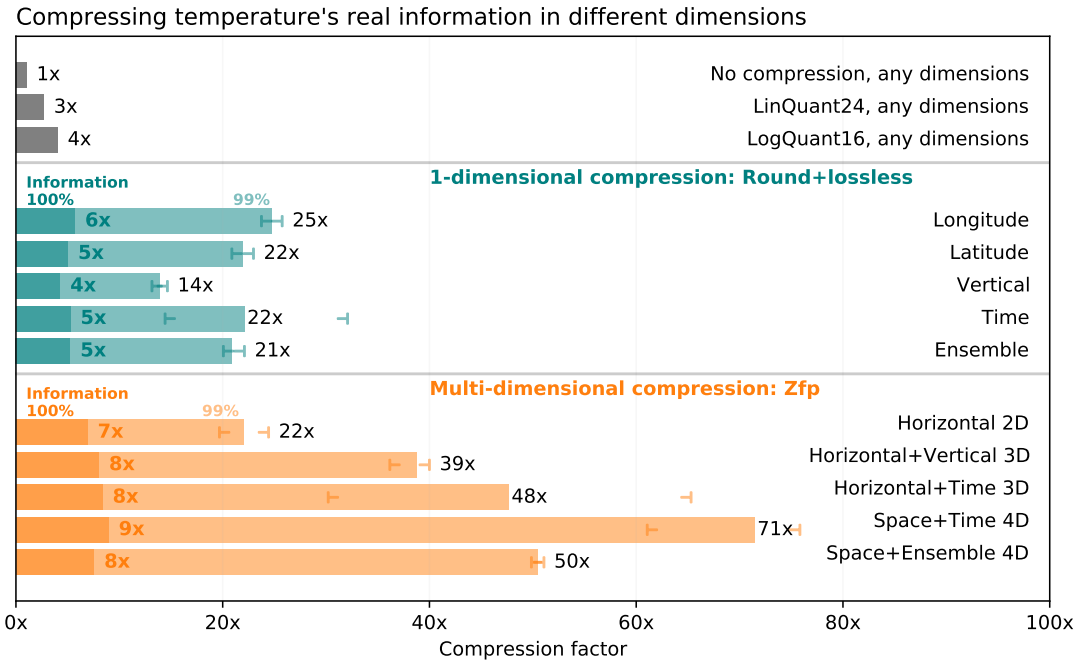

Figure 3.16 | Multidimensional compression allows for higher compression factors. 1-dimensional compression (round+lossless) of temperature reaches at most 25x when preserving 99% of real information with the round+lossless method, whereas 71x is reached with 4-dimensional (4D) space-time compression using Zfp compression. Preserving 100% of information considerably lowers the compression factors to 4-9x. Error brackets represent the min-max range of compression when applied to various data samples.

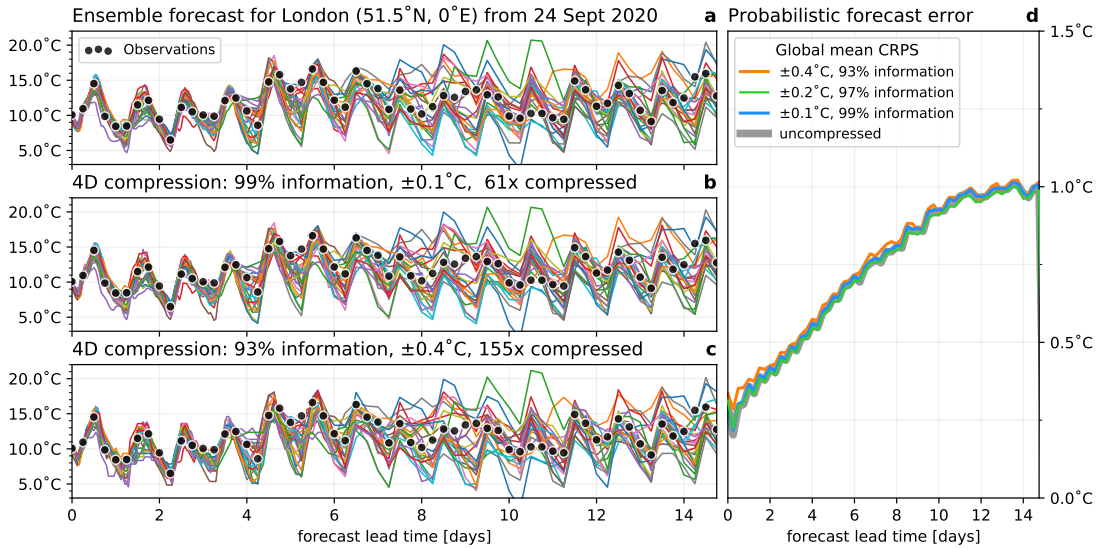

Figure 3.17 | Verification of an ensemble forecast with the probabilistic forecast error based on ensemble data with and without compression. **a** 25-member uncompressed ensemble forecast (lines) of surface temperature in London, UK from 24 Sept 2020 up to 15 days ahead. **b** as a but the data was compressed in 4-dimensional (4D) space-time with Zfp, preserving 99% of real information. **c** as **b** but only preserving 93% of real information. **d** Probabilistic forecast error (continuous ranked probability score, CRPS) for various levels of preserved information in the compression. The CRPS does not increases relative to the uncompressed reference for more than 93% of preserved information.

Multidimensional compression imposes additional inflexibilities for data retrieval: Data is compressed and decompressed in larger chunks, which can increase the load on the data archive. For example, if the data is compressed across the time dimension, data of several time steps have to be downloaded and decompressed although only data from a single time step might be requested. Downloads from an archive might therefore increase if the data chunking is not well suited to typical data requests from users.

For 1D compression the compressibility varies with the dimension: Longitude (i.e. in the zonal direction) is more compressible, reaching 25x for temperature at 99% preserved information, than compressing in the vertical which yields only 14x (Fig. 3.16). This agrees with the predominantly zonal flow of the atmosphere as spatial correlation in the zonal direction is usually highest. For a constant number of retained mantissa bits, higher resolution in the respective dimensions increases the compressibility as also the correlation in adjacent grid points increases (Fig. 3.1 and 3.2).

For multidimensional compression it is generally advantageous to include as many highly correlated dimensions as possible. In that sense, including the hourly-resolved forecast lead time instead of the vertical dimension in 3D compression yields higher compression factors. 4D space-time compression is the most efficient, reaching 60-75x at 99% preserved information. For temperature this is equivalent to a median absolute error of 0.1°C (Fig. 3.11b).

Compressing the entire CAMS dataset in the three spatial dimensions with Zfp while preserving 99% of the information yields an overall compression factor of 24x (Fig. **??**). Maximum absolute error and decimal errors are for most variables very similar to 1D round+lossless (see Methods for a discussion why they are not identical), providing evidence that a multidimensional compression is preferable for higher compression factors.

Due to the limited meaning of error norms in the presence of uncertainties in the uncompressed reference data, the forecast error is assessed to quantify the quality of compressed atmospheric data. The continuous ranked probability score50–52 (CRPS), a generalisation of the root-mean-square error for probabilistic forecasts, is evaluated for global surface temperature using observations every 6 hours as truth (Fig. **??**). Compared to the uncompressed data, no significant increase of the CRPS forecast error occurs for individual locations or globally at 99% and 97% preserved information. The usefulness for the end user of the global temperature forecast is therefore unaltered at these levels of preserved information in the compression. Contrarily, with an information loss larger than 5% the CRPS forecast error starts to increase, while large compres-

sion factors beyond 150x are achieved.

3.7 A data compression Turing test

In numerical weather predictions, progress in the development of global weather forecasts is often assessed using a set of error metrics, summarised in so-called *score cards*. These scores cover important variables in various large scale regions, such as 2m-temperature over Europe or horizontal wind speed at different vertical levels in the Southern Hemisphere. With a similar motivation as in **?**, we suggest assessing the efficiency of climate data compression using similar scores, which have to be passed similar to a Turing test33,54. The compressed forecast data should be indistinguishable from the uncompressed data in all of these score tests, or at least indistinguishable from the current compression method while allowing higher compression factors.

Many score tests currently in use represent area-averages (such as Fig. 3.17d), which would also be passed with coarse-grained data — reducing the horizontal resolution from 10km to 20km, for example, yields a compression factor of 4x. It is therefore important to include resolution-sensitive score tests such as the maximum error in a region. While a compression method either passes or fails such a data compression Turing test, there is additional value in conducting such a test. Evaluating the failures will highlight problems and evaluating the passes may identify further compression potential.

3.8 Discussion

While weather and climate forecast centres produce very large amounts of data, especially for future cloud and storm-resolving models, only the real information content in this data should be stored. We have here presented a methodology to identify real and false information in atmospheric, and more generally, climate data. This novel information-preserving compression relies on the removal of false information via rounding and can then be used in combination with any lossless compression algorithm. Applied to CAMS data we show that a high compressibility can be achieved without increasing the forecast error. The entire data set is 17x smaller in the compressed form when compared to 64-bit values but preserves 99% of the real information. This is about 6-times more efficient when compared to the current compression method.

Alternatively, the analysis of the bitwise real information content can be used to inform multidimensional compressors. Ideally, climate data compression should exploit

correlation in as many dimensions as possible for highest compression factors. The most important dimensions to compress along are longitude, latitude and time, which provide the highest compressibility. With Zfp we achieve factors of 60-75x for 4D space-time compression of temperature while preserving 99% of real information and without increasing forecast errors. Using the three spatial dimensions the entire set of variables in CAMS data can be compressed by 24x equivalently.

No additional uncertainty measure has to be assumed for the distinction of real and false information presented here. The uncertainty of a variable represented in a data array is directly obtained from the distribution of the data itself. Most lossy compression techniques leave the choice of precision to the user, which may lead to subjective choices or the same precision for a group of variables. Instead, our suggestion that 99% of information should be preserved may be altered by the user, which will implicitly determine the required precision for each variable individually.

To be attractive for large data sets, a compression method should enable compression as well as decompression at reasonable speeds. ECMWF produces data at about 2GB/s, including CAMS which creates about 15 MB/s. Data on ECMWF's archive is compressed once, but downloaded on average at 120 MB/s by different users, such that both high compression and decompression speeds are important. The (de)compression speeds obtained here are all at least 100MB/s single-threaded (Fig. 3.6), but faster speeds are available in exchange for lower compression factors (see ??). The real information is only analysed once and ultimately independent of the compressor choice.

Lossy compression inevitably introduces errors compared to the uncompressed data. Weather and climate forecast data, however, already contains uncertainties which are in most cases larger than the compression error. For example, limiting the precision of surface temperature to 0.1°C (as shown in Fig. 3.11b) is well below the average forecast error (Fig. 3.17d) and also more precise than the typical precision of 1°C presented to end users of a weather forecast. Reducing the precision to the real information content does not just increase compressibility but also helps to directly communicate the uncertainty within the data set — an important, often neglected, information by itself.

Satisfying requirements on size, precision and speed simultaneously is an inevitable challenge of data compression. As the precision can be reduced without losing information, we revisit this trade-off and propose an information-preserving compression. While current archives likely use large capacities to store random bits, the analysis of the bitwise real information content is essential towards efficient climate data compression.

4 Periodic orbits in chaotic systems

Contributions This chapter is largely based on the following publication

M Klöwer, P Coveney, EA Paxton, and TN Palmer, 2021. *On periodic orbits in chaotic systems simulated at low precision*, submitted.

with the following author contributions. Conceptualisation: MK, PC, EAP. Data curation: MK. Formal Analysis: MK. Methodology: MK. Visualisation: MK. Writing – original draft: MK. Writing – review and editing: MK, PC, EAP, TNP. The contributions of Peter, Adam and Tim are highly appreciated.

4.1 Introduction

Once upon a time

4.2 Methods

Wasserstein distance

An improved random number generator for uniformly distributed floats

Monte Carlo orbit search

Efficient orbit search with distributed computing

4.3 Revisiting the generalised Bernoulli map

The special $\beta = 2$ case

Effects of stochastic rounding

4.4 Orbits in the Lorenz 1996 system

The Lorenz 1996 system

Longer orbits with more variables

More variables instead of higher precision

4.5 Discussion

5 A 16-bit shallow water model

Contributions This chapter is largely based on the following publications

M Klöwer, PD Düben, and TN Palmer, 2019. *Posits as an alternative to floats for weather and climate models*, **CoNGA'19: Proceedings of the Conference for Next Generation Arithmetic**, Singapore, doi:10.1145/3316279.3316281.

M Klöwer, PD Düben, and TN Palmer, 2020. *Number Formats, Error Mitigation, and Scope for 16-Bit Arithmetics in Weather and Climate Modeling Analyzed With a Shallow Water Model*, **Journal of Advances in Modeling Earth Systems**, doi:10.1029/2020MS002246.

with the following author contributions. Conceptualisation: MK, PDD. Data curation: MK. Formal Analysis: MK. Methodology: MK. Visualisation: MK. Writing – original draft: MK. Writing – review and editing: MK, PDD, TNP. The contributions of Peter and Tim are highly appreciated.

5.1 Introduction

... which was often debated as a step in the wrong direction.

5.2 Methods

The shallow water model

Scaling and reordering the shallow water equations

Mixed precision

Compensated time integration

Reduced precision communication for distributed computing

A 16-bit semi-Lagrangian advection scheme

5.3 Impact of low-precision on the physics

Error growth

Mean and variability

Geostrophy

Gravity waves

Mass and tracer conservation

5.4 Discussion

6 Running on 16-bit hardware

6.1 Introduction

6.2 Methods

Choosing a scale with Sherlogs

Identifying subnormals with DrWatson

6.3 Squeezing ShallowWaters.jl into Float16

6.4 Approaching 4x speedups on A64FX

6.5 Discussion

7 Conclusions

7.1 Summary

7.2 Discussion

7.3 Outlook

Appendix

A.1 Open-source software developments

SoftPosit.jl

- Authors: M Kloewer, M Giordano, C Leong
- URL: github.com/milankl/SoftPosit.jl
- License: MIT
- Version: 0.3.0

SoftPosit.jl is a software emulator for posit arithmetic. The package exports the Posit8, Posit16, Posit32 number types among other non-standard types, as well as arithmetic operations, conversions and additional functionality. The package is a wrapper for the SoftPosit C-library written by C Leong.

StochasticRounding.jl

- Authors: M Kloewer
- URL: github.com/milankl/StochasticRounding.jl
- License: MIT
- Version: 0.1.0

StochasticRounding.jl is a software emulator for stochastic rounding in the Float32, Float16 and BFloat16 number formats. Both 16bit implementations rely on conversion to and from Float32 and stochastic rounding is only applied for arithmetic operations in the conversion back to 16bit. Float32 with stochastic rounding uses Float64 internally. Xoroshio128Plus is used as a high-performance random number generator.

ShallowWaters.jl

- Authors: M Kloewer
- URL: github.com/milankl/ShallowWaters.jl
- License: MIT
- Version: 0.3.0

ShallowWaters.jl is a shallow water model with a focus on type-flexibility and 16bit number formats, which allows for integration of the shallow water equations with arbitrary number formats as long as arithmetics and conversions are implemented. ShallowWaters also allows for mixed-precision and reduced precision communication.

ShallowWaters is fully-explicit with an energy and enstrophy conserving advection scheme and a Smagorinsky-like biharmonic diffusion operator. Tracer advection is implemented with a semi-Lagrangian advection scheme. Runge-Kutta 4th-order is used for pressure, advective and Coriolis terms and the continuity equation. Semi-implicit time stepping for diffusion and bottom friction. Boundary conditions are either periodic (only in x direction) or non-periodic super-slip, free-slip, partial-slip, or no-slip. Output via NetCDF.

Sherlogs.jl

- Authors: M Kloewer
- URL: github.com/milankl/Sherlogs.jl
- License: MIT
- Version: 0.1.0

Sherlogs.jl provides a number format Sherlog64 that behaves like Float64, but inspects your code by logging all arithmetic results into a 16bit bitpattern histogram during calculation. Sherlogs can be used to identify the largest or smallest number occuring in your functions, and where algorithmic bottlenecks are that limit the ability for your functions to run in low precision. A 32bit version is provided as Sherlog32, which behaves like Float32. A 16bit version is provided as Sherlog16T, which uses T for computations as well as for logging.

Sonums.jl

- Authors: M Kloewer
- URL: github.com/milankl/Sonums.jl
- License: MIT
- Version: 0.2.0

Sonums.jl is a software emulator for Sonums - the Self-Organizing NUMbers. A number format that learns from data. Sonum8 is the 8bit version, Sonum16 for 16bit computations. The package exports number types, conversions and arithmetics. Sonums con-

versions are based on binary tree search, and arithmetics are based on table lookups. Training can be done via maximum entropy or minimising the rounding error.

Float8s.jl

- Authors: M Kloewer, J Sarnoff
- URL: github.com/milankl/Float8s.jl
- License: MIT
- Version: 0.1.0

Float8s.jl is a software emulator for a 8bit floating-point format, with 3 exponent and 4 significant bits. The package provides the `Float8` number type, as well as arithmetic operations, conversions and additional functionality. The software emulator is based on conversion to and from Float32, which is used for arithmetic operations.

LogFixPoint16s.jl

- Authors: M Kloewer
- URL: github.com/milankl/LogFixPoint16s.jl
- License: MIT
- Version: 0.1.0

LogFixPoint16s.jl is a software emulator for 16-bit logarithmic fixed-point numbers with 7 signed integer bits and 8 fraction bits. The package provides the `LogFixPoint16` number type, as well as arithmetic operations, conversions and additional functionality. The software emulator is based on either integer addition or look-up tables and is therefore a comparably fast emulator.

Lorenz96.jl

- Authors: M Kloewer
- URL: github.com/milankl/Lorenz96.jl
- License: MIT
- Version: 0.3.0

Lorenz96.jl is a type-flexible one-level Lorenz 1996 model, which supports any number type, as long as conversions to and from Float64 and arithmetics are defined. Different number types can be defined for prognostic variables and calculations on the right-hand side, with automatic conversion on every time step. The equations are scaled such

that the dynamic range of numbers can be changed. The scaled equations are written division-free.

Lorenz63.jl

- Authors: M Kloewer
- URL: github.com/milankl/Lorenz63.jl
- License: MIT
- Version: 0.2.0

Lorenz63.jl is a type-flexible Lorenz 1963 model, which supports any number type, as long as conversions to and from Float64 and arithmetics are defined. The Lorenz equations are scaled such that the dynamic range of numbers can be changed. The scaled equations are written division-free.

Jenks.jl

- Authors: M Kloewer
- URL: github.com/milankl/Jenks.jl
- License: MIT
- Version: 0.1.0

Jenks.jl is the Jenks Natural Breaks Optimization, a data clustering method to minimise in-class variance or L1 rounding error. Jenks provides a data classification algorithm that groups one dimensional data to minimize an in-class error norm from the class mean but maximizes the same error norm between different classes.

Acknowledgements

I am very grateful for the support and very fruitful discussions with my supervisors Tim Palmer and Peter Düben, and especially for the freedom to develop my own ideas.

I would like to thank the whole Julia community for an uncountable effort to develop a very modern high performance computing language that is high-level, easy to learn and was proven to be increadibly useful for reduced precision simulations. I also would like to thank everybody who developed the matplotlib plotting library, which was used for every figure in this report.

Funding

I gratefully acknowledge funding from

- The European Research Council ERC under grant number 741112 *An Information Theoretic Approach to Improving the Reliability of Weather and Climate Simulations*
- The UK Natural Environmental Research Council NERC under grant number NE/L002612/1.
- The Copernicus Programme (EU Commission) through the ECMWF Summer of Weather Code 2020 and 2021
- The Graduate Research Allowance provided by Jesus College Oxford in 2018 and 2019