Weather and climate models in 16-bit arithmetics: Number formats, error mitigation and scope

M. Klöwer[1], P. D. Düben[2], and T. N. Palmer[1]

[1]Atmospheric, Oceanic and Planetary Physics, University of Oxford, Oxford, UK
[2]European Centre for Medium-Range Weather Forecasts, Reading, UK

Key Points:

○ Posit numbers have smaller rounding errors compared to floating-point numbers in weather and climate applications, enabling reliable shallow water simulations computed entirely with 16-bit arithmetic.
○ Errors caused by 16-bit floating-point arithmetic are strongly reduced with critical computations in 32 bit, which can be implemented on present-day hardware.
○ 16 or even 8-bit communication between processors, preferably encoded as posit numbers, introduces negligible errors, providing a perspective for reduced data communication for weather and climate models.

Corresponding author: M. Klöwer, `milan.kloewer@physics.ox.ac.uk`

Abstract

The need for high precision calculations with 64-bit floating-point arithmetic for weather and climate models has been questioned. Lower precision numbers can accelerate simulations and are increasingly supported by modern computing hardware. This paper investigates the potential of 16-bit arithmetic when applied within weather and climate applications. There are several 16-bit number formats that can potentially be used (IEEE half precision, BFloat16, posits, integer and fixed point). Furthermore, a number of mitigation methods are available to make model simulations resilient against a precision reduction. In this paper, the different number formats and mitigation methods are applied within the Lorenz 1963 toy model for atmospheric dynamics and a medium-complexity shallow water model. It is evident that a simple change from 64 to 16-bit arithmetic will not be possible for complex weather and climate applications as it will degrade model results or trigger number under- or overflows that cause a stalling of model dynamics or model instabilities. However, if mitigation methods are applied, 16-bit arithmetic can be used successfully within the applications of this paper. In particular, if the so-called posit number format is used as an alternative to the standard floating point representation for real numbers. The results of this paper show the potential of 16-bit formats for at least parts of complex weather and climate models where rounding errors would be entirely masked by intitial condition, model or discretization error.

1 Introduction

Weather and climate models provide predictions that are of great importance for society and economy. The Earth's climate system remains very difficult to predict even with the computational resources of the world's largest supercomputers, due to its complexity and non-linear dynamics that couple all features from the smallest time and length-scales to the largest. The forecast error of a weather forecast model has several origins (T. N. Palmer, 2012; T. Palmer, 2015): (i) Initial and boundary condition errors, which result from observations, data assimilation, model errors and external factors; (ii) model error, i.e. the difference between the mathematical model and the real world; (iii) discretisation error resulting from a finite spatial and temporal resolution of the discretised equations and (iv) rounding errors with finite precision arithmetic. In general, the forecast error is dominated by (i-iii), depending on the forecast variable and the forecast lead time. In contrast, rounding errors are usually negligible if the IEEE 754 standard on 64 bit double precision floating-point numbers (Float64) is used, which is the standard for the majority of operational weather forecasts and in climate models.

Research on reduced precision floating-point arithmetics is motivated by the potential for faster processing and communication between different elements of the computing architecture. The gained speed can be traded for increased complexity of simulations, resulting in more accurate predictions of weather and climate. The Integrated Forecast System at the European Centre for Medium-Range Weather Forecasts can be run almost entirely with 32-bit single precision (Float32) without a decrease in forecast skill (Váňa et al., 2017), but in 60% of the run-time. Similar progress was made at MeteoSwiss with their weather forecast model COSMO (Rüdisühli et al., 2013). For the European ocean model NEMO a mix of 32-bit and 64-bit arithmetic is a promising approach to keep accuracy-critical parts in high precision while increasing performance in others (Tintó Prims et al., 2019).

The recent boom of deep learning techniques, that require low numerical precision but high computational performance, is pushing hardware development to offer more flexibility for the use of reduced precision number formats. While 16-bit arithmetic was not available for use on commodity supercomputing hardware in the past, today most hardware vendors offer the use of 16-bit formats on the next generation of hardware. Graphic processing units started to support Float16 for increased performance (Markidis et al., 2018). Google's tensor processing units (TPU, Jouppi et al. (2017, 2018)) support the 16-bit BFloat16 format, a truncated version of Float32, as this format is sufficient for many deep learning applications (Kalamkar et al., 2019; Burgess et al., 2019; Gupta et al., 2015).

Using simplistic chaotic models, it was shown that the majority of 64 bits with Float64 do not contain real information (Jeffress et al., 2017). Running algorithms used for weather forecast models at precision lower than 32 bit, for example with 16-bit half precision floats (Float16), is an active field of research, but remains challenging (Düben & Palmer, 2014; Thornes et al., 2017; Hatfield et al., 2018; Düben, 2018). Most research on reduced precision modelling for weather and climate applications makes use of software emulators (Dawson & Düben, 2017) that provide other arithmetics than Float32 and Float64 which are widely supported on today's hardware. This comes with the disadvantage that simulations are orders of magnitude slower. However, software emulation allows a scientific evaluation of the use of reduced numerical precision for weather and climate simulations with no need to port the models to special hardware (such as FPGAs, (Russell et al., 2017)).

Posit numbers are a recently proposed alternative to floating point numbers and claim to provide more precision in arithmetic calculations with fewer bits in algorithms of linear algebra or machine learning (J. L. Gustafson & Yonemoto, 2017; Langroudi et al., 2019). However, posits remain untested for weather and climate simulations.

To get a better impression whether 16-bit arithmetic can be useful within weather and climate applications, which 16-bit formats are most promising, and how model simulations can be made resilient against a reduction in precision to 16 bits, this study will apply common types of 16-bit arithmetic in weather and climate applications and test approaches to mitigate negative impact if 16-bit arithmetic cannot be used for the entire model.

The paper is structured as follows: Section 2 outlines the different number formats, the concept of decimal precision, and mitigation methods how weather and climate models can be made resilient against a reduction of numerical precision to 16-bit formats. Section 3 presents results for the use of 16-bit arithmetic in a chaotic model at low complexity – namely the Lorenz 1963 system – and a model of medium complexity – namely the shallow water equations. The mitigation methods are applied to the two models to reduce the impact of the precision reduction. Section 4 discusses the results and provides the conclusions.

2 16-bit number formats and mitigation methods

This section will introduce the different types of 16-bit arithmetic that are available, will discuss similarities and differences between these types, and will introduce mitigation methods to allow for the use of 16-bit arithmetic within weather and climate applications.

2.1 16-bit number formats

2.1.1 The integer and fixed point number format

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 $0, 1, ..., n-2$. The largest representable integer for a signed integer format is $2^{n-1} - 1$. Fixed-point numbers extend the integer format by adding n_f fraction bits to decode an additional sum of powers of two with negative exponents $-1, -2, ..., -n_f$. 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 (Russell et al., 2017). 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. We will therefore focus on the discussion of the other formats in the rest of the study.

2.1.2 The floating-point number format

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

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

The exponent bits e are interpreted as unsigned integers, such that $e - bias$ converts them effectively to signed integers. The fraction (or significant) bits f_i are defined as before, such that the significand $(1 + f)$ 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

$$3.14 \approx 01001001_{\text{Float8}} = (-1)^{0} \cdot 2^{4-bias} \cdot (1 + 2^{-1} + 2^{-4}) = 3.125 \tag{2}$$

with $bias = 2^{n_e-1}-1 = 3$. Exceptions to Eq. 1 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 floating 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 signficant bits. 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 bits in the significand, consequently, they have a constant number of significant digits throughout their range of representable numbers (subnormals excluded).

2.1.3 The posit number format

The posit number format is explained in more detail in J. Gustafson (2017). Posit numbers arise from a projection of the real axis onto a circle (Fig. 1), with only one bit-pattern for zero and one for complex infinity (or Not-a-Real, NaR), 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. 1b). 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 very large or very small numbers. A positive posit number p is decoded as (J. L. Gustafson & Yonemoto, 2017; J. Gustafson, 2017) (negative posit numbers are converted first to their two's complement, see Eq. 5)

$$p = (-1)^{sign\ bit} \cdot useed^k \cdot 2^e \cdot (1 + f) \tag{3}$$

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. 3. They fill gaps of powers of 2 spanned by $useed = 4, 16, 256, ...$ 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 (J. L. Gustafson & Yonemoto, 2017; Chen et al., 2018). 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)

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

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 (Choo et al. (2003), denoted with an underscore subscript) by flipping all bits and adding 1,

$$\begin{aligned} -0.28 \approx 11011110_{\text{Posit}(8,1)} &= 10100010_{\_} \\ &= (-1)^1 \cdot 4^{-1} \cdot 2^0 \cdot (1 + 2^{-3}) = -0.28125. \end{aligned} \tag{5}$$

After the conversion to the two's complement, the bits are interpreted in the same way as in Eq. 4.

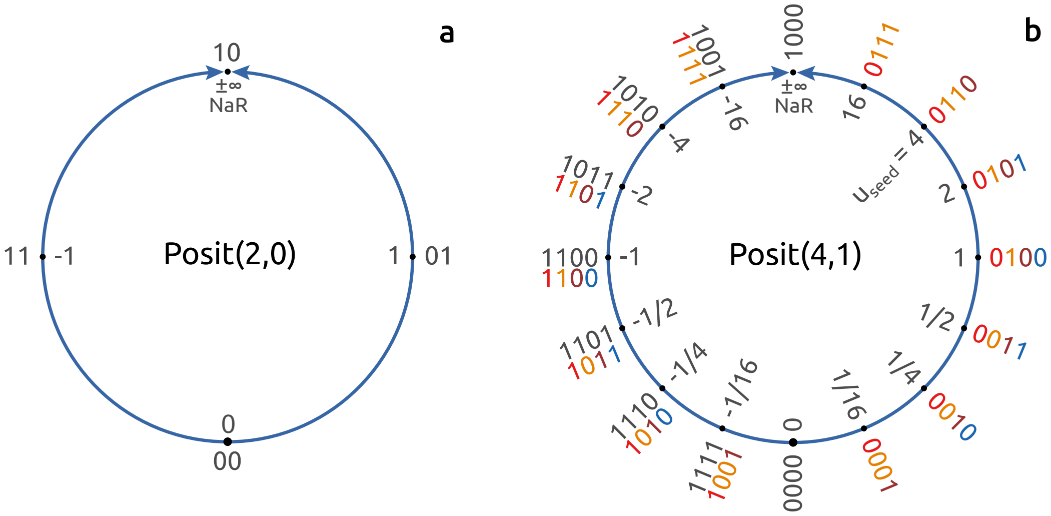

Figure 1. Two posit number formats obtained by projecting the real axis onto a circle. (a) 2-bit Posit(2,0) and (b) 4-bit Posit(4,1). The bit patterns are marked on the outside and the respective values on the inside of each circle. Bit patterns of negative numbers (black) have to be converted to their two's complement (colours) first (see text). At the top of every circle is complex infinity (±∞) or NaR (Not-a-Real). After J. Gustafson (2017).

Posits also working in 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 (J. Gustafson, 2017). Many functions are simplified for posits, as only two exceptions cases have to be handled, zero and NaR.

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.

As posit research currently focuses on hardware implementations (van Dam et al., 2019; Chen et al., 2018; Chaurasiya et al., 2018; Glaser et al., 2017), a standardized posit hardware is not yet available. In order to use posits on a conventional processor we developed for the Julia programming language (Bezanson et al., 2017) the posit emulator *SoftPosit.jl* (Klöwer & Giordano, 2019), which is a wrapper for the C-based library SoftPosit (Leong, 2020). The type-flexible programming paradigm, facilitated by the Julia language, is outlined in Appendix A.

2.2 The concept of decimal precision and summary of number formats

The decimal precision is defined as (J. L. Gustafson & Yonemoto, 2017; J. Gustafson, 2017)

$$\text{decimal precision} = -\log_{10} |\log_{10}(\frac{x_\text{repr}}{x_\text{exact}})| \tag{6}$$

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. The machine epsilon ϵ, a relative rounding error in floating-point arithmetic, usually defined as the distance δ between 1 and the next largest representable number, can be given in terms of decimal precision as $\epsilon = -\log_{10}(\log_{10}(1 + \frac{\delta}{2}))$.

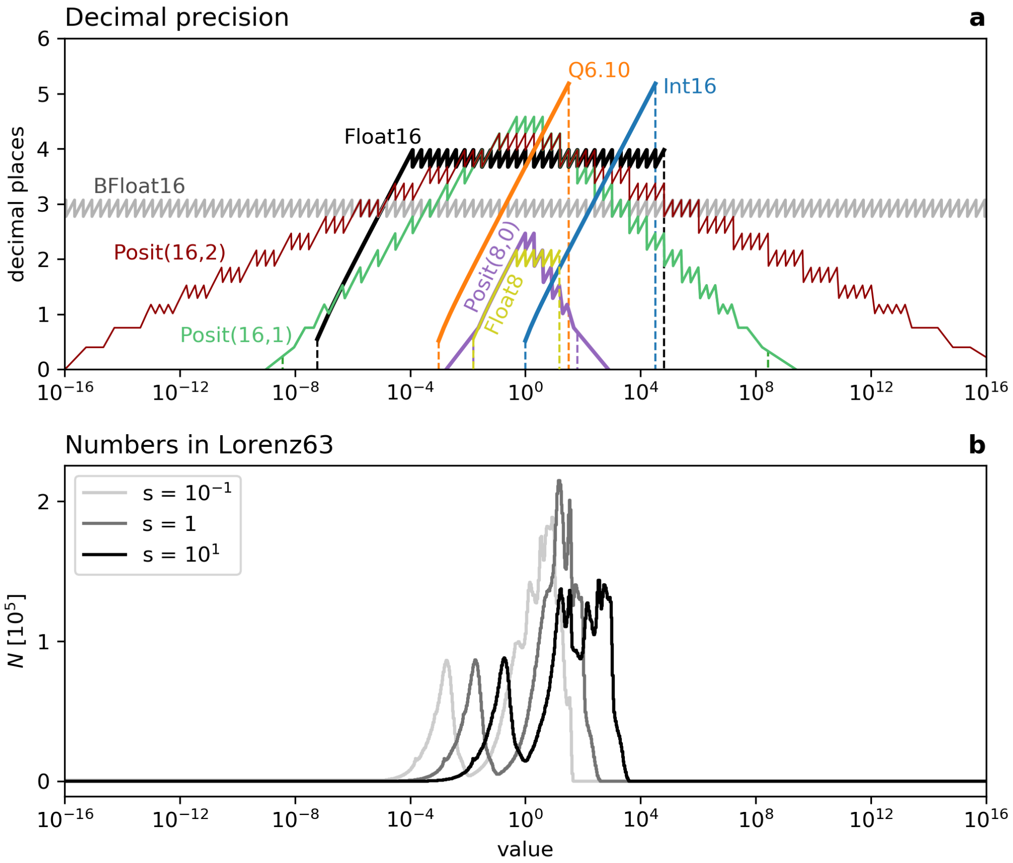

Figure 2. (a) Decimal precision of various number formats. Dashed vertical lines indicate the range of representable numbers for each format. Float64, Float32 and Posit32 are beyond the axes limits. (b) Histogram of results of all arithmetic operations in the rescaled Lorenz system, considering absolute values.

Fig. 2a compares the worst-case decimal precision for various 16 and 8-bit floats and posits, as well as 16-bit integers and the fixed-point format Q6.10 (6 integer bits, 10 fraction bits). Float16 has a nearly constant decimal precision of almost 4 decimal places, which decreases in 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}. 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.

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

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

Characteristics of various formats are summarised in Table 1. The machine error ϵ, defined as half the distance between 1 and the next representable number, can be given in terms of decimal precision and is summarized in Table 1 for the various formats. Float64 has more than 10^{15} bitpatterns reserved for NaN, but these only make up $< 0.05\%$ of all available bit patterns. However, the percentage of redundant bitpatterns for NaN increases for floats with fewer exponent bits and poses a noticable issue for Float16 and Float8.

2.3 Mitigation measures to allow for the use of 16-bit arithmetic in weather and climate applications

2.3.1 Mixed precision arithmetic

In many models, it will not be possible (or useful) to use 16-bit arithmetic throughout the entire model. Often, some model components will be more sensitive to a reduction in precision when compared to others and it often makes sense to reduce precision only in those places where a precision reduction does not deteriorate results while keeping precision high in precision-sensitive model components. This approach is called *mixed-precision* and is already used for the reduction to single precision in ocean and atmosphere models (Váňa et al., 2017; Tintó Prims et al., 2019).

2.3.2 Algorithmic changes: Rescaling, reordering and precomputations

It is possible to influence the range of numbers occuring in arithmetic operations in an algorithm using a *rescaling* of the equations via a simple multiplication of the variables with a constant rescaling factor s. Similar to the shift of the number range in fixed-point number representation, the rescaling of a linear equation will shift the dynamic range of the numbers that are used to the left (for $s < 1$) or right ($s > 1$) in Fig. 2. This process can be used to adjust the number range to the optimal distribution given the decimal precision of the different number formats. Rescaling is, however, more complicated if non-linear equations are considered as the scaling factor s appears inside the non-linear terms, such that the non-linear terms are invariant under scaling, that means only the linear terms are effectively shifted by s.

Furthermore, it is sometimes possible to avoid intermediate arithmetic results, which may be outside the dynamic range of a number format, by changing the order in which multiplications and divisions are executed. In general, it is preferable to combine such operations to a single multiplication with a constant, that can be precomputed. Although this will have a negligible effect on the rounding error for floating-point arithmetic due to the approximately constant decimal precision throughout the range of numbers (subnormals excluded), it reduces the risk of over- or underflow.

2.3.3 Reduced precision communication

Complex weather and climate models rely on parallelisation to distribute the computational cost of simulations efficiently among the processing units in a large cluster or supercomputer. Parallel execution typically requires domain decomposition, where the spatial domain is split into many subdomains to be calculated separately on individual processing units. Domain decomposition requires communication of the boundary values of a subdomain with the neighbouring subdomains. If 16-bit arithmetic cannot be used within the entire model, it may still be possible to reduce precision in the communication between processors.

Not all weather and climate models would benefit from a reduced precision communication as the acceleration potential depends on many factors specific to a model and the used hardware, e.g. number of nodes in a cluster and how shared and distributed memory management is realized. It will also be important whether communication is latency or volume bound. Latency bound communication is bound by the time a piece of information requires to travel between processors. In contrast, volume bound communication is limited by the bandwidth that is available for communication. Only the latter will benefit from a reduction in precision and therefore data volume.

However, in the case that the volume of the communication is an identified bottleneck in a given application, which is often the case in weather and climate models, it is possible that reliable model simulations can be achieved with 16 or even 8-bit communication allowing for a signficant reduction in computing time. The range of values that are communicated can be adjusted, facilitating a strong compression of the data that is communicated and opening the door to communication in very low precision.

3 Weather and climate applications with 16-bit arithmetics

In this section we will study the use of the different formats for 16-bit arithmetic and apply the mitigation methods to achive optimal results for the Lorenz 1963 system and a shallow water model.

3.1 The Lorenz 1963 system

The Lorenz system (called L63 in the following; (Lorenz, 1963)) is a chaotic attractor and serves as a simplistic model for atmospheric convection. It is an extensively studied toy model for forecast uncertainty (Lorenz, 1963; Kwasniok, 2014; Jeffress et al., 2017; Tantet et al., 2018) and is used here to investigate the accumulation of rounding errors in the numerical integration of a chaotic system. The Lorenz system consists of the variables x,y and z that are described by the following non-linear differential equations

$$\frac{dx}{dt} = \sigma(y - x) \tag{7a}$$
$$\frac{dy}{dt} = x(\rho - z) - y \tag{7b}$$
$$\frac{dz}{dt} = xy - \beta z \tag{7c}$$

with the typical parameter choices $\sigma = 10, \rho = 28$ and $\beta = \frac{8}{3}$, that permit chaotic behaviour.

Regardless of the initial conditions, the Lorenz system will evolve towards a set of (x, y, z) points called attractor (the x,z-section of the attractor is shown in Fig. 3a). This attractor is *strange*, i.e. its geometric structure cannot be described in two dimensions, but is of fractal nature. While points on the attractor will get infinitesimally close to each other, the trajectory of the analytical Lorenz system will never repeat itself. However, if the model is discretised and if the variables are represented with finite precision, only a finite amount of distinct states can be represented and the model trajectory will eventually repeat itself if integrated for long enough.

Integrating the Lorenz system with Float16 yields an attractor that is repeating itself fairly early and the space that is filled by the line of the trajectory is significantly smaller when compared to the space of a trajectory with double precision (compare Fig. 3a and b). However, when using Posit(16,1) and a rescaling factor of $s = 0.1$ (see section 3.1.1) the representation of the attractor is improved significantly (Fig. 3c). The results for posits look similar to the results with Float16 if no rescaling was used (not shown here). Only a time step $\Delta t > 0.008$ yields tendencies that are large enough to overcome the rounding errors of BFloat16 that otherwise inhibit a temporal evolution of the system. Although a time step of 0.008s allows the Lorenz system to be integrated with BFloat16, only a poor representation of the fractal attractor is achieved (Fig. 3d).

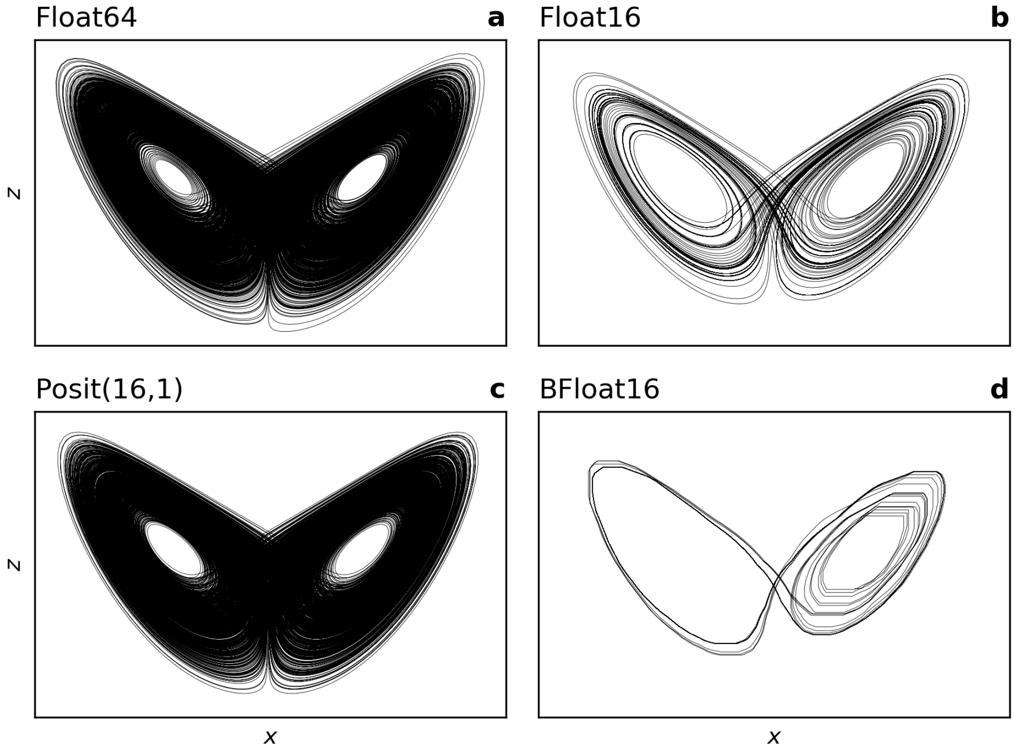

Figure 3. The Lorenz attractor computed with different number formats. The scaling of the Lorenz equations (Eq. 8) is (a,b,d) $s = 1$, (c) $s = 0.1$. All trajectories are integrated from the same initial conditions for 100,000 time steps with spacing $\Delta t = 0.008$ using the Runge-Kutta 4th order scheme.

The solution of the Lorenz system with 16-bit integers presents a similarly simple Lorenz attractor (Klöwer et al., 2019).

3.1.1 Rescaling in L63

To improve results, we apply a rescaling of the equations with a constant rescaling factor s. The rescaled variables are denoted as $\tilde{x} = sx$, and similarly for $\tilde{y}, \tilde{z}$. Fig. 2b shows histograms for all numbers that are used to solve the Lorenz system (including intermediate calculations). A comparison to the decimal precision in Fig. 2a reveals the benefit of rescaling, especially for posit arithmetic: To profit from the increased decimal precision around 1, a scaling with 1/10 is proposed to shift most calculations towards the centre of the dynamic range of representable numbers. Due to the constant decimal precision for floats, rescaling is less relevant for float arithmetic as long as no overflow nor underflow occurs. For integers, on the other hand, the Lorenz equations should be upscaled by a factor of approximately 100, to shift the range of numbers to a higher decimal precision.

We solve the equations using a fourth order Runge-Kutta method (Butcher, 2016). Each substep in the time integration can be written as

$$\tilde{x}^{n+1} = \tilde{x}^n + RK_x\left(\tilde{y}^n - \tilde{x}^n\right) \tag{8a}$$

$$\tilde{y}^{n+1} = \tilde{y}^n + RK_y\left(\tilde{x}^n(\rho - \frac{\tilde{z}^n}{s}) - \tilde{y}^n\right) \tag{8b}$$

$$\tilde{z}^{n+1} = \tilde{z}^n + RK_z\left(\tilde{x}^n\frac{\tilde{y}^n}{s} - \beta\tilde{z}^n\right) \tag{8c}$$

where RK_x, RK_y, RK_z contain the Runge-Kutta coefficient and the time step Δt. RK_x also contains the parameter σ. The superscripts n and $n{+}1$ denote the current and next time substep.

The rescaling of the Lorenz system has its limitations: The non-linear terms in Eq. 8 involve a division by the scaling constant s, which leads to the result of the arithmetic operations $\frac{\tilde{z}}{s}, \rho - \frac{\tilde{z}}{s}$, and $\frac{\tilde{y}}{s}$ being invariant under scaling. This is observed in the histograms of arithmetic results (Fig. 2b), as high counts of values between 1 and 50 exist for different choices of s. A changing shape of the histogram with s is a consequence. Following these results an underlying challenge of reduced precision modelling becomes apparent: One has either to find a number format that fits the range of computed numbers, or rescale the equations to optimise their range for a given number format.

3.2 Shallow water model

This section will evaluate the different number formats Float16, BFloat16, Posit(16,1) and Posit(16,2) when solving the shallow water equations. The shallow water equations result from a vertical integration of the Navier-Stokes equations under the assumption that horizontal length scales are much greater than vertical scales. This assumption holds for many features of the general circulation of atmosphere and ocean (Gill, 1982; Vallis, 2006). The shallow water equations for the prognostic variables velocity $\mathbf{u} = (u, v)$ and sea surface elevation η are

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} + f\hat{\mathbf{z}} \times \mathbf{u} = -g\nabla\eta + \mathbf{D} + \mathbf{F} \tag{9a}$$

$$\frac{\partial \eta}{\partial t} + \nabla \cdot (\mathbf{u}h) = 0. \tag{9b}$$

For the atmosphere, η is interpreted as pressure (Gill, 1982). The shallow water system is forced with a zonal wind stress $\mathbf{F}$. The dissipation term $\mathbf{D}$ removes energy on large scales (bottom friction) and on small scales (diffusion). The non-linear term $(\mathbf{u} \cdot \nabla)\mathbf{u}$ represents advection of momentum. The term $f\hat{\mathbf{z}} \times \mathbf{u}$ is the Coriolis force and $-g\nabla\eta$ is the pressure gradient force, with g being the gravitational acceleration. Eq. 9b is the shallow water-variant of the continuity equation, ensuring conservation of mass.

The shallow water equations are solved in the (x, y)-plane over the zonally periodic rectangular domain $L_x \times L_y$, of size $2000\,\mathrm{km} \times 1000\,\mathrm{km}$. We associate x with the zonal and y with the meridional direction. The domain is centred at 45N and the beta-plane approximation (Vallis, 2006) is used to linearize the Coriolis parameter which varies linearly from $9.5 \times 10^{-5}\,\mathrm{s}^{-1}$ at the southern boundary to $1.1 \times 10^{-4}\,\mathrm{s}^{-1}$ at the northern boundary. The boundary conditions are periodic in zonal direction and no-slip at the northern and southern boundary. The layer thickness is $h = \eta + H(x)$, with

$$H(x) = H_0 - H_1 \exp\left(-H_\sigma^{-2}(x - \tfrac{L_x}{2})^2\right) \tag{10}$$

being the undisturbed depth, representing a meridional mountain ridge at $x = \frac{L_x}{2}$ spanning from the southern to the northern boundary. The standard depth is $H_0 = 500\,\mathrm{m}$. The ridge has a height of $H_1 = 50\,\mathrm{m}$. The characteristic width of the ridge is $H_\sigma = 300\,\mathrm{km}$. The time step $\Delta t = 282\,\mathrm{s}$ is chosen to resolve surface gravity waves, traveling at an estimated phase speed of $\sqrt{gH_0}$ with CFL number being close to 1 and gravitational acceleration $g = 10\,\mathrm{ms}^{-1}$. The wind stress forcing $\mathbf{F} = (F_x, 0)$ is constant in time, acts only on the zonal momentum budget

$$Fx = \frac{F_0}{\rho h} \cos\left(\pi\left(y{L_y}^{-1} - 1\right)\right)^2 \tag{11}$$

and vanishes at the boundaries. The water density is $\rho = 1000\,\mathrm{kg\,m^{-3}}$ and $F_0 = 0.12\,\mathrm{Pa}$. The dissipation term $\mathbf{D}$ is the sum

$$\mathbf{D} = -\frac{c_D}{h}\|\mathbf{u}\|\mathbf{u} - \nu\nabla^4\mathbf{u} \tag{12}$$

of a quadratic bottom drag with dimensionless coefficient $c_D = 10^{-5}$ (Arbic & Scott, 2008) and a biharmonic diffusion with viscosity coefficient $\nu \approx 1.33\times10^{11}\,\mathrm{m^4\,s^{-1}}$ (Griffies & Hallberg, 2000).

The shallow water equations are discretised using 2nd order centred finite differences on an Arakawa C-grid (Arakawa & Lamb, 1977) and the fourth order Runge-Kutta method (Butcher, 2016) is used for time integration of the pressure, coriolis and advective terms, whereas a semi-implicit method is used for the dissipative terms $\mathbf{D}$. We present results of simulations with three different levels of resolution: high resolution simulations with a grid spacing of $\Delta = 5\,\mathrm{km}$ (400x200 grid points), medium resolution simulations with a grid-spacing of $\Delta = 20\,\mathrm{km}$ (100x50 grid points) and low resolution with a grid-spacing of $\Delta = 40\,\mathrm{km}$ (50x25 grid points). The advection terms are discretised using an energy and enstrophy conserving scheme (Arakawa & Hsu, 1990).

To test the use of the different number formats for the representation of passive tracers in atmosphere and ocean, we extend the shallow water equations with an advection equation. Tracers could, for example, be temperature and salinity in the ocean or aerosols in the atmosphere, which are regarded here, for simplicity, as passive (i.e. they do not influence the flow). The change of the distribution of a passive tracer q that is advected by the underlying flow field is described by

$$\frac{\partial q}{\partial t} + \mathbf{u}\cdot\nabla q = 0. \tag{13}$$

We discretise Eq. 13 with a semi-Lagrangian advection scheme (Smolarkiewicz & Pudykiewicz, 1992), which calculates the tracer concentration for a given grid cell from the concentration at the previous time step at a departure point, which in turn is determined from the flow field. As the departure point is usually in between grid nodes an interpolation is required to find the concentration at the departure point, which is then the concentration at the arrival point.

3.2.1 Rescaling the shallow water equations

For 16-bit arithmetic it is essential to re-order the calculations in the shallow water equations to avoid calculations with very large or very small results, as the dynamic range of representable numbers is limited (Fig. 2a and Table 1). This is especially true for some sophisticated schemes like the biharmonic diffusion (Griffies & Hallberg, 2000), which is often used to remove energy from the grid scale to ensure numerical stability. For biharmonic diffusion a fourth derivative in space is calculated. Due to the large dimension of geophysical applications, this term can get very small $\mathcal{O}(10^{-20})$ while viscosity coefficients are typically very large $\mathcal{O}(10^{11})$. The prognostic variables of Eq. 9 and 13 are typically $\mathcal{O}(1\,\mathrm{ms}^{-1})$ for $\mathbf{u}$, $\mathcal{O}(1\,\mathrm{m})$ for η and $\mathcal{O}(1)$ for q. We therefore retain their physical units in the discretised numerical model and do not apply a rescaling of the shallow water equations. However, due to the grid spacing Δ of unit meter being large for geophysical flows, we need to use dimensionless Nabla operators $\tilde{\nabla} = \Delta\nabla$. The continuity equation Eq. 9b, for example, is discretised with an explicit time integration method as

$$\eta^{n+1} = \eta^n + RK_\eta \left(-\tilde{\nabla} \cdot (\mathbf{u}h)^n \right) \tag{14}$$

where RK_η is the Runge-Kutta coefficient times $\frac{\Delta t}{\Delta}$ which is precomputed at high precision, to avoid a division by a large value for Δ and a subsequent multiplication with a large value for Δt. The other terms are rescaled accordingly ($\tilde{f} = f\Delta$; $\tilde{\mathbf{F}} = \mathbf{F}\Delta$). As these terms remain constant, they are precomputed at higher precision during model initialisation to avoid problems with the dynamic range. To avoid division and subsequent multiplication with large numbers in the dissipative terms (Eq. 12) throughout the numerical model integration, we rescale $\mathbf{D}$ accordingly

$$\tilde{\mathbf{D}} = -\frac{\tilde{c_D}}{h} \|\mathbf{u}\|\mathbf{u} - \tilde{\nu}\tilde{\nabla}^4\mathbf{u} \tag{15}$$

with $\tilde{c_D} = c_D\Delta = 0.2\,\mathrm{m}$, and $\tilde{\nu} = \nu\Delta^{-3} \approx 0.16\,\mathrm{ms}^{-1}$, which are precomputed. Computing the term $\tilde{\mathbf{D}}$ instead of $\mathbf{D}$ is required to avoid arithmetic under and overflow with floats or huge rounding errors with posit arithmetic.

How to reformulate the semi-Lagrangian advection scheme for 16-bit arithmetics is explained in the following. This semi-Lagrangian advection scheme is based on the idea to solve the advection equation with respect to its Lagrangian formulation. In the absence of sources and sinks, the Lagrangian point-of-view states that the tracer concentration q does not change following a flow trajectory. The concentration q at departure points $\mathbf{x}_d$ at time t is therefore the same as the concentration at time $t+\Delta t_{\text{adv}}$ at arrival points $\mathbf{x}_a$, which are chosen to coincide with the grid points. Based on the flow velocity at the arrival point, the departure point is derived. In order to avoid large numbers of the coordinates ($L_x = 2 \cdot 10^6$m), non-dimensional departure points $\tilde{\mathbf{x}}_{d,rel}$ relative to the arrival point are computed as

$$\tilde{\mathbf{x}}_{d,rel} = -\mathbf{u}(\mathbf{x}_a, t + \Delta t_{\text{adv}}) \left(\frac{\Delta t_{\text{adv}}}{\Delta} \right). \tag{16}$$

A scaling with the grid-spacing inverse Δ^{-1} is applied such that all terms are $\mathcal{O}(1)$ and therefore representable with 16-bit arithmetics. In practice, when converting the relative departure point $\tilde{\mathbf{x}}_{d,rel}$ to an array index for the interpolation, the floor function is used in combination with integer arithmetics. This essentially separates a computation with reals into two parts. One that can be computed with integers without rounding errors, and a calculation with reals, with a removed offset to reduce rounding errors.

3.2.2 Shallow water simulations in 16-bit arithmetic

The solution to the shallow water equations includes vigorous turbulence that dominates a meandering zonal current. Using either float or posit arithmetic in 16 bit the simulated fluid dynamics are very similar to a Float64 reference: As shown in a snapshot of tracer concentration (Fig. 4) turbulent stirring and mixing can be well simulated with posits. However, the Float16 simulation (Fig. 4d) deviates much faster than the posit simulations (Fig. 4b and c) from the Float64 reference (Fig. 4a), presumably due to the small scale instabilities visible in the snapshot as wavy filaments and fronts. These instabilities are clearly triggered by Float16 arithmetics, but to a lower degree also visible for posits. This provides a first evidence that the accumulated rounding errors with posits are smaller than with floats. BFloat16 arithmetic is not able to simulate the shallow water dynamics, presumably as tendencies are too small to be added to the prognostic variables, an issue that also occurs in the Lorenz system (Fig. 3d) and even in the harmonic sum (Fig. A2).

To quantify differences between the different 16-bit arithmetics we perform short-term forecasts with the medium-resolution configuration. To quantify the error growth of rounding errors with different arithmetics in a statistically robust way, we create a number of forecasts with each member starting from one of 200 randomly picked start dates from a 50 year long control simulation. The forecast error in the shallow water model

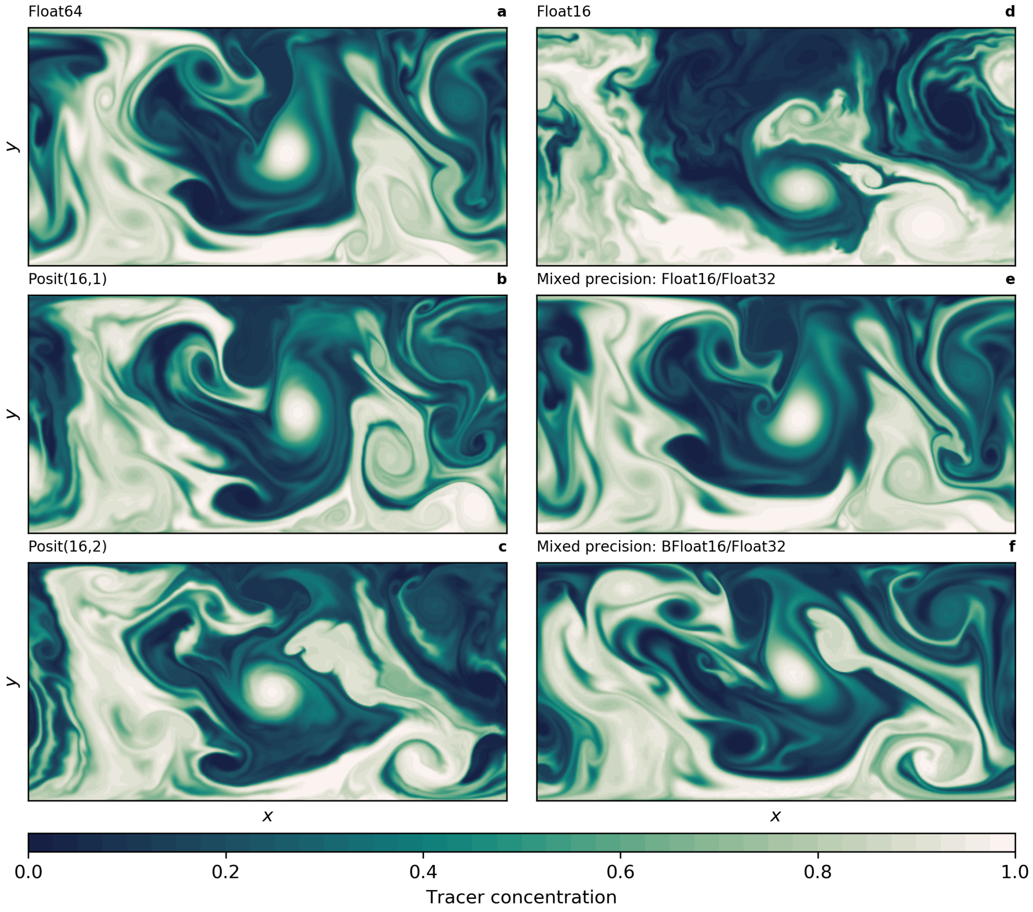

Figure 4. Snapshot of tracer concentration simulated by the shallow water model using different 16-bit number formats and the medium-resolution configuration. The mixed precision simulations presented in (e) and (f) are using Float32 for the representation of prognostic variables only. The tracer was injected uniformly in the lower half of the domain 50 simulation days before the plot. These simulations were run in the high-resolution configuration.

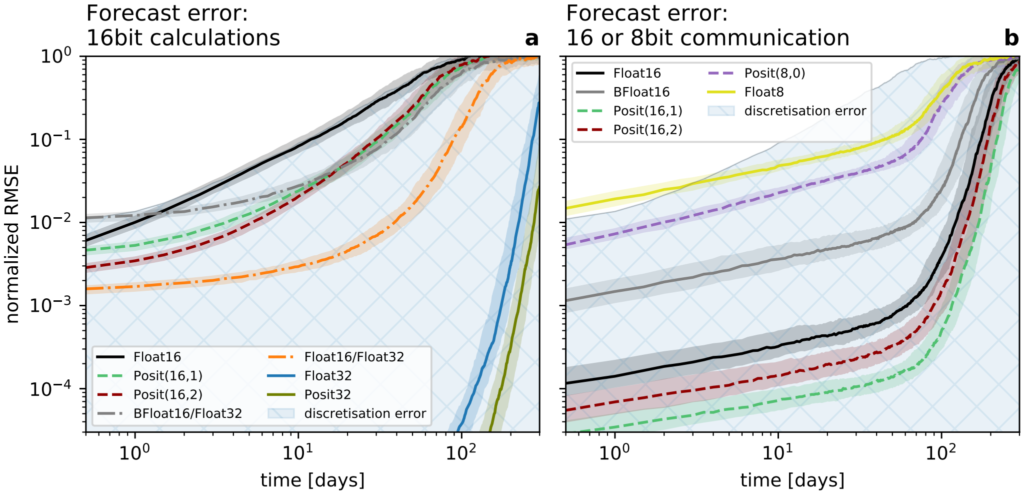

Figure 5. Forecast error measured as the root mean square error (RMSE) of sea surface height η taking Float64 as reference. (a) Forecast error for various 16-bit number formats and mixed 16/32 bit simulations for which the prognostic variables are kept at Float32. (b) Forecast error for reduced precision communication in 8 or 16 bit with various number formats used for encoding, with Float64 used for all calculations. The communication of boundary values occurs at every time step for the prognostic variables. The RMSE is normalised by a mean forecast error at very long lead times. Solid lines represent the median of 200 forecasts per number format. The shaded areas of each model configuration denote the interquartile range of the forecast experiments.

is computed as root mean square error (RMSE) of sea surface height with respect to Float64 simulations. Other variables yield similar results. Each forecast is performed several times from identical initial conditions but with the various number formats. To compare the magnitude of rounding error that are caused by a reduction in precision to a realistic level of error that is caused by model discretisation, we also perform forecasts with Float64 and the low-resolution model configuration, which are used to estimate the discretization error. We normalise the RMSE by the climatological mean forecast error at very long lead times, which is the same for all model configurations. A normalised RMSE of 1 therefore means that all information of the initial conditions is removed by chaos.

The forecast error of Float16 is as large as the discretisation error and clearly outperformed by 16-bit posit arithmetic (Fig. 5a). Both Posit(16,1) and Posit(16,2) yield a forecast error that is several times smaller than Float16. The forecast error of 32-bit arithmetic is several orders of magnitude smaller and is only after 200 days as large as the error for 16-bit arithmetic at very short lead times. Also at 32 bit, posits clearly outperform floats.

To investigate the effect of rounding errors on the climatological mean state of the shallow water system, we zonally average the zonal velocity u. The mean state is an eastward flow of about 0.3 m/s, about 3 to 4 times weaker than individual velocities throughout the domain (Fig. 6a), which is typical for turbulent flows. A weak westward mean flow is found at the northern and southern boundary. No 16-bit format was found to have a significant impact on the mean state. The variability of the flow around its mean state is high throughout the domain (Fig. 6b). The variability is significantly increased by 10 – 30% with 16-bit arithmetic, especially with Posit(16,2). This is probably caused by rounding errors that are triggering local perturbation which increase variability.

The turbulence in shallow water simulations is largely geostrophic, such that the pressure gradient force opposes the Coriolis force. The resulting geostrophic velocities $\mathbf{u}_g$ can be derived from the sea surface height η

$$\mathbf{u}_g = \frac{g}{f}\hat{\mathbf{z}} \times \nabla\eta \tag{17a}$$

$$\mathbf{u} = \mathbf{u}_g + \mathbf{u}_{ag} \tag{17b}$$

and deviations from the actual flow $\mathbf{u}$ are the ageostrophic velocity components $\mathbf{u}_{ag}$. We project both components on the actual velocities to obtain the flow-parallel components

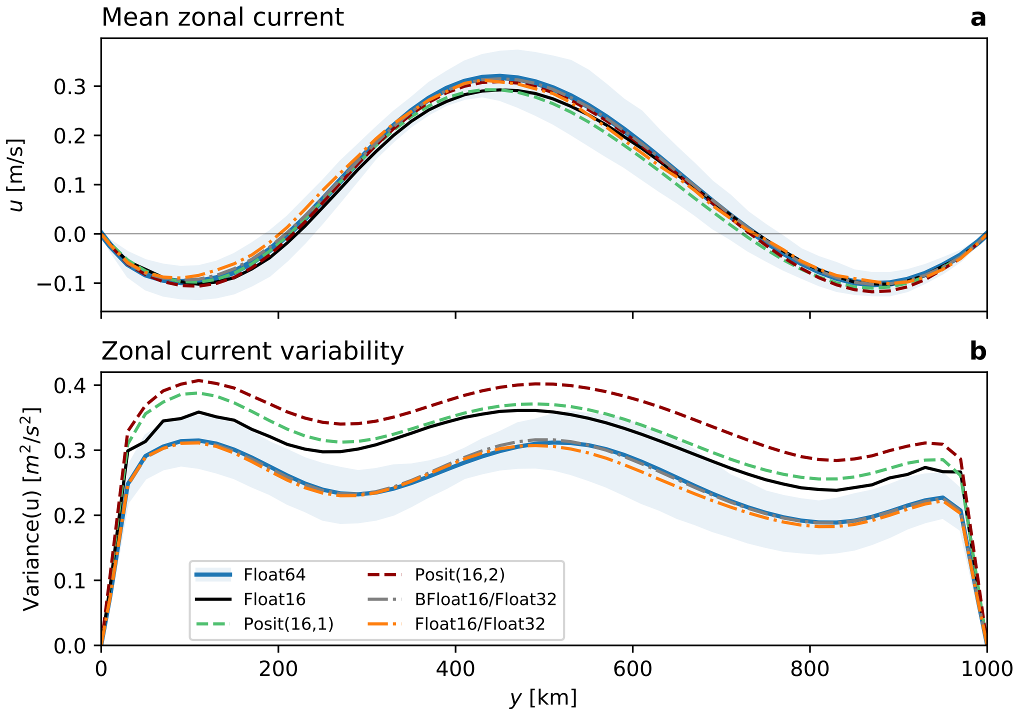

Figure 6. Climatology and variability of the zonal current in the medium-resolution simulations. (a) Zonally-averaged zonal current u as a function of the meridional coordinate y. (b) Zonal variance of the zonal current as a function of y. The shaded area denotes the interquartile temporal variability around the (a) mean and (b) variance of reference simulation with Float64.

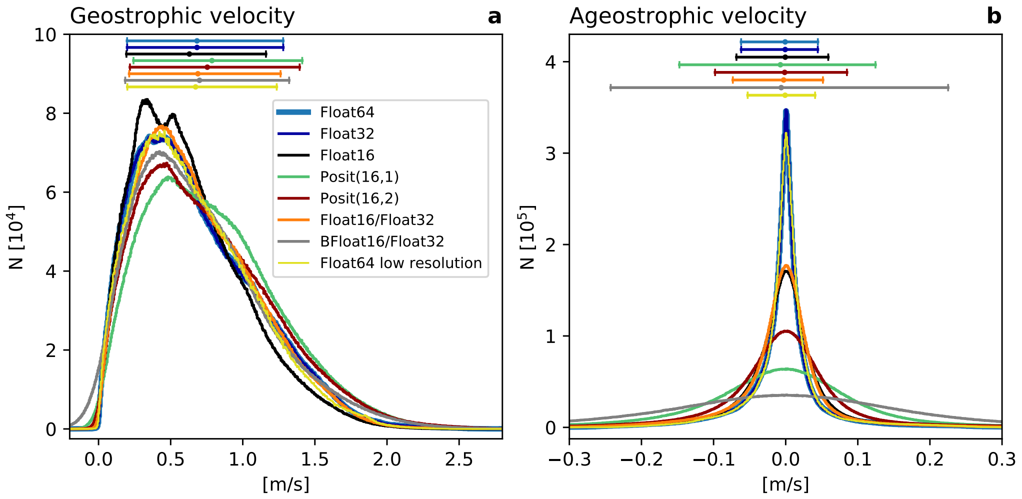

Figure 7. Geostrophic balance as simulated with different number formats. (a) Histograms of flow-parallel components of geostrophic velocity. (b) as (a) but for the ageostrophic velocities. Horizontal bars denote the mean, 10th and 90th-percentile in respective colours.

$\tilde{u}_g$ and $\tilde{u}_{ag}$ via

$$\tilde{u}_g = \frac{\mathbf{u}_g \cdot \mathbf{u}}{\|\mathbf{u}\|}, \quad \tilde{u}_{ag} = \frac{\mathbf{u}_{ag} \cdot \mathbf{u}}{\|\mathbf{u}\|}. \tag{18}$$

The geostrophic velocities in the shallow water simulations can reach up to 2 m/s, are hardly negative (i.e. against the flow) and have a mean of about 0.7 m/s (Fig. 7a). This behaviour is well simulated with 16-bit number formats, although posits increase the strength of geostrophic velocities slightly. Ageostrophic velocity components are found to be isotropic, and are oriented equally frequent with and against the prevailing flow, but rarely exceed ±0.1m/s and are therefore comparably small as expected in geostrophically balanced turbulence. Ageostrophic velocities can be seen as a measure of the physical instabilities in the flow field and their variance is indeed increased when simulated with 16-bit number formats. Float16 shows clearly fewer ageostrophic velocities around 0, pointing towards an increased number of simulated instabilities. Posits have an even further increased number of ageostrophic velocities, and especially Posit(16,1) increases the variance of those by more than factor of two. It is unclear where in the model integration rounding errors of 16-bit arithmetic trigger instabilities that lead to the observed increase in ageostrophy. We conclude that although the geostrophic balance in the simulations is maintained, rounding errors lead, likely due to an increase in ageostrophy, to a higher variability in the flow field.

As 16-bit arithmetics have no significant impact on the climatological mean state, histograms of prognostic variables are also not changed (Fig. 8a and b). However, the tendencies are increased by orders of magnitude with 16-bit arithmetics (Fig. 8d and e), as rounding errors cause gravity waves to radiate away from eddies (Fig. 8f). Gravity waves are identified from the tendency of sea surface height. Comparing their propagation to the location of anomalous sea surface height, which is used as a proxy for eddies, we assume that rounding errors in regions of high eddy activity lead to instabilities that propagate away in the form of gravity waves. These gravity waves are not present in Float64 simulations (Fig. 8c) and tend to have only a small impact on quasi-gestrophic dynamics, as they act on different time and length scales. It is unclear whether these gravity waves cause the observed ageostrophic velocities.

The tendencies are about 4 orders of magnitude smaller than the prognostic variables. This poses a problem for number formats with a machine epsilon, measured as decimal precision, significantly lower than 4 decimal places (Table 1). Float16 has a ma-

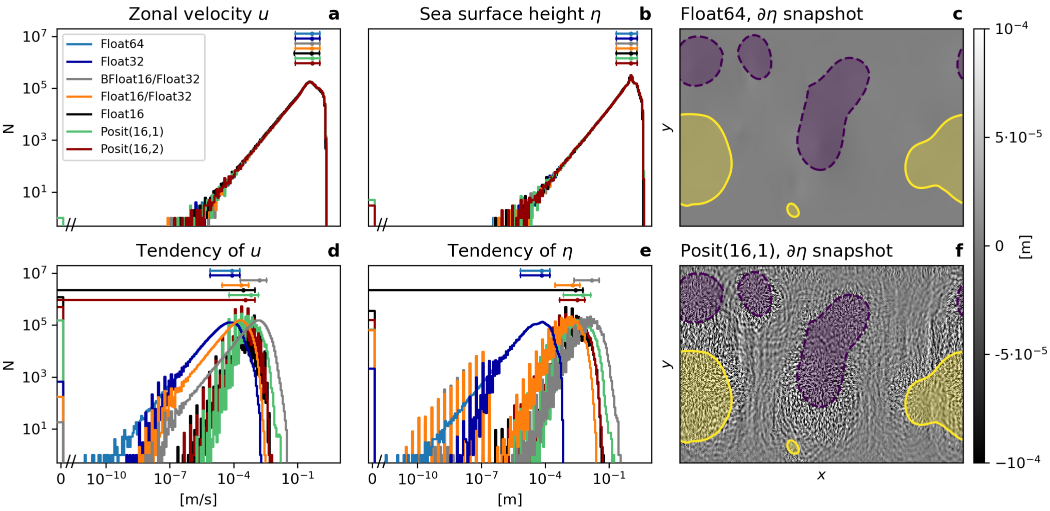

Figure 8. Histograms of the numeric values of the prognostic variables (a) zonal velocity u, (b) sea surface height η, and the respective tendencies of (d) u and (e) η, simulated with different 16, 32 and 64-bit number formats. Mean, 10th and 90th percentile are shown above the histograms in respective colors. Snapshots of the tendencies of η simulated with (c) Float64 and (f) Posit(16,1). Results are similar for other 16-bit formats (not shown here). Areas of sea surface height anomalies exceeding ±1.4 m are shown in purple (negativ) and yellow (positive). Note the break on the x-axis close to zero in (a,b,d) and (e).

chine epsilon of 3.7, which is presumably close to the lower limit beyond which the addition of tendencies will be round back. The BFloat16 number format has a machine error of 2.8, which explains why no change from initial conditions in the shallow water system can be simulated with BFloat16.

3.2.3 Mixed precision arithmetic in the shallow water model

In the previous simulations the entire shallow water simulation was performed with the specified number format. As the addition of tendencies to the prognostic variables was identified as a key calculation that is error-prone, we investigate now the benefits of mixed precision arithmetic, where Float32 is used for the prognostic variables but the tendencies are computed with either Float16 or BFloat16, two number formats that have the lowest decimal precision for numbers around 1. The prognostic variables are now reduced to Float16 or BFloat16 before calculations of the right-hand side and every term of the tendencies is converted back before addition to the prognostic variables. Using subscripts 16 and 32 to denote variables held at 16 and 32-bit precision, respectively, and let Float32() be the conversion function then the continuity Eq. 9b becomes

$$\frac{\partial \eta_{32}}{\partial t} = -\operatorname{Float32}(\partial_x(u_{16}h_{16}) + \partial_y(v_{16}h_{16})) \tag{19}$$

and similar for u and v in Eq. 9a.

Snapshots of tracer concentration reveal well simulated geostrophic turbulence (Fig. 4e and f) with Float16/Float32 or BFloat16/Float32 and instabilites at fronts or in filaments are visibly reduced compared to pure 16-bit arithmetic. The forecast error is strongly reduced once the prognostic variables are kept as Float32 (Fig. 5a), supporting the hypothesis that the addition of tendencies to the prognostic variables is a key computation with low rounding error-tolerance. Despite BFloat16 not being suitable for shallow water simulations when applied to all computations, mixing BFloat16 with Float32 arithmetic yields a similar error growth to posits, which is well below the discretization error. Mean state or variability are virtually identical for both mixed precision cases (Fig. 6) compared to the Float64 reference. The geostrophic balance is largely unaffected, but ageostrophic velocities increase in variance, especially for BFloat16 (Fig. 7). Gravity waves are similarly present for mixed precision although weaker for tendencies computed with Float16 (Fig. 8d) and, as discussed, they tend to not interact with the geostrophic time and length scales. Although the results show that Float16 is generally a preferrable number format over BFloat16 for the applications presented here, we acknowledge that the conversion between Float32 and Float16 will come with some computational cost. In contrast, the conversion between BFloat16 and Float32 is computationally very cheap as both formats have the same number of exponent bits. Removing signifcant bits, potentially applying rounding, and padding trailing zeros, are the only operations for this conversion. Following the results here, mixing 16 and 32 bit precision is found to be a possible solution to circumvent spurious behaviour due to 16-bit floating-point arithmetics. Performance benefits are still possible as most calculations are performed with 16 bit, with key computations in 32 bit to reduce the overall error. Depending on the application, the conversions between number formats are assumed to be of negligible cost. This is an attractive solution as hardware-accelerated 16-bit floating-point arithmetic is already available on graphic or tensor processing units and implementations therefore do not rely on the development of future computing hardware, as it is the case for posits.

3.2.4 Reduced precision communication for the shallow water model

A standard method to parallelise simulations is the distributed-memory parallelism via Message Passing Interface (MPI). We emulate MPI-like communication in the shallow water model with the copying of boundary values between the right and left boundary (periodic boundary conditions). Although the shallow water model does not run in parallel, reducing the precision in the copying of boundary values introduces an equivalent error as if reduced precision MPI was used to communicate between subdomains. Reduced precision is applied for the communication of the prognostic variables at every Runge-Kutta substep.

Regarding snapshots of tracer concentration simulated with reduced precision communication show a negligible error for Float16 and posits (Fig. 9). The error is largest at fronts and not concentrated around the boundaries. Encoding the communication with BFloat16 introduces a larger error than for the other 16-bit formats as the decimal precision is with 2.8 clearly lower (Table 1) for the range of values occuring within the prognostic variables (Fig. 8a and b). The errors are quantified by the RMSE of surface height η as before and are up to about two orders of magnitude smaller than the errors that result from 16-bit arithmetic. As even the worst 16-bit communication format, BFloat16, has a smaller error than the best mixed precision formats, Float16 with Float32, we extend the short-term forecast experiments to include two 8-bit formats, Posit(8,0) and Float8

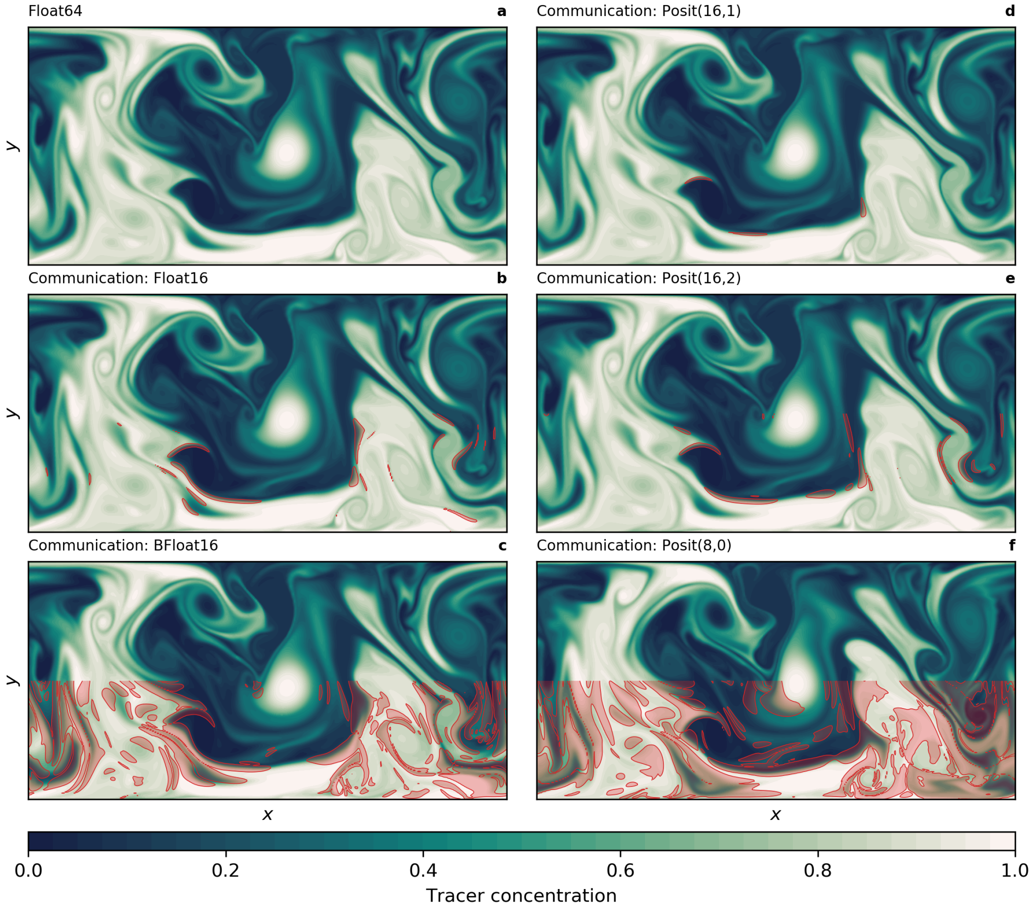

Figure 9. Snapshot of tracer concentration simulated by the shallow water model using reduced precision communication. The communication of boundary values occurs at every time step for the prognostic variables. Float64 was used for all calculations. Areas where the absolute error exceeds 0.05 are shaded in red only in the lower half of the domain. The tracer was injected uniformly in the lower half of the domain 50 days before. This simulation was run with the high-resolution configuration.

(see Table 1 for a description). Both formats are found to be suitable for reduced precision communication here and do not introduce an error that is larger than the discretization error. Having said that, Float8 communication introduces an error that is comparably large initially but growths only linearly in the first 50 days of the simulation, which is in contrast to the exponential error growth observed for 16-bit arithmetic.

4 Conclusion and Discussion

Running computationally demanding algorithms at 16-bit could reduce the wall-clock time for weather and climate simulations on future high performance computing architecture. Using a software emulator, we have tested a number of options for 16-bit arithmetic for weather and climate applications. We achieved the best results with 16-bit posits (with either 1 or 2 exponent bits) which appear very promising for application in high performance computing for Earth System modelling. Float16 can be used to perform forecasts with the shallow water model while the application of BFloat16 or integer arithmetic was not successful. In general, 16-bit arithmetics were not found to alter the climatological mean state or the large-scale dynamics. However, variability and ageostrophic velocities were increased, such that second and higher-order statistics should undergo testing to assess the models reliability. Depending on the application, an increased variability does not necessarily deteriorate the model, especially for more realistic model set-ups as considered here.

To enable shallow water simulations with 16-bit arithmetic required the rearrangement of a couple of terms but no major revisions of the model code or algorithms. Given that only floats are currently hardware-supported, we investigated mixed precision approaches, where the prognostic variables are kept at 32 bit and the tendencies are computed in 16 bit and found that the impact of rounding errors can be reduced significantly. We also showed that numerical precision for communication between compute nodes can be greatly reduced down to 16 or even 8-bit without introducing a large error. Reduced precision communication was not found to have a significant impact on either mean state, variability, geostrophy or tendencies.

In this study, we perform model forecasts with a *perfect model.* Any form of model error is ignored, as the Float64 reference is exactly the same model as its reduced precision counterparts and any form of initial condition error is also ignored. Only discretisation errors are estimated by lowering the spatial resolution by a factor of 2. This is not a realistic set-up for weather or climate models. Real models include many other sources of forecast error and it is likely that the contributions of rounding errors from 16-bit arithmetic would be dwarfed by errors in initial conditions or discretisation errors in many applications.

The numerical discretisation that was used for the shallow water model in this paper, with an explicit time stepping scheme and 2nd order centred finite differences, is common to solve the equations of motion in fluid dynamics. However, various different methods of discretisation exist, including spectral methods, finite element/volume and implicit time stepping. The requirements on reduced precision will differ for the different algorithms and some methods may be more sensitive to rounding errors compared to the techniques that were studied here.

As there is no hardware available for posit arithmetic that we could have used for performance testing, we cannot draw any conclusion about the performance of posit arithmetic operations in comparison to Float16 or the other formats. Until progress is made on hardware implementations for posits, the results here suggest that also 16-bit float arithmetic can succesfully be used for parts of complex weather and climate models with the potential for acceleration on graphic and tensor processing units. It is therefore recommended to adapt a type-flexible programming paradigm, ideally in a language that supports portability, with algorithms written to reduce the dynamic range of arithmetic results. Hardware progress on central, graphic or tensor processing units, with various numbers formats supported, can subsequently be utilised to accelerate weather and climate simulations.

Appendix A Emulating posit numbers in the Julia language

The SoftPosit.jl emulator defines conversion and arithmetic operations with posits. Julia's programming paradigms of *multiple-dispatch* and *type-stability* facilitate the use of arbitrary number formats without the need to rewrite an algorithm. As this is an essential feature of Julia and extensively made use of in this study, we briefly outline the benefits of Julia by computing the harmonic sum with various number types as an example.

```julia
function harmonic_sum(::Type{T},steps::Int=2000) where T

    s = zero(T)
    o = one(T)

    for i in 1:steps

        s_old = s
        s += o/T(i)

        if s == s_old    # check for convergence
            println(Float64(s),i)
            break
        end
    end
end
```

Figure A1. A type-flexible harmonic sum function in the Julia language.

Executing the function `harmonic_sum` for the first time with a type `T1` as the first argument, triggers Julia's *just-in-time* compiler (Fig. A1). The function is type-stable, as the types of all variables are declared. 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. A2). 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 round 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 3.2.3. Using Posit(16,1), the sum only converges after 1024 terms, due to the higher decimal precision of posits between 1 and 10.

```
julia> using SoftPosit
julia> using BFloat16s
julia> harmonic_sum(Float16)
(7.0859375, 513)

julia> harmonic_sum(BFloat16)
(5.0625, 65)

julia> harmonic_sum(Posit16)
(7.77734375, 1024)
```

Figure A2. Harmonic sum example use of the posit emulator *SoftPosit.jl* in the Julia shell. `Posit16` is the Posit(16,1) standard.

We implement this type-flexible programming paradigm in the numerical integration of the Lorenz 1963 system (section 3.1) and the shallow water model (section 3.2), which allows various number types to be used interchangeably.

Acknowledgments

Milan Klöwer and Tim N. Palmer gratefully acknowledge funding by the European Research Council under grant number 741112 *An Information Theoretic Approach to Improving the Reliability of Weather and Climate Simulations*. Milan Klöwer is also funded by NERC grant number NE/L002612/1. Peter D. Düben gratefully acknowledges funding from the Royal Society for his University Research Fellowship as well as funding from the ESIWACE2 project. ESIWACE2 has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement 823988.

References

Arakawa, A., & Hsu, Y.-J. G. (1990, October). Energy Conserving and Potential-Enstrophy Dissipating Schemes for the Shallow Water Equations. *Monthly Weather Review*, *118*(10), 1960–1969. doi: 10.1175/1520-0493(1990)118⟨1960:ECAPED⟩2.0.CO;2

Arakawa, A., & Lamb, V. R. (1977, January). Computational Design of the Basic Dynamical Processes of the UCLA General Circulation Model. In J. Chang (Ed.), *Methods in Computational Physics: Advances in Re-*

search and Applications (Vol. 17, pp. 173–265). Elsevier. doi: 10.1016/B978-0-12-460817-7.50009-4

Arbic, B. K., & Scott, R. B. (2008, January). On Quadratic Bottom Drag, Geostrophic Turbulence, and Oceanic Mesoscale Eddies. *Journal of Physical Oceanography*, *38*(1), 84–103. doi: 10.1175/2007JPO3653.1

Bezanson, J., Edelman, A., Karpinski, S., & Shah, V. B. (2017, January). Julia: A Fresh Approach to Numerical Computing. *SIAM Review*, *59*(1), 65–98. doi: 10.1137/141000671

Burgess, N., Milanovic, J., Stephens, N., Monachopoulos, K., & Mansell, D. (2019, June). Bfloat16 Processing for Neural Networks. In *2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)* (pp. 88–91). doi: 10.1109/ARITH.2019.00022

Butcher, J. C. (2016). *Numerical Methods for Ordinary Differential Equations* (3edition ed.). Chichester, West Sussex, United Kingdom: Wiley.

Chaurasiya, R., Gustafson, J., Shrestha, R., Neudorfer, J., Nambiar, S., Niyogi, K., … Leupers, R. (2018, October). Parameterized Posit Arithmetic Hardware Generator. In *2018 IEEE 36th International Conference on Computer Design (ICCD)* (pp. 334–341). Orlando, FL, USA: IEEE. doi: 10.1109/ICCD.2018.00057

Chen, J., Al-Ars, Z., & Hofstee, H. P. (2018). A matrix-multiply unit for posits in reconfigurable logic leveraging (open)CAPI. In *Proceedings of the Conference for Next Generation Arithmetic on - CoNGA '18* (pp. 1–5). Singapore, Singapore: ACM Press. doi: 10.1145/3190339.3190340

Choo, H., Muhammad, K., & Roy, K. (2003). Two's complement computation sharing multiplier and its applications to high performance DFE. *IEEE Transactions on Signal Processing*, *51*, 458–469.

Dawson, A., & Düben, P. D. (2017, June). Rpe v5: An emulator for reduced floating-point precision in large numerical simulations. *Geoscientific Model Development*, *10*(6), 2221–2230. doi: 10.5194/gmd-10-2221-2017

Düben, P. D. (2018, November). A New Number Format for Ensemble Simulations. *Journal of Advances in Modeling Earth Systems*, *10*(11), 2983–2991. doi: 10.1029/2018MS001420

Düben, P. D., & Palmer, T. N. (2014, July). Benchmark Tests for Numerical

Weather Forecasts on Inexact Hardware. *Monthly Weather Review*, *142*(10), 3809–3829. doi: 10.1175/MWR-D-14-00110.1

Gill, A. E. (1982). *Atmosphere-ocean dynamics* (No. 30). San Diego: Acad. Press. (OCLC: 249294465)

Glaser, F., Mach, S., Rahimi, A., Gürkaynak, F. K., Huang, Q., & Benini, L. (2017, December). An 826 MOPS, 210 uW/MHz Unum ALU in 65 nm. *arXiv:1712.01021 [cs]*.

Griffies, S. M., & Hallberg, R. W. (2000). Biharmonic Friction with a Smagorinsky-Like Viscosity for Use in Large-Scale Eddy-Permitting Ocean Models. *MONTHLY WEATHER REVIEW*, *128*, 12.

Gupta, S., Agrawal, A., Gopalakrishnan, K., & Narayanan, P. (2015, February). Deep Learning with Limited Numerical Precision. *arXiv:1502.02551 [cs, stat]*.

Gustafson, J. (2017). *Posit Arithmetic.*

Gustafson, J. L., & Yonemoto, I. (2017). Beating Floating Point at its Own Game: Posit Arithmetic. *Supercomputing Frontiers and Innovations*, *4*(2), 16.

Hatfield, S., Düben, P., Chantry, M., Kondo, K., Miyoshi, T., & Palmer, T. (2018, September). Choosing the Optimal Numerical Precision for Data Assimilation in the Presence of Model Error. *Journal of Advances in Modeling Earth Systems*, *10*(9), 2177–2191. doi: 10.1029/2018MS001341

Jeffress, S., Düben, P., & Palmer, T. (2017, September). Bitwise efficiency in chaotic models. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, *473*(2205), 20170144. doi: 10.1098/rspa.2017.0144

Jouppi, N. P., Young, C., Patil, N., & Patterson, D. (2018, August). A domain-specific architecture for deep neural networks. *Communications of the ACM*, *61*(9), 50–59. doi: 10.1145/3154484

Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., ... Yoon, D. H. (2017, June). In-Datacenter Performance Analysis of a Tensor Processing Unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture* (pp. 1–12). Toronto, ON, Canada: Association for Computing Machinery. doi: 10.1145/3079856.3080246

Kalamkar, D., Mudigere, D., Mellempudi, N., Das, D., Banerjee, K., Avancha, S., ... Dubey, P. (2019, June). A Study of BFLOAT16 for Deep Learning Training. *arXiv:1905.12322 [cs, stat]*.

Klöwer, M., Düben, P. D., & Palmer, T. N. (2019). Posits as an alternative to floats for weather and climate models. In *Proceedings of the Conference for Next Generation Arithmetic 2019 on - CoNGA'19* (pp. 1–8). Singapore, Singapore: ACM Press. doi: 10.1145/3316279.3316281

Klöwer, M., & Giordano, M. (2019, December). *SoftPosit.jl - A posit arithmetic emulator.* Zenodo. doi: 10.5281/zenodo.3590291

Kwasniok, F. (2014, June). Enhanced regime predictability in atmospheric low-order models due to stochastic forcing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, *372*(2018), 20130286. doi: 10.1098/rsta.2013.0286

Langroudi, H. F., Carmichael, Z., & Kudithipudi, D. (2019, July). Deep Learning Training on the Edge with Low-Precision Posits. *arXiv:1907.13216 [cs, stat]*.

Leong, S. H. (2020, March). *SoftPosit.* Zenodo. (Language: eng) doi: 10.5281/zenodo.3709035

Lorenz, E. N. (1963, March). Deterministic Nonperiodic Flow. *Journal of the Atmospheric Sciences*, *20*(2), 130–141. doi: 10.1175/1520-0469(1963)020⟨0130:DNF⟩2.0.CO;2

Markidis, S., Chien, S. W. D., Laure, E., Peng, I. B., & Vetter, J. S. (2018, May). NVIDIA Tensor Core Programmability, Performance Precision. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)* (pp. 522–531). doi: 10.1109/IPDPSW.2018.00091

Palmer, T. (2015, October). Modelling: Build imprecise supercomputers. *Nature News*, *526*(7571), 32. doi: 10.1038/526032a

Palmer, T. N. (2012, April). Towards the probabilistic Earth-system simulator: A vision for the future of climate and weather prediction. *Quarterly Journal of the Royal Meteorological Society*, *138*(665), 841–861. doi: 10.1002/qj.1923

Rüdisühli, S., Walser, A., & Fuhrer, O. (2013). COSMO in single precision. *Cosmo Newsletter*(14), 5–1.

Russell, F. P., Düben, P. D., Niu, X., Luk, W., & Palmer, T. (2017, December). Exploiting the chaotic behaviour of atmospheric models with reconfigurable architectures. *Computer Physics Communications*, *221*, 160–173. doi: 10.1016/j.cpc.2017.08.011

Smolarkiewicz, P. K., & Pudykiewicz, J. A. (1992, November). A Class of Semi-

Lagrangian Approximations for Fluids. *Journal of the Atmospheric Sciences*, *49*(22), 2082–2096. doi: 10.1175/1520-0469(1992)049⟨2082:ACOSLA⟩2.0.CO;2

Tantet, A., Lucarini, V., & Dijkstra, H. A. (2018, February). Resonances in a Chaotic Attractor Crisis of the Lorenz Flow. *Journal of Statistical Physics*, *170*(3), 584–616. doi: 10.1007/s10955-017-1938-0

Thornes, T., Düben, P., & Palmer, T. (2017, January). On the use of scale-dependent precision in Earth System modelling: Scale-Dependent Precision in Earth System Modelling. *Quarterly Journal of the Royal Meteorological Society*, *143*(703), 897–908. doi: 10.1002/qj.2974

Tintó Prims, O., Acosta, M. C., Moore, A. M., Castrillo, M., Serradell, K., Cortés, A., & Doblas-Reyes, F. J. (2019, July). How to use mixed precision in ocean models: Exploring a potential reduction of numerical precision in NEMO 4.0 and ROMS 3.6. *Geoscientific Model Development*, *12*(7), 3135–3148. doi: https://doi.org/10.5194/gmd-12-3135-2019

Vallis, G. K. (2006). *Atmospheric and Oceanic Fluid Dynamics.* Cambridge University Press.

van Dam, L., Peltenburg, J., Al-Ars, Z., & Hofstee, H. P. (2019, March). An Accelerator for Posit Arithmetic Targeting Posit Level 1 BLAS Routines and Pair-HMM. In *Proceedings of the Conference for Next Generation Arithmetic 2019* (pp. 1–10). Singapore, Singapore: Association for Computing Machinery. doi: 10.1145/3316279.3316284

Váňa, F., Düben, P., Lang, S., Palmer, T., Leutbecher, M., Salmond, D., & Carver, G. (2017, February). Single Precision in Weather Forecasting Models: An Evaluation with the IFS. *Monthly Weather Review*, *145*(2), 495–502. doi: 10.1175/MWR-D-16-0228.1