Weather and climate models in 16bit: Posit, floating-point or mixed precision arithmetic?

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 entirely computed with 16bit arithmetic.
- Errors caused by 16bit floating-point arithmetic are strongly reduced with critical computations in 32bit, which can be implemented on present-day hardware.
- 16 or even 8bit communication of boundary values, preferably encoded as posit numbers, introduces negligible errors, providing perspectives for reduced data communication for weather and climate models computed in parallel.

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

Abstract

The need for high precision calculations with 64bit floating-point numbers for weather and climate models has been questioned. Lower precision numbers can accelerate simulations and are increasingly supported by modern computing architecture. Posit numbers, a recently proposed alternative to floating-point numbers, claim to have smaller arithmetic rounding errors in many applications. As a standardized posit processor does not exist yet, we emulate posit arithmetic on a conventional processor. Benefits of posits compared to floats at 16bit are presented in the Lorenz system and with a shallow water model. Forecasts based on posits are clearly more accurate than floats in shallow water simulations. Even the rounding error of 16bit floats is not larger than the discretization error, that results from simulations run at half the spatial resolution. No 16bit format was found to have a significant impact on the climatological mean state, however, the variability increased up to 30%. Instabilities and gravity waves, triggered by rounding errors of 16bit formats, were found to not alter the dynamics of the simulated flow significantly. Mixing 16bit arithmetic with 32bit for critical computations strongly reduces errors and is promising for present-day float-based hardware. Reduced precision communication of boundary values with 16 or 8bit encoded as floats or posits introduces negligible errors, presenting a perspective for reduced data communication within a computer cluster. The results promote the potential of 16bit 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 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 with the IEEE 754 standard on 64 bit double precision floating-point numbers (Float64), 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 32bit 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).

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 16bit half precision floats (Float16), is an active field of research, but remains challenging (Düben et al., 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 the widely supported Float32 and Float64. 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)).

The recent boom of deep learning techniques, that require low numerical precision but high computational performance, will influence hardware development to offer more flexibility for the use of reduced precision number formats. Graphic processing units started to support Float16 for increased performance (Markidis et al., 2018). Google's tensor processing units (TPU, N. P. Jouppi et al. (2017); N. Jouppi et al. (2018)) support the 16bit BFloat16 format, a truncated version of Float32, as deep learning training was found to be succesful despite the degradation in precision with 16bit arithmetic (Kalamkar et al., 2019; Burgess et al., 2019).

Posit numbers are a recently proposed alternative to floats 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. This study therefore focuses on posit arithmetic as an alternative to floating-point arithmetic at the appealing size of 16 bit for weather and climate models. We use a Julia-based emulator on a conventional CPU, as standardized posit hardware is not yet available. Posit research currently focuses on hardware implementations (van Dam et al., 2019; Chen et al., 2018; Chaurasiya et al., 2018; Glaser et al., 2017).

The study is structured as follows: Section 2 introduces the posit number format, its emulation in the Julia language and the concept of decimal precision. We analyse the dynamics of a chaotic model at low complexity with posit arithmetic using the Lorenz 1963 system in section 3. In section 4 we evaluate posit arithmetic in the shallow water equations, investigate a mixed-precision approach and analyse reduced precision communication. Section 5 discusses the results and summarises the conclusions.

2 Posit numbers

2.1 The posit number format

The IEEE standard on floating-point arithmetic defines how floats encode a real number in terms of a sign bit, and a fixed number of exponent and significant bits (16bit half-precision floats have 1 sign, 5 exponent and 10 significant bits). Consequently, they have a constant number of significant digits throughout their range of representable numbers. This is in contrast to posit numbers, which arise from a projection of the real axis onto a circle (Fig. 1), with only one bitpattern 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 $useed^{\pm 2}$, $useed^{\pm 3}$, $useed^{\pm 4}$, 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

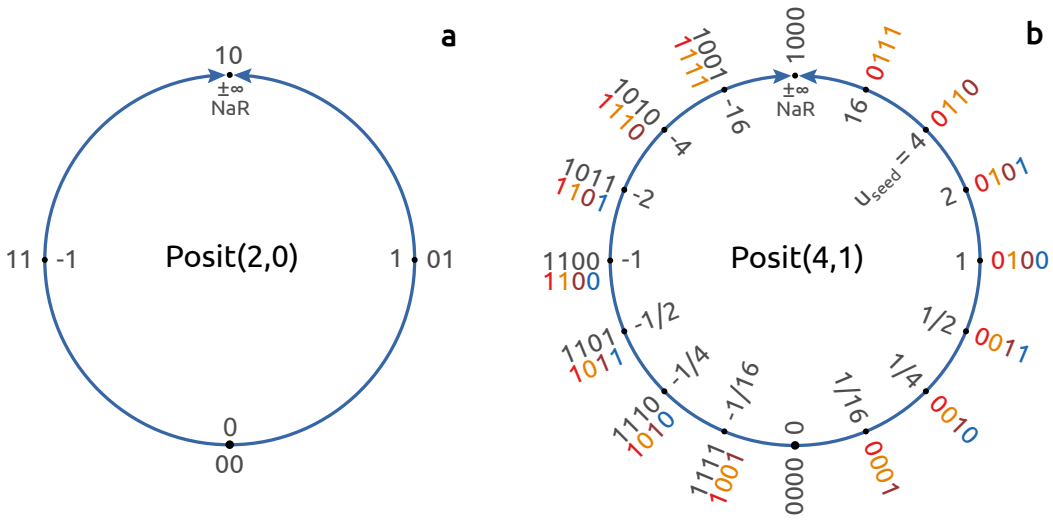

Figure 1. Two posit number formats obtained by projecting the real axis onto a circle. (a) 2bit Posit(2,0) and (b) 4bit Posit(4,1). The bit patterns are marked on the outside and the respective values on the inside of each circle. Bit patterns of negative numbers (black) have to be converted to their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

are converted first to their two's complement, see Eq. 3)

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

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. 1. 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. $useed = 4$)

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

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,

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

After the conversion to the two's complement, the bits are interpreted in the same way as in Eq. 2.

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 (J. Gustafson, 2017). 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 1), and only poses a noticable issue for Float16 and Float8.

The posit number framework also highly recommends *quires*, an additional register on hardware to store intermediate results. Fused operations like *multiply-add* can therefore be executed with a single rounding error without the rounding of intermediate results. The quire concept could also be applied to floating-point arithmetic, but is technically difficult to implement on hardware as the required registers would need to be much larger in size. For fair comparison we do not take quires into account. The posit number format is explained in more detail in J. Gustafson (2017).

2.2 Emulating posit numbers in the Julia language

In order to use posits on a conventional CPU 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. This 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 2. A type-flexible harmonic sum function in the Julia language.

Executing the function `harmonic_sum` for the first time with a type `T` as the first argument, triggers Julia's *just-in-time* compiler. 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 aribitrary 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`.

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

The harmonic sum converges after 513 elements when using Float16. 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 4.3. Using Posit(16,1), the sum only converges after 1024 terms, due to the higher decimal precision of posits between 1 and 10.

We implement this type-flexible programming paradigm in the numerical integration of the Lorenz equations (section 3) and the shallow water model (section 4), which allows various number types to be used interchangeably.

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	$1.0 \cdot 10^{-45}$	$3.4 \cdot 10^{38}$	2.8	0.4
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
Float8	8	3	$1.5 \cdot 10^{-2}$	15.5	1.9	12.5

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).

2.3 Decimal precision

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

$$\text{decimal precision} = -\log_{10}|\log_{10}(\frac{x_{\text{repr}}}{x_{\text{exact}}})| \tag{4}$$

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.

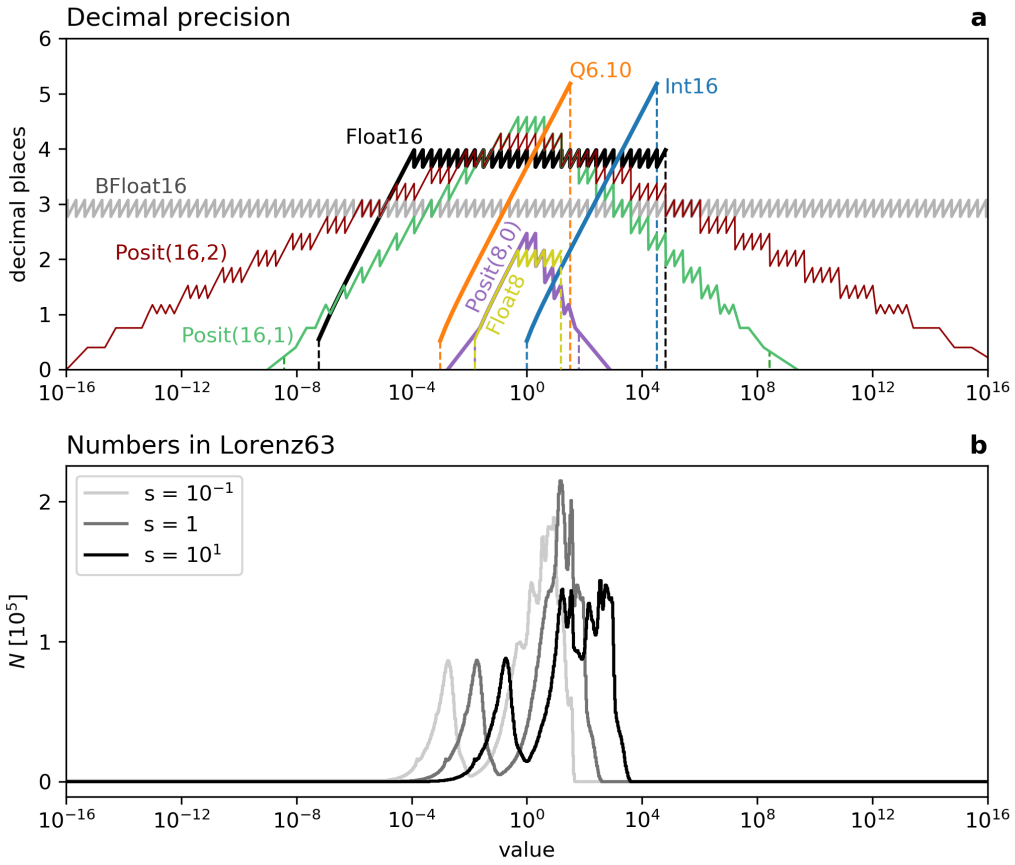

Figure 4. (a) Decimal precision of various number formats. Dashed vertical lines indicate the range of representable numbers for each format. Float64, Float32 and Posit32 are beyond the axes limits. (b) Histogram of results of all arithmetic operations in the rescaled Lorenz system, considering absolute values.

Fig. 4a compares the worst-case decimal precision for various 16 and 8bit floats and posits, as well as 16bit 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 for the subnormal numbers towards the smallest representable number *minpos*. 16bit 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 on the order of 10^4 as well as 10^{-4}. 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. Due to the no overflow/no underflow-rounding mode, the decimal precision is slightly above zero outside the dynamic range.

The decimal precision of 16bit 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. Flexibility regarding the dynamic range can therefore be achieved with integer arithmetic if fixed point numbers are used (Russell et al., 2017). However, 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.

3 Lorenz 1963 system

3.1 Methods

The Lorenz system (L63, (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

$$\frac{dx}{dt} = \sigma(y - x) \tag{5a}$$

$$\frac{dy}{dt} = x(\rho - z) - y \tag{5b}$$

$$\frac{dz}{dt} = xy - \beta z \tag{5c}$$

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

To find the optimal number representation to solve Eq. 5 requires considering the dynamic range of all intermediate calculations. It is possible to influence this dynamic range using a *rescaling* of the equations via a simple multiplication of the variables with a constant rescaling factor s. The rescaled variables are denoted as $\tilde{x} = sx$, and similarly for $\tilde{y}, \tilde{z}$. Fig. 4b shows histograms for all numbers that are used to solve the Lorenz system (including intermediate calculations). A comparison to the decimal precision in Fig. 4a 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

$$\tilde{x}^{n+1} = \tilde{x}^n + RK_x \left(\tilde{y}^n - \tilde{x}^n\right) \tag{6a}$$

$$\tilde{y}^{n+1} = \tilde{y}^n + RK_y \left(\tilde{x}^n(\rho - \frac{\tilde{z}^n}{s}) - \tilde{y}^n\right) \tag{6b}$$

$$\tilde{z}^{n+1} = \tilde{z}^n + RK_z \left(\tilde{x}^n \frac{\tilde{y}^n}{s} - \beta \tilde{z}^n\right) \tag{6c}$$

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. 6 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. 4b), 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 Results

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. 5a). This attractor is *strange*, i.e. its geometric structure cannot be described in two dimensions, but is of fractal nature. While points on the model trajectory 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. 5a and b). However, when using Posit(16,1) and a rescaling factor of $s = 0.1$ the representation of the attractor is improved significantly (Fig. 5c). The results for posits looks 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. This allows the Lorenz system to be integrated with BFloat16, but with a poor representation of the fractal attractor (Fig. 5d). The solution of the Lorenz system with 16bit integers presents a similarly simple Lorenz attractor (Klöwer et al., 2019).

4 Shallow water model

4.1 Methods

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,

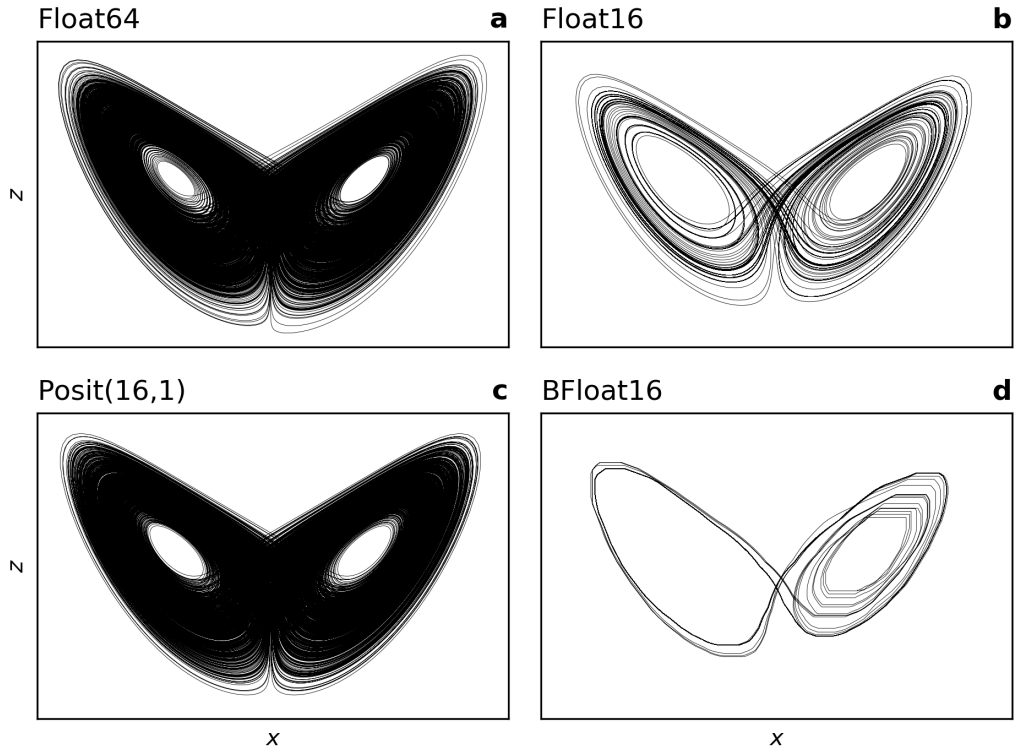

Figure 5. The Lorenz attractor computed with different number formats. The scaling of the Lorenz equations (Eq. 6) is (a,b,d) $s = 1$, (c) $s = 0.1$. All trajectories are integrated from the same initial conditions for 100,000 time-steps with spacing $\Delta t = 0.008$.

1982; Vallis, 2006). The shallow water equations for the prognostic variables velocity $\mathbf{u} = (u, v)$ and sea surface elevation η are

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} + f\hat{\mathbf{z}} \times \mathbf{u} = -g\nabla\eta + \mathbf{D} + \mathbf{F} \tag{7a}$$

$$\frac{\partial \eta}{\partial t} + \nabla \cdot (\mathbf{u}h) = 0. \tag{7b}$$

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. 7b is the shallow water-variant of the continuity equation, ensuring conservation of mass. The domain is a zonally periodic rectangular channel of size 2000 km ×1000 km, with a meridional mountain ridge in the middle of the domain. A more detailed description of the shallow water model, introducing the remaining parameters and variables in Eq. 7, is presented in Appendix A. The shallow water equations are discretised using 2nd order centred finite differences on an Arakawa C-grid (Arakawa & Lamb, 1977) with a grid spacing of $\Delta = 20$ km (100x50 grid points) and the Runge-Kutta fourth order method (Butcher, 2016) is used for time integration. The advection terms are discretised using an energy and enstrophy conserving scheme (Arakawa & Hsu, 1990).

To also 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

$$\frac{\partial q}{\partial t} + \mathbf{u} \cdot \nabla q = 0. \tag{8}$$

We discretise Eq. 8 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 in general in between grid nodes an interpolation is required to find the concentration at the departure point. The discretisation of Eq. 8 is therefore turned into an interpolation problem.

For 16bit arithmetic it is essential to rescale the shallow water equations to avoid calculations with very large or very small results, as the dynamic range of representable numbers is limited (Fig. 4a 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. 7 and 8 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 can therefore retain their physical units in the discretised numerical model. 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. 7b, for example, is discretised with an explicit time integration method as

$$\eta^{n+1} = \eta^n + RK_\eta \left(-\tilde{\nabla} \cdot (\mathbf{u}h)^n\right) \tag{9}$$

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

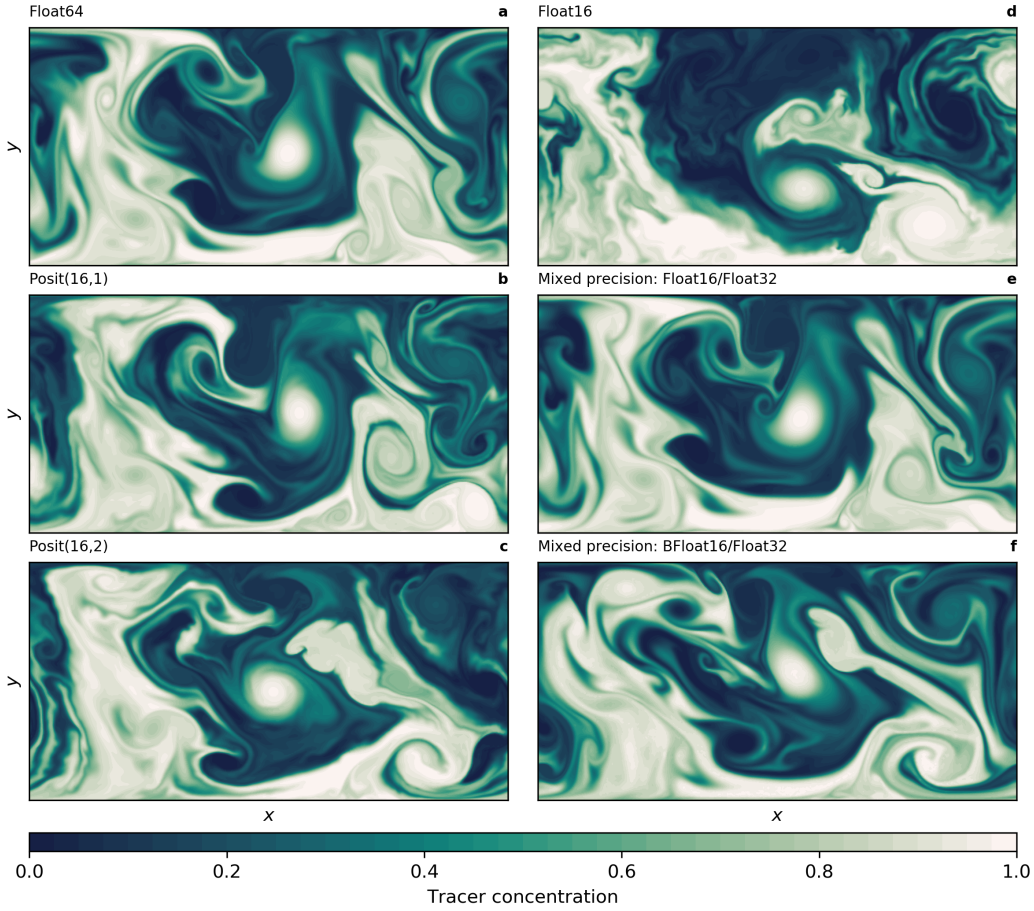

Figure 6. Snapshot of tracer concentration simulated by the shallow water model using different 16bit number formats. Mixed precision using Float32 for the prognostic variables only is used for (e) and (f). The tracer was injected uniformly in the lower half of the domain 50 days before. This simulation was run at an increased resolution of $\Delta = 5$km (400x200 grid points).

with a large value for Δt. The other terms are rescaled accordingly ($\tilde{f} = f\Delta$; $\tilde{\mathbf{F}} = \mathbf{F}\Delta$; please see Appendix A for a discussion of the dissipation term $\mathbf{D}$). The entire numerical integration is performed using the various reduced precision number formats, however, converted back to Float32 for model output. As some of the forcing and boundary terms remain constant, they are precomputed at higher precision during model initialisation to avoid problems with the dynamic range.

4.2 Results with 16bit 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. 6) turbulent stirring and mixing can be well simulated with posits. However, the Float16 simulation (Fig. 6d) deviates much faster than the posit simulations (Fig. 6b and c) from the Float64 reference (Fig. 6a), 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

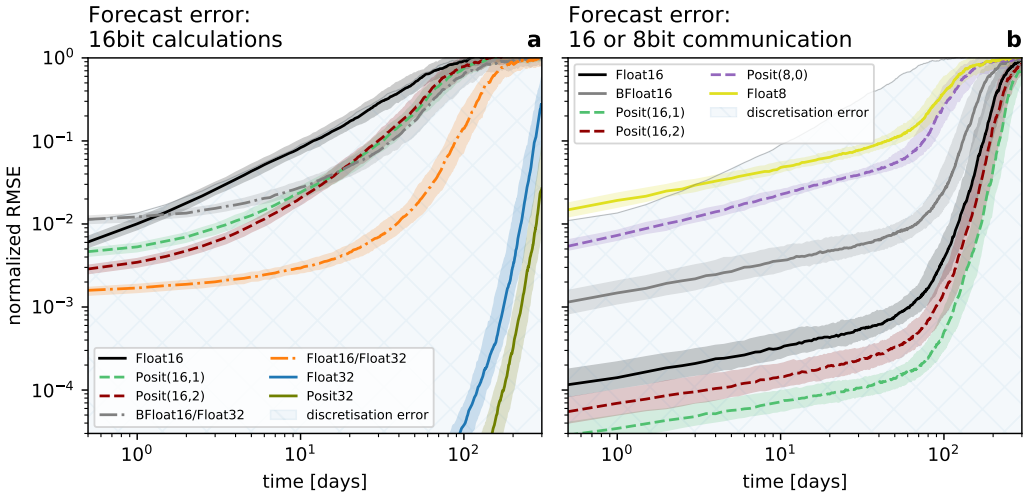

Figure 7. Forecast error measured as the root mean square error (RMSE) of sea surface height η taking Float64 as reference. (a) Forecast error for various 16bit number formats and mixed 16bit/32bit simulations for which the prognostic variables are kept at Float32. (b) Forecast error for reduced precision communication in 8 or 16bit 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 denote the interquartile range of the ensemble.

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 add be added to the prognostic variables, an issue that also occurs in the Lorenz system (Fig. 5d) and even in the harmonic sum (Fig. 3).

To quantify differences between the different 16bit arithmetics we perform ensemble forecasts that compare rounding errors. The forecast error in the shallow water model is computed as root mean square error (RMSE) of sea surface height with respect to Float64. Other variables yield similar results. The forecasts start from 200 different initial conditions of random start dates in a 50 year long control simulation. 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 at half the spatial resolution $\Delta = 40$ km. We normalise the RMSE by the climatological mean forecast error at very long lead times. 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 16bit posit arithmetic (Fig. 7a). Both Posit(16,1) and Posit(16,2) yield a forecast error that is several times smaller than Float16. The forecast error of 32bit arithmetic is several orders of magnitude smaller and is only after 200 days as large as the error for 16bit arithmetic at very short lead times. Also at 32bit, 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

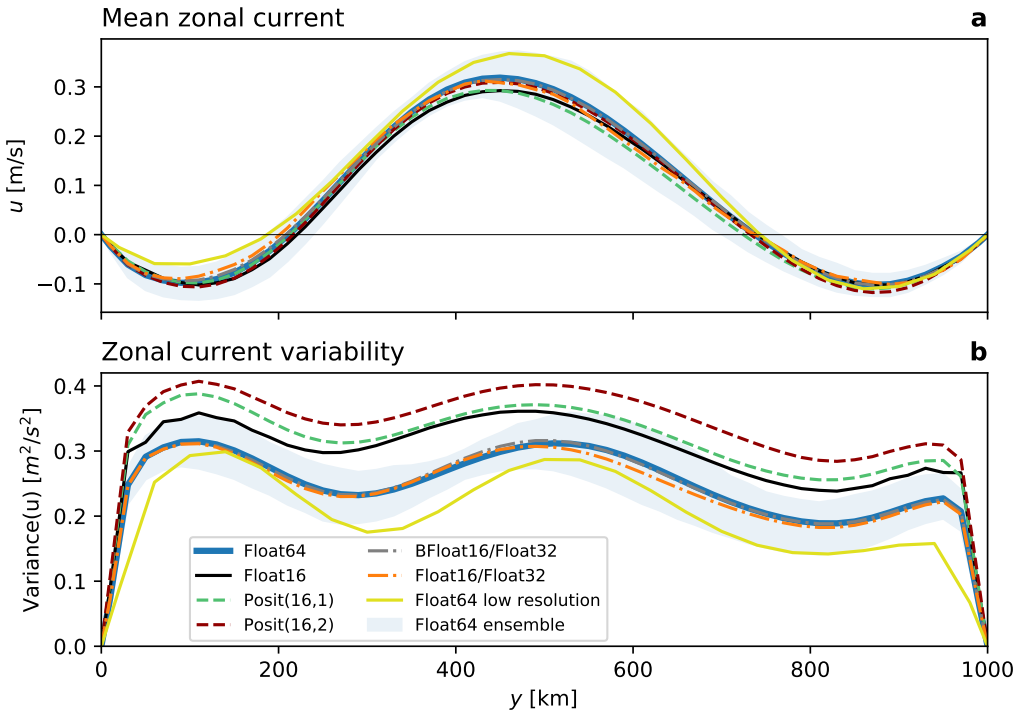

Figure 8. Climatology and variability of the zonal current. (a) Zonally-averaged zonal current u as a function of the meridional coordinate y. (b) Zonal variance of the zonal current as a function of y. Shaded areas denote the interquartile temporal variability around the (a) mean and (b) variance of reference simulation with Float64.

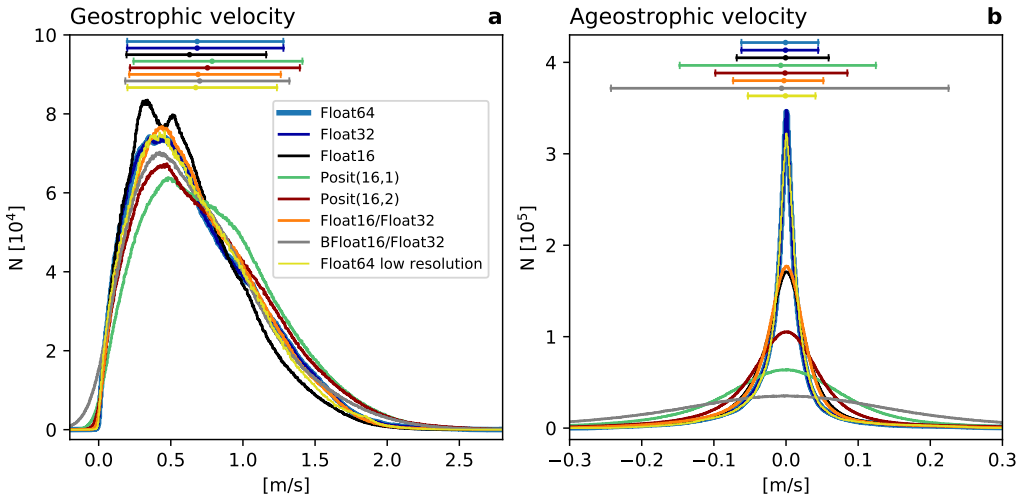

Figure 9. Geostrophic balance as simulated with different number formats. (a) Histograms of flow-parallel components of geostrophic velocity. (b) as (a) but for the ageostrophic velocities. Horizontal bars denote the mean, 10th and 90th-percentile in respective colours.

an eastward flow of about 0.3 m/s which is about 3 to 4 times weaker than individual velocities throughout the domain (Fig. 8a), which is typical for turbulent flows. A weak westward mean flow is found at the northern and southern boundary. No 16bit format was found to have a significant impact on the mean state. This is in contrast to simulations run at half the spatial resolution ($\Delta = 40$ km), which increase the strength of the mean zonal flow and decrease the return flow at the southern boundary.

The variability of the flow around its mean state is high throughout the domain (Fig. 8b). The variability is significantly increased by 10 – 30% with 16bit arithmetic, especially with Posit(16,2). The increased rounding errors with 16bit arithmetic likely trigger instabilities in the flow, which grow and subsequently explain the increased variability. Simulations at half the spatial resolution present a variability that is up to 20% weaker. We therefore assume that the increased variability of 16bit arithmetic is similar to an increase in resolution.

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

$$\mathbf{u}_g = \frac{g}{f}\hat{\mathbf{z}} \times \nabla\eta \tag{10a}$$

$$\mathbf{u} = \mathbf{u}_g + \mathbf{u}_{ag} \tag{10b}$$

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

$$\tilde{u}_g = \frac{\mathbf{u}_g \cdot \mathbf{u}}{\|\mathbf{u}\|}, \quad \tilde{u}_{ag} = \frac{\mathbf{u}_{ag} \cdot \mathbf{u}}{\|\mathbf{u}\|}. \tag{11}$$

The geostrophic velocities in the shallow water simulations can reach up to 2 m/s, are virtually never negative (i.e. against the flow) and have a mean of about 0.7 m/s (Fig. 9a). This behaviour is well simulated with 16bit number formats, although posits increase the strength of geostrophic velocities slightly. Ageostrophic velocity components are found to equally increase and decrease the flow speeds, but rarely

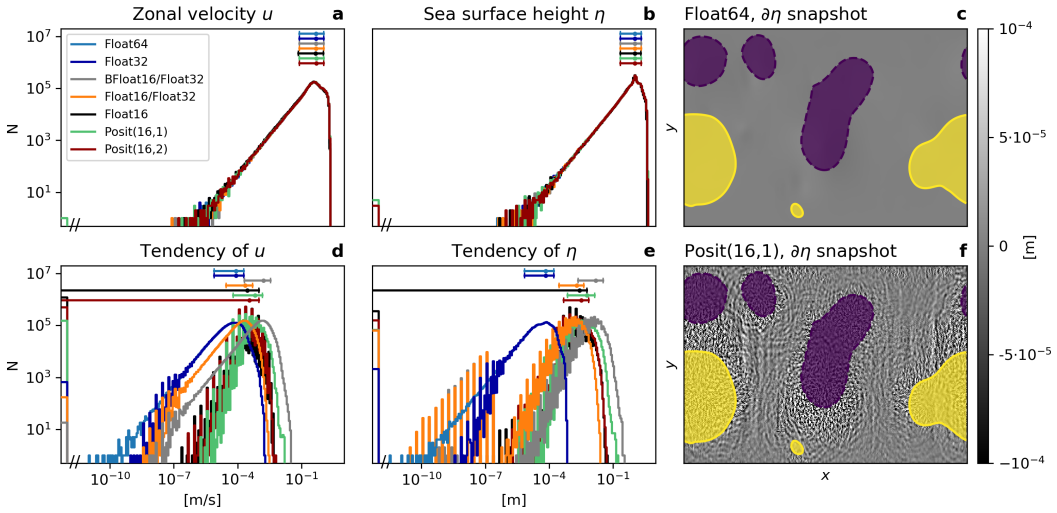

Figure 10. Histograms of the numeric values of the prognostic variables (a) zonal velocity u, (b) sea surface height η, and the respective tendencies of (d) u and (e) η, simulated with different 16, 32 and 64bit number formats. Mean, 10th and 90th percentile are shown above the histograms in respective colors. Snapshots of the tendencies of η simulated with (c) Float64 and (f) Posit(16,1), other 16bit formats are similar. 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 in (a,b,d) and (e).

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 16bit 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 16bit 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 the tracer field simulated by Float16 includes recognizably more small scale variablity compared to posits, it follows that some of this variability might actually be geostrophically balanced.

As 16bit arithmetics have no significant impact on the climatological mean state, histograms of prognostic variables are also not changed (Fig. 10a and b). However, the tendencies are increased by orders of magnitude with 16bit arithmetics (Fig. 10d and e), as rounding errors cause gravity waves to radiate away from eddies (Fig. 10f). 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. 10c) 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 error, measured as decimal precision, significantly lower than 4 decimal places (Table 1). Float16 has a machine error of 3.7, which is presumably close to the lower limit beyond which the addition of tendencies will be round back, an effect already observed for the harmonic sum. 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.

4.3 Mixed precision arithmetic

In the previous section 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-pron, 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 32bit precision respectively and let T be the conversion function then Eq. 9 becomes

$$\eta_{32}^{n+1} = \eta_{32}^{n} + RK_\eta \left(-T(\tilde{\partial}_x(u_{16}h_{16})) - T(\tilde{\partial}_y(v_{16}h_{16})) \right) \tag{12}$$

and similar for u and v.

Snapshots of tracer concentration reveal well simulated geostrophic turbulence (Fig. 6e and f) with Float16/Float32 or BFloat16/Float32 and instabilites at fronts or in filaments are visibly reduced compared to pure 16bit arithmetic. The forecast error is strongly reduced once the prognostic variables are kept as Float32 (Fig. 7a), 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. 8) compared to the Float64 reference. The geostrophic balance is largely unaffected, but ageostrophic velocities increase in variance, especially for BFloat16 (Fig. 9). Gravity waves are similarly present for mixed precision although weaker for tendencies computed with Float16 (Fig. 10d) 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 32bit precision is found to be a possible solution to circumvent roundings errors with 16bit floating-point arithmetics. Performance benefits are still possible as most calculations are performed with 16bit, with key computations in 32bit to reduce the overall error. Depending on the application, the conversions between number formats are assumed to be likely of negligible cost. This is an attractive solution as hardware-accelerated 16bit 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.

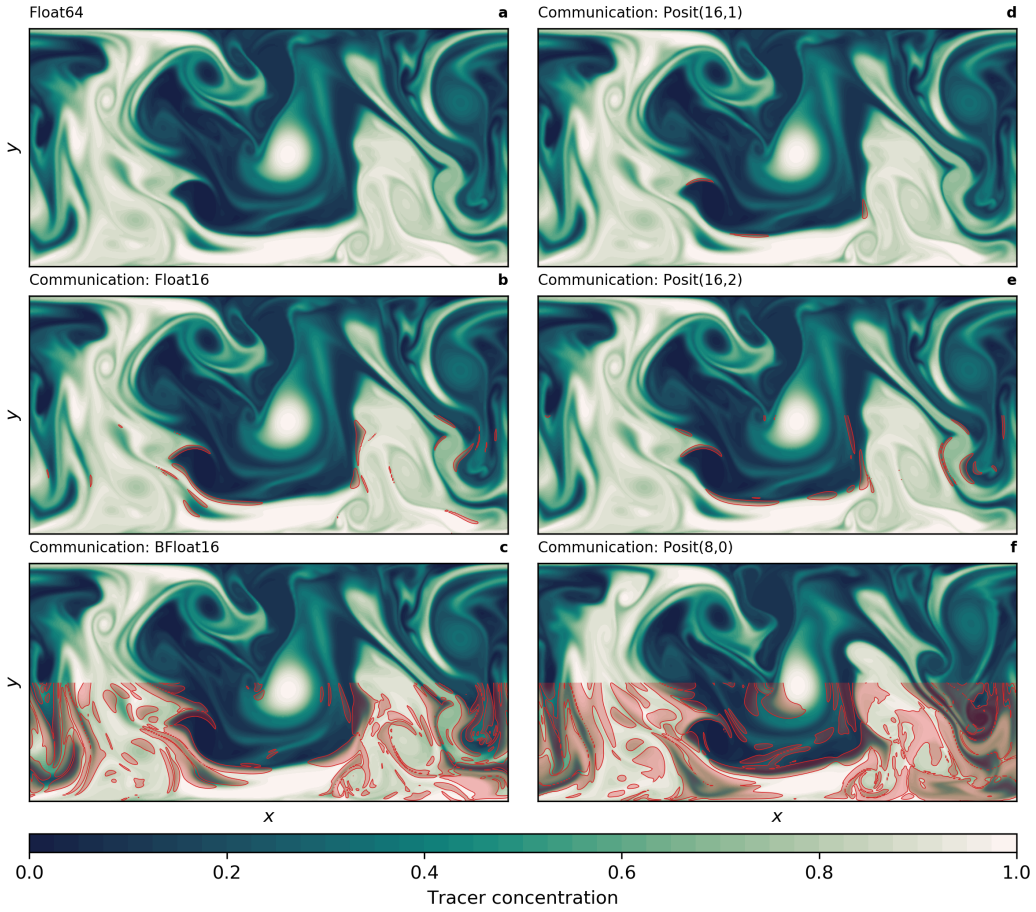

Figure 11. Snapshot of tracer concentration simulated by the shallow water model using reduced precision communication. The communication of boundary values occurs at every time step for the prognostic variables. Float64 was used for all calculations. Areas where the absolute error exceeds 0.05 are shaded 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 at a resolution of $\Delta = 5$km (400x200 grid points).

4.4 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. One of this parallelism paradigm is 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. A standard for this distributed-memory parallelism is the 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. 11). 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 16bit formats as the decimal precision is with 2.8 clearly lower (Table 1) for the range of values occuring within the prognostic variables (Fig. 10a 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 16bit arithmetic. As even the worst 16bit communication format, BFloat16, has a smaller error than the best mixed precision formats, Float16 with Float32, we extend the ensemble prediction experiments to include two 8bit formats, Posit(8,0) and Float8 (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 16bit arithmetic.

Reduced precision communication was not found to have a significant impact on either mean state, variability, geostrophy or tendencies. We acknowledge that 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. However, in the case that communication is an identified bottleneck in a given application, the results here suggest that reliable model simulations can be achieved with 16 or even 8bit communication. The range of values for the prognostic variables here is comparably small, facilitating 8bit communication. Such reductions might be in general difficult to implement. Although we show that posits are a prefereable number format to be used for 16bit communication, it remains an open question how efficient an implementation can be, given the computational cost of the conversion between formats.

5 Conclusion and Discussion

Using a software emulator we have tested posit arithmetic with a perspective for weather and climate simulations. The attractor of the Lorenz 1963 model, a chaotic but simplistic model of atmospheric convection, is considerably improved using 16bit posits with compared to Float16 or BFloat16, two formats that are supported on modern tensor or graphic processing units. Float16 can be used to perform forecasts with the shallow water model, however, 16bit posits with either 1 or 2 exponents clearly outperform floats and appear very promising for application in high performance computing for Earth System modelling. Running computationally very demanding algorithms at 16 bit could greatly reduce the wall-clock time for weather and climate simulations on

future high performance computing architecture. 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. Although the vast majority of calculations are still performed with 16 bit, the error is greatly reduced and adding the tendencies on the prognostic variables was identified as critical to reduce rounding errors. Further results show that communication, as occuring in weather and climate models run in parallel, can be greatly reduced down to 16 or even 8bit without introducing a large error that would not be masked by other sources of error in less a idealistic simulation. The approaches to reduced precision presented here can in general be combined and the overall error will be determined by the largest contributing error.

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. 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 16bit arithmetic would be dwarfed by errors in initial conditions or discretisation errors in many applications.

The numerical discretisation that was used 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. However, there is no prior reason why floats should be superior to posits in these cases and the smaller rounding errors of 16bit posits compared to Float16 and BFloat16 in our applications suggest that posits are competitive. Nonetheless, we cannot draw any preliminary conclusion about the performance of posit arithmetic operations in a standardized posit processor. Although less error-prone, posit arithmetic could be slower than float arithmetic due to hardware limitations.

Until progress is made on hardware implementations for posits, the results here suggest that also 16bit 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 Shallow water model

The shallow water equations are discretised on the (x, y)-plane over the rectangular domain $L_x \times L_y$. 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 $7.27 \times 10^{-5}\,\mathrm{s}^{-1}$ at the southern boundary to $9.25 \times 10^{-5}\,\mathrm{s}^{-1}$ 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

$$H(x) = H_0 - H_1 \exp\left(-H_\sigma^{-2}(x - \tfrac{L_x}{2})^2\right) \tag{A1}$$

being the undisturbed depth, representing a 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 maximum phase speed $\sqrt{gH_0}$ with CFL number being 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

$$Fx = \frac{F_0}{\rho h}\cos\left(\pi\left(y{L_y}^{-1} - 1\right)\right)^2 \tag{A2}$$

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

$$\mathbf{D} = -\frac{c_D}{h}\|\mathbf{u}\|\mathbf{u} - \nu\nabla^4\mathbf{u} \tag{A3}$$

of a quadratic bottom drag with dimensionless coefficient $c_D = 10^{-5}$ (Arbic & Scott, 2008) and a biharmonic diffusion with viscosity coefficient $\nu \approx 1.33 \times 10^{11}\,\mathrm{m}^4\,\mathrm{s}^{-1}$ (Griffies & Hallberg, 2000). To avoid division and subsequent multiplication with large numbers throughout the numerical model integration, we use instead

$$\tilde{\mathbf{D}} = -\frac{\tilde{c_D}}{h}\|\mathbf{u}\|\mathbf{u} - \tilde{\nu}\tilde{\nabla}^4\mathbf{u} \tag{A4}$$

with $\tilde{c_D} = c_D\Delta = 0.2\,\mathrm{m}$, and $\tilde{\nu} = \nu\Delta^{-3} \approx 0.16\,\mathrm{ms}^{-1}$. 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.

Acknowledgments

The authors would like to thank Cerlane Leong and Mosè Giordano who contributed to *SoftPosit.jl*. 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 ESIWACE project. ESIWACE has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement 675191.

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. Retrieved 2020-02-25, from `https://journals.ametsoc.org/doi/10.1175/1520-0493%281990%29118%3C1960%3AECAPED%3E2.0.CO%3B2` (Publisher: American Meteorological Society) 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 Research and Applications* (Vol. 17, pp. 173–265). Elsevier. Retrieved 2020-02-25, from `http://www.sciencedirect.com/science/article/pii/B9780124608177500094` 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. Retrieved 2020-02-25, from `http://journals.ametsoc.org/doi/10.1175/2007JPO3653.1` 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. Retrieved 2020-02-25, from `https://epubs.siam.org/doi/10.1137/141000671`

(Publisher: Society for Industrial and Applied Mathematics) 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). (ISSN: 1063-6889) 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. Retrieved 2020-02-25, from https://ieeexplore.ieee.org/document/8615707/ 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. Retrieved 2020-02-25, from http://dl.acm.org/citation.cfm?doid=3190339.3190340 doi: 10.1145/3190339.3190340

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. Retrieved 2020-02-25, from https://www.geosci-model-dev.net/10/2221/2017/ 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. Retrieved 2020-02-25, from https://onlinelibrary.wiley.com/doi/abs/10.1029/2018MS001420 doi: 10.1029/2018MS001420

Düben, P. D., McNamara, H., & Palmer, T. (2014, August). The use of imprecise processing to improve accuracy in weather & climate prediction. *Journal of Computational Physics*, *271*, 2–18. Retrieved 2020-02-25, from https://linkinghub.elsevier.com/retrieve/pii/S0021999113007183 doi: 10.1016/j.jcp.2013.10.042

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]*. Retrieved 2020-02-25, from http://arxiv.org/abs/1712.01021 (arXiv: 1712.01021)

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.

Gustafson, J. (2017). *Posit Arithmetic.* Retrieved 2020-02-26, from https://posithub.org/docs/Posits4.pdf

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. Retrieved 2020-02-25, from https://onlinelibrary.wiley.com/doi/abs/10.1029/2018MS001341 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. Retrieved 2020-02-25, from https://royalsocietypublishing.org/doi/10.1098/rspa.2017.0144 doi: 10.1098/rspa.2017.0144

Jouppi, N., Young, C., Patil, N., & Patterson, D. (2018, May). Motivation for and Evaluation of the First Tensor Processing Unit. *IEEE Micro*, *38*(3), 10–19. (Conference Name: IEEE Micro) doi: 10.1109/MM.2018.032271057

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. Retrieved 2020-02-26, from https://doi.org/10.1145/3079856.3080246 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]*. Retrieved 2020-02-26, from http://arxiv.org/abs/1905.12322 (arXiv: 1905.12322)

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. Retrieved 2020-02-25, from http://dl.acm.org/citation.cfm?doid=3316279.3316281 doi: 10.1145/3316279.3316281

Klöwer, M., & Giordano, M. (2019, December). *SoftPosit.jl - A posit arithmetic emulator.* Zenodo. Retrieved 2020-02-26, from https://zenodo.org/record/3590291 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. Retrieved 2020-02-25, from https://royalsocietypublishing.org/doi/10.1098/rsta.2013.0286 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]*. Retrieved 2020-02-26, from http://arxiv.org/abs/1907.13216 (arXiv: 1907.13216)

Lorenz, E. N. (1963, March). Deterministic Nonperiodic Flow. *Journal of the Atmospheric Sciences*, *20*(2), 130–141. Retrieved 2020-02-26, from https://journals.ametsoc.org/doi/abs/10.1175/1520-0469(1963)020%3C0130:DNF%3E2.0.CO;2 (Publisher: American Meteorological Society) 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). (ISSN: null) doi: 10.1109/IPDPSW.2018.00091

Palmer, T. (2015, October). Modelling: Build imprecise supercomputers. *Nature News*, *526*(7571), 32. Retrieved 2020-02-26, from http://www.nature.com/news/modelling-build-imprecise-supercomputers-1.18437 (Section: Comment) 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. Retrieved 2020-02-25, from http://doi.wiley.com/10.1002/qj.1923 doi: 10.1002/qj.1923

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. Retrieved 2020-02-25, from https://linkinghub.elsevier.com/retrieve/pii/S0010465517302564 doi: 10.1016/j.cpc.2017.08.011

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

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. Retrieved 2020-02-26, from https://journals.ametsoc.org/doi/abs/10.1175/1520-0469%281992%29049%3C2082%3AACOSLA%3E2.0.CO%3B2 (Publisher: American Meteorological Society) 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. Retrieved 2020-02-25, from http://link.springer.com/10.1007/s10955-017-1938-0 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. Retrieved 2020-02-25, from http://doi.wiley.com/10.1002/qj.2974 doi: 10.1002/qj.2974

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. Retrieved 2020-02-26, from https://doi.org/10.1145/3316279.3316284 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. Retrieved 2020-02-25, from http://journals.ametsoc.org/doi/10.1175/MWR-D-16-0228.1 doi: 10.1175/MWR-D-16-0228.1