

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

M. Klöwer¹, P. D. Düben², and T. N. Palmer¹

¹Atmospheric, Oceanic and Planetary Physics, University of Oxford, Oxford, UK

²European Centre for Medium-Range Weather Forecasts, Reading, UK

Key Points:

- Posit numbers have smaller arithmetic rounding errors compared to floating-point numbers in weather and climate applications
- Reduced rounding errors with posits enable reliable shallow water simulations at 16bit
- Mixed 16bit/32bit precision is recommended for weather and climate models on available floating-point based hardware.

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

Abstract

Posit numbers, a recently proposed alternative to floating-point numbers, claim to have smaller arithmetic rounding errors in many applications. By studying weather and climate models of low and medium complexity (the Lorenz system and a shallow water model) we present benefits of posits compared to floats at 16bit. As a standardized posit processor does not exist yet, we emulate posit arithmetic on a conventional CPU. Using a shallow water model, forecasts based on 16bit posits with 1 or 2 exponent bits are clearly more accurate than half precision floats. We therefore propose 16 bit with 2 exponent bits as a standard posit format, as its wide dynamic range of 32 orders of magnitude provides a great potential for many weather and climate models. Although the focus is on geophysical fluid simulations, the results are also meaningful and promising for reduced precision posit arithmetic in the wider field of computational fluid dynamics.

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 (IEEE, 2008), which is still 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 at single precision (32 bit) without a decrease in forecast skill (Vaña 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).

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 numerical precision. Using simplistic chaotic models, it was shown that the majority of 64 bits at double precision do not contain real information (Jeffress et al., 2017). Running algorithms used for weather forecast models at precision lower than single, for example with half precision 16-bit floats, 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 the widely supported single and double precision floats. 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 floats and claim to provide more precision in arithmetic calculations with fewer bits in algorithms of linear algebra

or machine learning (Gustafson & Yonemoto, 2017). 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 posit hardware is not yet available. Posit research currently focuses on hardware implementations (van Dam, 2018; Chen & Hofstee, 2018; Chaurasiya et al., 2018; Glaser et al., 2017).

The study is structured as follows: Section 2 introduces the posit number format 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, a two dimensional fluid circulation model. Section 5 discusses the results and summarises the conclusions.

2 Posit numbers

2.1 The posit number format

Following the IEEE standard on floating-point arithmetic (IEEE, 2008), floats encode a real number in terms of a sign bit, and a fixed number of exponent and significant bits (16-bit half precision floats have 1 sign, 5 exponent and 10 significant bits). Consequently, they have a constant number of significant digits throughout their dynamic range of representable numbers. This is in contrast to posit numbers, which extend floating-point arithmetic by introducing regime bits, that are mostly 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 (Gustafson & Yonemoto, 2017; Gustafson, 2017) (negative posit numbers are converted first to their two's complement, see Eq. 3)

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

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 $\text{useed} = 2^{2^{e_s}}$ 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 $\text{useed} = 4, 16, 256, \dots$ for $e_s = 1, 2, 3, \dots$, and every posit number can be written as $p = \pm 2^n \cdot (1 + f)$ with a given integer n (Gustafson & Yonemoto, 2017; Chen & Hofstee, 2018). Throughout this article we will use a notation where $\text{Posit}(n, e_s)$ defines the posit numbers with n bits including e_s exponent bits. A posit example is provided in the $\text{Posit}(8, 1)$ -system (i.e. $\text{useed} = 4$)

$$57 \approx \textcolor{red}{0}\textcolor{orange}{1}\textcolor{orange}{1}\textcolor{blue}{0}\textcolor{brown}{1}\textcolor{black}{1}\textcolor{black}{1}\textcolor{black}{1}_{\text{Posit}(8,1)} = (-1)^0 \cdot 4^2 \cdot 2^1 \cdot (1 + 2^{-1} + 2^{-2}) = 56 \quad (2)$$

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,

$$\begin{aligned} -0.28 &\approx 11011110_{\text{Posit}(8,1)} = \textcolor{red}{1}\textcolor{orange}{0}\textcolor{orange}{1}\textcolor{blue}{0}\textcolor{brown}{0}\textcolor{black}{0}\textcolor{black}{1}\textcolor{black}{0}_- \\ &= (-1)^1 \cdot 4^{-1} \cdot 2^0 \cdot (1 + 2^{-3}) = -0.28125. \end{aligned} \quad (3)$$

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

Furthermore, 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 simply returns *maxpos* instead, and similarly for underflow where the smallest representable 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 (Gustafson, 2017).

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. Appendix Appendix B includes a short discussion on the benefits of quires. The posit number format is explained in more detail in Gustafson (2017).

2.2 Emulating posit numbers in the Julia language

In order to use posits on a conventional CPU we developed the posit emulator *SoftPosit.jl* (?), a wrapper for the C-based library SoftPosit (?), written in Julia (Bezanson et al., 2014). This emulator defines conversion and arithmetic operations with posits. Julia’s programming paradigms around *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 the Julia language 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.

```
function harmonic_sum(::Type{T}, steps::Int) 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 1. A type-stable 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 *mul-*

140 *tuple dispatch* means that calling `harmonic_sum` with another type `T1` will result in a sep-
 141 arately compiled function for `T1`. We can therefore compute the harmonic sum with aribi-
 142 trary number types, as long as the zero-element `zero(T)`; the one-element `one(T)`; ad-
 143 dition; division; conversion from integer and conversion to float are defined for `T`.

```
julia> using SoftPosit
julia> using BFloat16s
julia> harmonic_sum(Float16,2000)
(7.0859375, 513)
julia> harmonic_sum(BFloat16,2000)
(5.0625, 65)
julia> harmonic_sum(Posit16,2000)      # Posit16 is the Posit(16,1) standard
(7.77734375, 1024)
```

Figure 2. Harmonic sum example use of the posit emulator *SoftPosit.jl* in the Julia shell.

144 The harmonic sum converges after 513 elements when using `Float16`. The decimal
 145 precision of `BFloat16` is so low that the sum already converges after 65 elements, as the
 146 addition of the next term $1/66$ is round back to 5.0625. We identify the addition of small
 147 terms to prognostic variables of size $\mathcal{O}(1)$ as one of the major challenges with low pre-
 148 cision arithmetic, which is discussed in more detail in section ???. Using 16bit posits with
 149 1 exponent bit, the sum only converges after 1024 terms, due to the higher decimal pre-
 150 cision of posits around 1.

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

154 2.3 Decimal precision

155 The decimal precision is defined as (Gustafson & Yonemoto, 2017; Gustafson, 2017)

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

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

163 Fig. 3a compares the worst-case decimal precision for various 16-bit number for-
 164 mats: Half precision floats, 16-bit posits with various number of exponent bits, 16-bit
 165 integers and the fixed-point format `Q6.10` (6 integer bits, 10 fraction bits). Floats have
 166 a nearly constant decimal precision of almost 4 decimal places, which decreases for the
 167 subnormal numbers towards the smallest representable number *minpos*. Posits, on the
 168 other hand, show an increased decimal precision for numbers around 1. Posits with 1 or
 169 2 exponent bits also have a wider dynamic range than half precision floats, in exchange
 170 for less precision for numbers on the order of 10^4 as well as 10^{-4} . Due to the no over-
 171 flow/no underflow-rounding mode, the decimal precision is slightly above zero outside
 172 the dynamic range.

173 The decimal precision of 16-bit integers is negative infinity for any number below
 174 0.5 (round to 0) and maximised for the largest representable integer $2^{15} - 1 = 32767$.

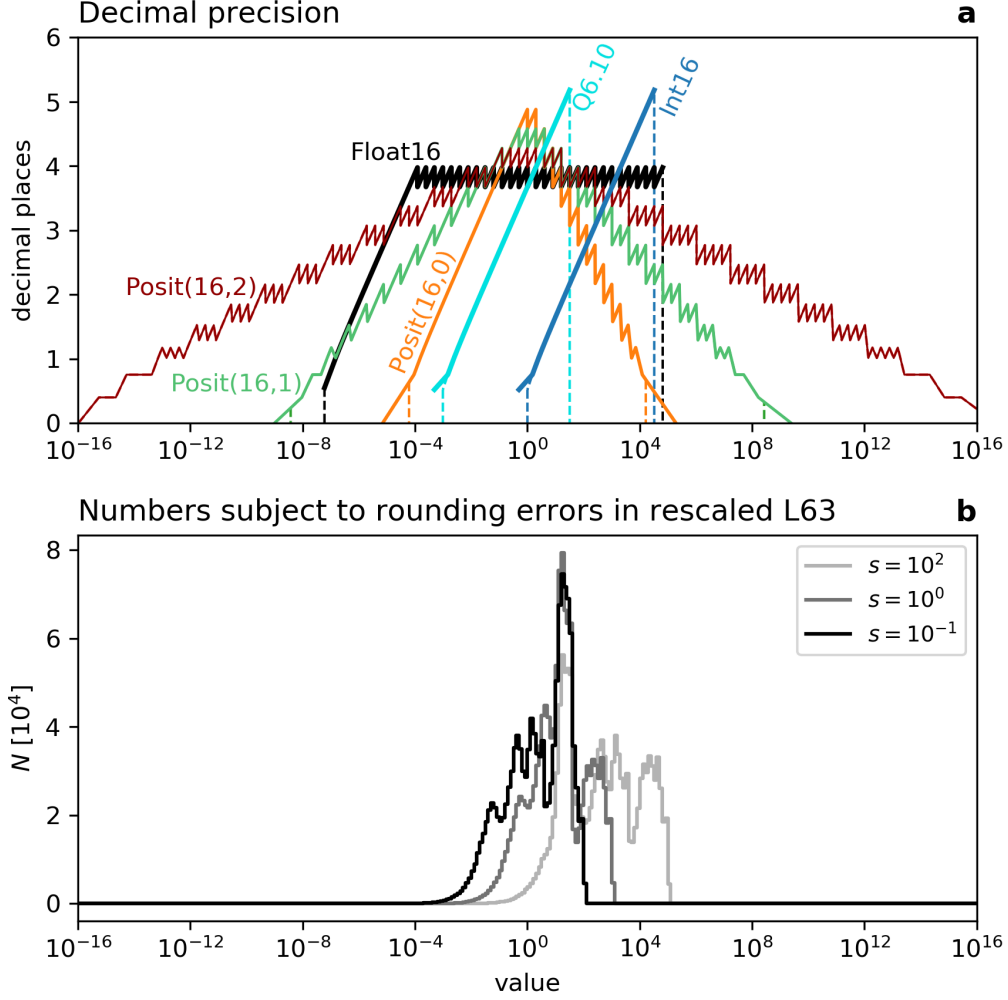


Figure 3. (a) Decimal precision of various 16-bit number formats. Dashed vertical lines indicate the dynamic range of representable numbers for each format. (b) Histogram of results of all arithmetic operations in the rescaled Lorenz system, that are subject to rounding errors, considering absolute values.

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. This indicates problems for reduced precision modelling: 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. Flexibility regarding the dynamic range can 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 paper (see section 3.2).

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

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

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

$$\frac{dz}{dt} = xy - \beta z \quad (5c)$$

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. 3b shows histograms for all numbers that are used to solve the Lorenz system (including intermediate calculations). A comparison to the decimal precision in Fig. 3a 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, 2008). Each substep in the time integration can be written as

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

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

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

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 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. 3b), 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.

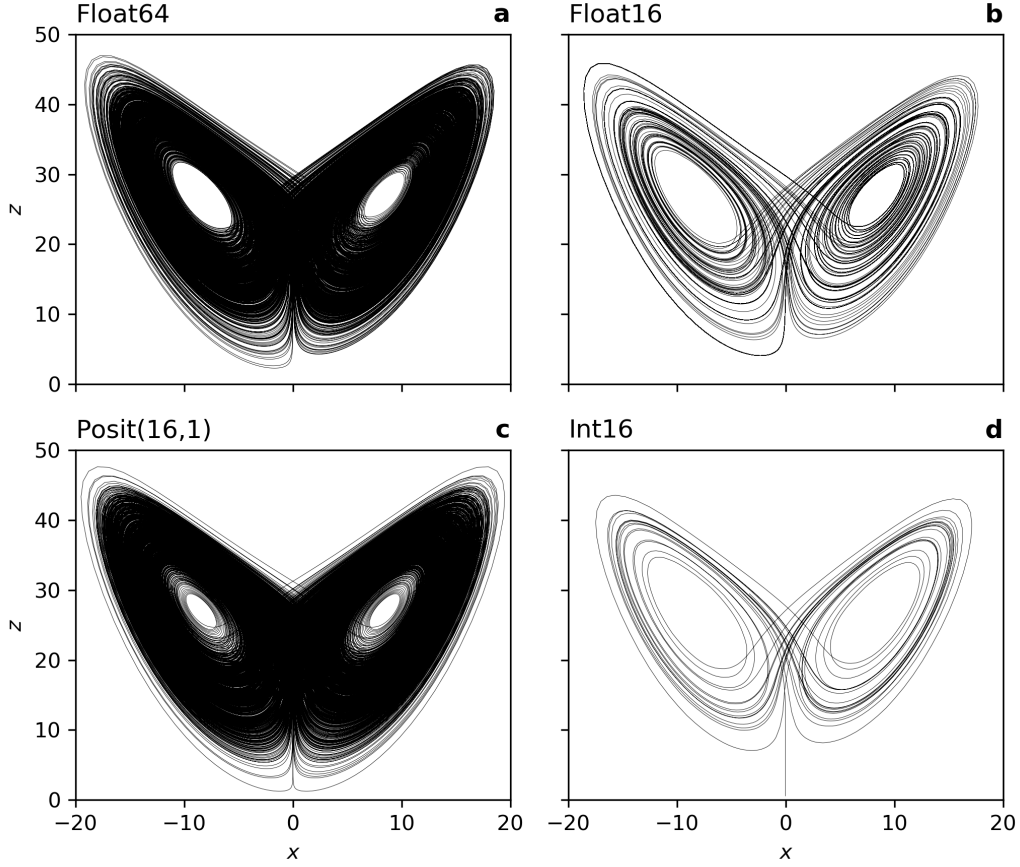


Figure 4. The Lorenz attractor computed with different arithmetics and precision. (a) 64bit double precision floats, (b) 16-bit half precision floats, (c) 16-bit posits with 1 exponent bit and (d) 16-bit integers. The scaling of the Lorenz equations (Eq. 6) is (a,b) $s = 1$, (c) $s = 0.1$ and (d) $s = 100$. All curves are integrated for the same number of time-steps.

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. 4a). 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 necessarily repeat itself if integrated for long enough.

Integrating the Lorenz system with half precision floats 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. 4a and b). However, when using posits and a rescaling factor of $s = 0.1$ the representation of the attractor is improved significantly (Fig. 4c). The results for posits looks similar to the results with 16-bit floats if no rescaling was used (not shown here). The solution of the Lorenz system with integers fails to represent the true dynamics as the model converges to the origin (Fig. 4d).

We have calculated the so-called *fractal dimension* as a diagnostic to quantify the fidelity of simulations of the discretised Lorenz equations when different number formats are used. The fractal dimension quantifies how space-filling an attractor is. Using a box-counting algorithm, we estimate the dimension of the posit attractor to be 1.78, whereas the half precision float attractor is only 1.29, compared to the true value of approximately 2.06 (Grassberger & Procaccia, 1983; McGuinness, 1983).

4 Fluid circulation model

4.1 Methods

This section will evaluate the different number formats (16-bit half precision floats, 16-bit posits with 0,1 or 2 exponent bits) 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

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

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

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 $2000 \text{ km} \times 1000 \text{ km}$, 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 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 $\Delta = 20 \text{ km}$ (100×50 grid points) and the Runge-Kutta fourth order method (Butcher, 2008) 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

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

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 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 reduced precision it is essential to rescale the shallow water equations to avoid arithmetic operations with very large or very small results, as the dynamic range of representable numbers is limited (Fig. 3a). 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 $\mathcal{O}(1 \text{ ms}^{-1})$ for \mathbf{u} , $\mathcal{O}(1 \text{ m})$ 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 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

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

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 Δ and a subsequent multiplication with a large value for Δt . The other terms are rescaled accordingly ($\tilde{f} = f\Delta$; $\tilde{\mathbf{F}} = \mathbf{F}\Delta$; please see appendix 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, posits are converted back to single precision floats for model output and some of the forcing and boundary terms that remain constant throughout the model integration are computed at higher precision during model initialisation to avoid problems with the dynamic range.

4.2 Results

The solution to the shallow water equations includes vigorous turbulence that dominates a meandering zonal current. Using either float or posit arithmetic with 16 bit the simulated fluid dynamics are very similar to a double precision reference: As shown in a snapshot of tracer concentration (Fig. 5) stirring and mixing can be well simulated with half precision floats and with 16-bit posits (2 exponent bits). However, the half precision simulation (Fig. 5c) deviates much faster than the posit simulation (Fig. 5b) from the double precision reference (Fig. 5a). This provides a first evidence that the accumulated rounding errors with posits are smaller than with floats. Only the posit simulations without exponent bit suffer from numerical instabilities, due to the limited dynamic range (Fig. 3a).

To quantify differences between reduced precision arithmetics we perform model forecasts that compare rounding errors. The forecast error in the shallow water model

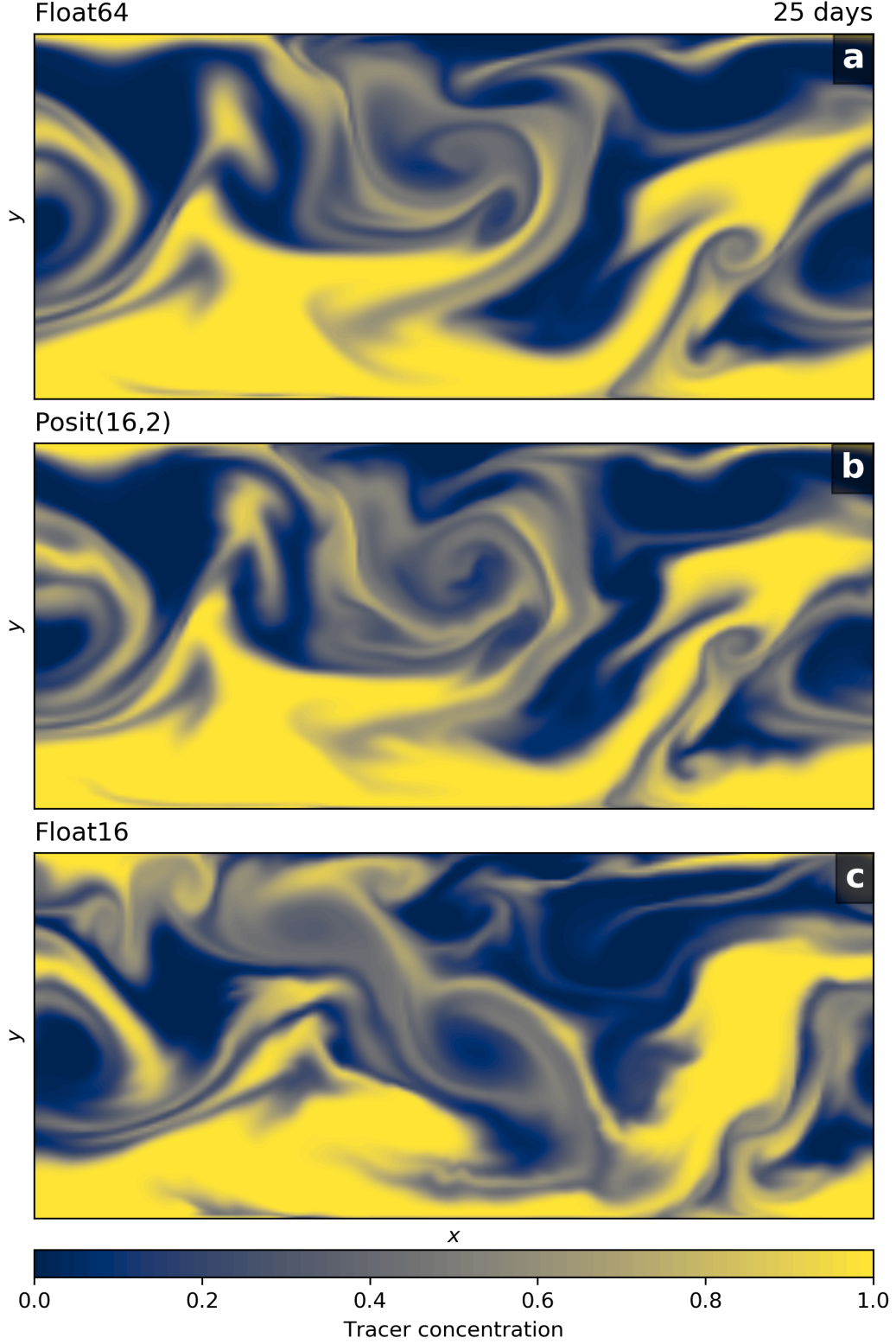


Figure 5. Snapshot of tracer concentration simulated by the shallow water model, based on (a) double precision floats and (b) posit arithmetic (16 bit with 2 exponent bits) and (c) half precision floats. The tracer was injected uniformly in the left half of the domain 25 days before. This simulation was run at a resolution of $\Delta = 10\text{km}$ (200x100 grid points). The corresponding video can be found at http://milank.de/videos/swm_posit_tracer.mp4

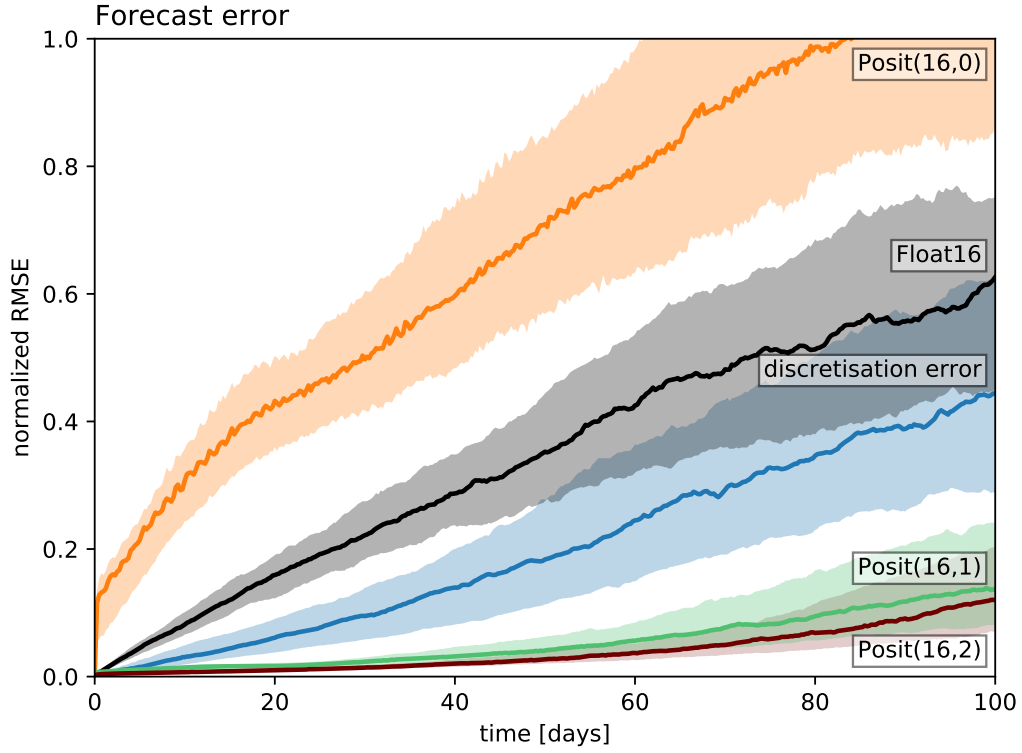


Figure 6. Forecast error measured as the root mean square error (RMSE) of sea surface height taking the double precision forecast as reference. The RMSE is normalised by a mean forecast error at very long lead times. Solid lines represent the median of 280 forecasts per number format. The shaded areas denote the interquartile range.

is computed as root mean square error (RMSE) taking the model based on double precision floating-point arithmetics as reference truth. We use the sea surface height (equivalent to pressure) to compute the forecast error as this variable captures the large scale circulation. The forecasts are created based on 280 different initial conditions from random start dates of 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 at double precision that fall back to a 3rd-order Runge-Kutta scheme for time integration and a simpler enstrophy conserving advection scheme described in Sadourny (1975). Both advection schemes have the same continuous formulation, but the Arakawa and Hsu (1990) advection scheme has a wider stencil. 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.

Clearly the best forecast is obtained for posit arithmetic with 1 or 2 exponent bits (Fig. 6), with a small accumulation of rounding errors even for lead times of 100 days. The forecast error for 16-bit posits without exponent bit increases quickly (Fig. 6), especially for short forecast lead times, but a persistence forecast, i.e. assuming the initial conditions persist over time, is still worse (not shown). Half precision floats outperform 16-bit posits without exponent bit, presumably due to the limited dynamic range of only 8 orders of magnitude compared to 12 for half precision floats (Fig. 3a). The forecast error of half precision floats is larger than the discretisation error.

5 Discussion and Conclusion

Using a software emulator we have tested posit arithmetic for weather and climate simulations. The attractor of the Lorenz 1963 model, a chaotic but simplistic model of atmospheric convection, is considerably improved using 16-bit posits with one exponent bit when compared to 16-bit half precision floats. Half precision floats can be used to perform forecasts with the shallow water model, a two-dimensional fluid circulation model that represents either atmospheric or oceanic flows. However, 16-bit posits with either 1 or 2 exponents clearly outperform floats and appear very promising for application in high performance computing for Earth System modelling. Especially 16-bit posits with 2 exponent bits, that have a wide dynamic range of 32 orders of magnitude, are likely to be widely applicable. 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.

The numerical discretisation that was used in this paper, with a fully-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 in this paper. However, there is no prior reason why floats should be superior to posits in these cases and the smaller rounding errors of 16-bit posits compared to half precision floats in our applications suggest that posits are very competitive. In contrast, the wider dynamic range of posits with 1 or 2 exponent bits compared to half precision floats will facilitate the application in more complex numerical models since it will become difficult to reduce the dynamic range of all intermediate operations for complex applications. While floating-point arithmetic is then prone to overflows or underflows, posit arithmetic will be able to tolerate very small and very large numbers, although decimal precision is decreasing away from 1 for posits.

We do not show results for the bfloat16 floating-point format in this paper as rounding errors destroy the dynamics of the shallow water model. Due to the 8 exponent bits the dynamic range of bfloat16 (10^{-40} to 10^{38}) remains the same as for single precision simulations. However, the small number of 7 fraction bits in this format causes rounding errors that inhibit the time evolution of the model.

In this paper, we perform model forecasts with a *perfect model*. Any form of model error is ignored, as the double precision 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 changing the advection scheme to a simpler non-energy conserving form (Sadourny, 1975) and by using a third-order instead of a fourth-order Runge-Kutta method. Here, we are likely underestimating the discretisation error of real models which also arises from the limited accuracy of spatial discretisation schemes.

This is not a realistic set-up for weather or climate models. Real models include many other sources of forecast error (see section 1) 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. As the forecast error with 16-bit posits (1 or 2 exponent bits) are still considerably lower than the discretisation error, this suggests that simulations with posit arithmetic of even less than 16 bit may be feasible. However, as 8bit numbers are very likely unsuitable for applications in weather and climate models, we propose 16-bit posit with 2 exponent bits as a format that would likely meet the requirements of many algorithms used in weather and climate models. Together with a 32-bit posit format with 2 exponent bits (to match the dynamic range of single precision floats), a posit processor based on these two formats could greatly support the transition of models that are rewritten to use less than 32 bits to represent real numbers. We believe that high performance computing for Earth System modelling would benefit greatly from a processor that would support both 16 and 32-bit posit formats with 2 exponent bits.

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 30N and the beta-plane approximation (Vallis, 2006) is used to linearize the Coriolis parameter which varies linearly from $7.27 \times 10^{-5} \text{ s}^{-1}$ at the southern boundary to $9.25 \times 10^{-5} \text{ s}^{-1}$ at the northern boundary. The boundary conditions are periodic in zonal direction and partial slip at the northern and southern boundary. The layer thickness is $h = \eta + H(x)$, with

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

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 $H_0 = 500 \text{ m}$. The ridge has a height of $H_1 = 50 \text{ m}$. The characteristic width of the ridge is $H_\sigma = 300 \text{ km}$. The time step $\Delta t = 282 \text{ s}$ is chosen to resolve surface gravity waves, traveling at maximum phase speed $\sqrt{gH_0}$ with CFL number being 1 and gravitational acceleration $g = 10 \text{ ms}^{-1}$. The wind stress forcing $\mathbf{F} = (F_x, 0)$ is constant in time, acts only on the zonal momentum budget

$$Fx = \frac{F_0}{\rho h} \cos\left(\pi\left(yL_y^{-1} - 1\right)\right)^2 \quad (\text{A2})$$

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

$$\mathbf{D} = -r\mathbf{u} - \nu\nabla^4\mathbf{u} \quad (\text{A3})$$

of a linear bottom drag with time scale $r^{-1} = 300 \text{ days} \approx 2.6 \times 10^7 \text{ s}$ (Arbic & Scott, 2008) and a biharmonic diffusion with viscosity coefficient $\nu \approx 1.33 \times 10^{11} \text{ m}^4 \text{ s}^{-1}$ (Griffies & Hallberg, 2000). To avoid division and subsequent multiplication with large numbers throughout the numerical model integration, we use instead

$$\tilde{\mathbf{D}} = -\tilde{r}\mathbf{u} - \tilde{\nu}\tilde{\nabla}^4\mathbf{u} \quad (\text{A4})$$

with $\tilde{r} = r\Delta \approx 0.0008 \text{ ms}^{-1}$, and $\tilde{\nu} = \nu\Delta^{-3} \approx 0.16 \text{ ms}^{-1}$. 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.

The semi-Lagrangian advection scheme involves the computation of a departure point \mathbf{x}_d . In order to avoid large numbers of the coordinates ($L_x = 2 \cdot 10^6 \text{ m}$), we use instead a non-dimensional coordinate $\tilde{\mathbf{x}} = \mathbf{x}\Delta^{-1}$ to compute the departure point as

$$\tilde{\mathbf{x}}_d = \tilde{\mathbf{x}}_a - \mathbf{u}_i \frac{\Delta t_{\text{adv}}}{\Delta} \quad (\text{A5})$$

where $\tilde{\mathbf{x}}_a$ are the coordinates of the arrival grid node and \mathbf{u}_i the velocity interpolated on the mid-point in space and time (Smolarkiewicz & Pudykiewicz, 1992). The advective time step Δt_{adv} is much larger than Δt to reduce numerical diffusion of the tracer due to a smaller number of interpolations. In the simulations of Fig. 5 ($\Delta = 10 \text{ km}$) the rescaled time step is $\frac{\Delta t_{\text{adv}}}{\Delta} \approx \frac{2 \cdot 10^4 \text{ s}}{10^4 \text{ m}} = 2 \text{ sm}^{-1}$ and therefore precomputed.

Appendix B Perspectives for quires

Although we do not use quires throughout the simulations, for completeness we want to discuss computations that could greatly benefit from the use of quires. Summing the tendencies of the right-hand side of Eq. 7a involves computations like

$$u^{n+1} = u^n + RK_u (Qhv + \partial_x p + D_x + F_x) \quad (\text{B1})$$

where RK_u is a constant that includes the Runge-Kutta coefficient and the time step. Qhv is the advection of potential vorticity, $\partial_x p$ is the gradient of the Bernoulli potential, D_x is the u -component of bottom friction and diffusion and F_x is the wind forcing. It is a priori not clear which of the terms $u^n, Qhv, \partial_x p, D_x$ or F_x dominate the sum. Physically speaking, the shallow water model is often close to geostrophic balance, which means that the Coriolis term (which is included in Qhv) and the pressure gradient term (which is included in $\partial_x p$) oppose each other. In general, however, the dominating balance will vary in space and time and therefore it is only clear at runtime what the preferred order of addition is, which is crucial to reducing the rounding error. Using quires could solve this problem since quires will be able to reduce rounding errors greatly when performing intermediate operations. Quires may allow to perform the sum over different terms to calculate the right-hand side of the equations with a single rounding error due to 16-bit arithmetic when the final result for the new velocity value is stored.

Acknowledgments

The authors would like to thank Isaac Yonemoto for providing the Julia-based emulator for posit numbers, that was used for this study.

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). *Energy Conserving and Potential-Enstrophy Dissipating Schemes for the Shallow Water Equations* (Vol. 118) (No. 10). Retrieved from [http://dx.doi.org/10.1175/1520-0493\(1990\)118<3C1960:ECAPED>3.0.CO;2](http://dx.doi.org/10.1175/1520-0493(1990)118<3C1960:ECAPED>3.0.CO;2) doi: 10.1175/1520-0493(1990)118<1960:ECAPED>2.0.CO;2
- Arakawa, A., & Lamb, V. R. (1977). Computational design of the basic dynamical processes of the UCLA general circulation model. *Methods of Computational Physics*, 17, 173–265.
- Arbic, B. K., & Scott, R. B. (2008). 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. (2014). Julia: A Fresh Approach to Numerical Computing. , 59(1), 65–98. Retrieved from <http://arxiv.org/abs/1411.1607> doi: 10.1137/141000671
- Butcher, J. C. (2008). *Numerical Methods for Ordinary Differential Equations* (2nd ed.). Wiley.
- Chaurasiya, R., Gustafson, J., Shrestha, R., Neudorfer, J., Nambiar, S., & Niyogi, K. (2018). Parameterized Posit Arithmetic Hardware Generator. , 9. Retrieved from https://posithub.org/docs/iccd_submission_v1.pdf
- Chen, J., & Hofstee, H. P. (2018). A Matrix-Multiply Unit for Posits in Reconfigurable Logic Leveraging (Open) CAPI. , 1–5. doi: 10.1145/3190339.3190340
- Dawson, A., & Düben, P. D. (2017). 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). A new number format for ensemble simulations. *Journal of Advances in Modeling Earth Systems*. Retrieved from <http://doi.wiley.com/10.1029/2018MS001420> doi: 10.1029/2018MS001420
- Düben, P. D., & Palmer, T. N. (2014). Benchmark Tests for Numerical Weather Forecasts on Inexact Hardware. *Monthly Weather Review*, 142(10), 3809–3829. Retrieved from <http://journals.ametsoc.org/doi/abs/10.1175/MWR-D-14-00110.1> doi: 10.1175/MWR-D-14-00110.1
- Gill, A. E. (1982). *Atmosphere-Ocean Dynamics*. Academic Press.
- Glaser, F., Mach, S., Rahimi, A., Gürkaynak, F. K., Huang, Q., & Benini, L. (2017). An 826 MOPS, 210 uW/MHz Unum ALU in 65 nm. Retrieved from <http://arxiv.org/abs/1712.01021> doi: 10.1109/ISCAS.2018.8351546
- Grassberger, P., & Procaccia, I. (1983). Measuring the strangeness of strange attractors. *Physica D: Nonlinear Phenomena*, 9(1-2), 189–208. doi: 10.1016/0167-2789(83)90298-1
- Griffies, S. M., & Hallberg, R. (2000). Biharmonic Friction with a Smagorinsky-Like Viscosity for Use in Large-Scale Eddy-Permitting Ocean Models. *Monthly Weather Review*, 128(8), 2935–2946. doi: 10.1175/1520-0493(2000)128<2935
- Gustafson, J. L. (2017). *Posit Arithmetic*. Retrieved 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), 71–86. Retrieved from <http://superfri.org/superfri/article/view/137> doi: 10.14529/jsfi170206
- Hatfield, S., Düben, P., Chantry, M., Kondo, K., Miyoshi, T., & Palmer, T. (2018). Choosing the Optimal Numerical Precision for Data Assimilation in the Presence of Model Error. *Journal of Advances in Modeling Earth Systems*. doi: 10.1029/2018MS001341
- IEEE. (2008, aug). IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2008*, 1–70. doi: 10.1109/IEEESTD.2008.4610935

- Jeffress, S., Düben, P., & Palmer, T. (2017). Bitwise efficiency in chaotic models. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Science*, 473(2205), 20170144. Retrieved from <http://rspa.royalsocietypublishing.org/lookup/doi/10.1098/rspa.2017.0144> doi: 10.1098/rspa.2017.0144
- Kwasniok, F. (2014). 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). doi: 10.1098/rsta.2013.0286
- Lorenz, E. N. (1963). Deterministic Nonperiodic Flow. *Journal of the Atmospheric Sciences*, 20, 130–141. doi: 10.1175/1520-0469(1963)020<0130:DNF>2.0.CO;2
- McGuinness, M. J. (1983). The fractal dimension of the Lorenz attractor. *Physics Letters A*, 99(1), 5–9. doi: 10.1016/0375-9601(83)90052-X
- Palmer, T. (2015). Build imprecise supercomputers. *Nature*, 526, 2–3. doi: 10.1038/526032a
- Palmer, T. N. (2012). 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*. Retrieved from http://www.cosmo-model.org/content/model/documentation/newsLetters/newsLetter14/cn14_09.pdf
- Russell, F. P., Düben, P. D., Niu, X., Luk, W., & Palmer, T. N. (2017). Exploiting the chaotic behaviour of atmospheric models with reconfigurable architectures. *Computer Physics Communications*, 221, 160–173. Retrieved from <http://dx.doi.org/10.1016/j.cpc.2017.08.011> doi: 10.1016/j.cpc.2017.08.011
- Sadourny, R. (1975). *The Dynamics of Finite-Difference Models of the Shallow-Water Equations* (Vol. 32) (No. 4). Retrieved from [http://dx.doi.org/10.1175/1520-0469\(1975\)032<0320:TDOFDM>2.0.CO;2](http://dx.doi.org/10.1175/1520-0469(1975)032<0320:TDOFDM>2.0.CO;2) doi: 10.1175/1520-0469(1975)032<0680:TDOFDM>2.0.CO;2
- Smolarkiewicz, P. K., & Pudykiewicz, J. A. (1992). 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). Resonances in a Chaotic Attractor Crisis of the Lorenz Flow. *Journal of Statistical Physics*, 170(3), 584–616. Retrieved from <https://doi.org/10.1007/s10955-017-1938-0> doi: 10.1007/s10955-017-1938-0
- Thornes, T., Düben, P., & Palmer, T. (2017). On the use of scale-dependent precision in Earth System modelling. *Quarterly Journal of the Royal Meteorological Society*, 143(703), 897–908. doi: 10.1002/qj.2974
- Vallis, G. K. (2006). *Atmospheric and Ocean Fluid Dynamics*. Cambridge University Press.
- Váňa, F., Düben, P., Lang, S., Palmer, T., Leutbecher, M., Salmond, D., & Carver, G. (2017). Single Precision in Weather Forecasting Models: An Evaluation with the IFS. *Monthly Weather Review*, 145(2), 495–502. Retrieved from <http://journals.ametsoc.org/doi/10.1175/MWR-D-16-0228.1> doi: 10.1175/MWR-D-16-0228.1
- van Dam, L. (2018). *Enabling High Performance Posit Arithmetic Applications Using Hardware Acceleration* (Unpublished doctoral dissertation). Delft University of Technology.