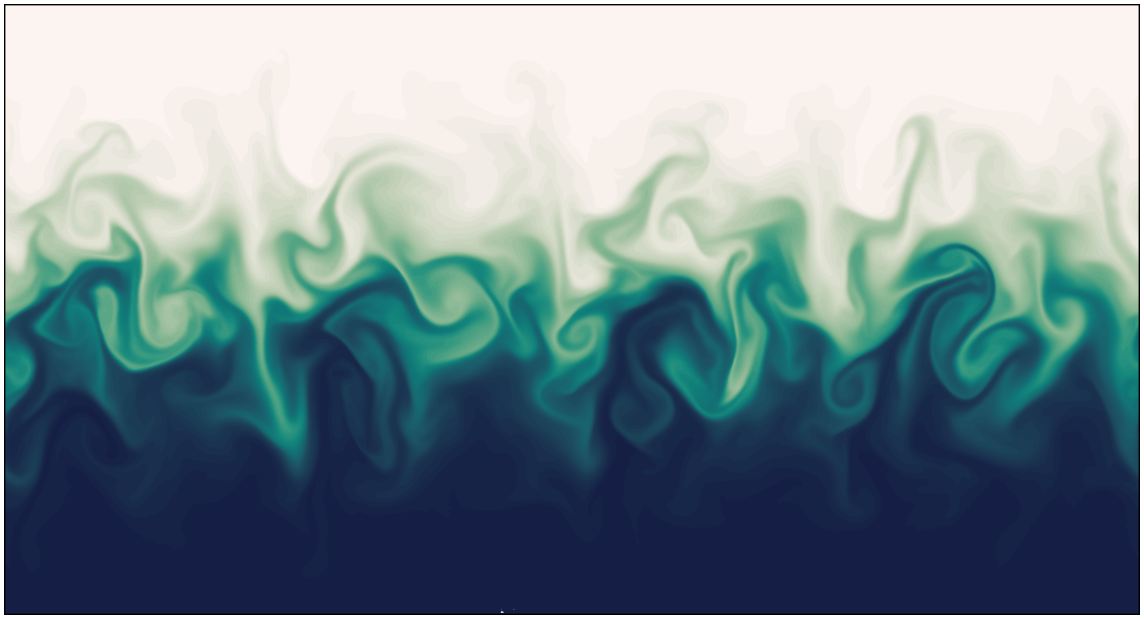
Towards 16bit weather and climate models: Posit numbers as an alternative to floating-point numbers



Milan Klöwer
Atmospheric, Oceanic and Planetary Physics
University of Oxford

supervised by

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

A report submitted for the transfer of status to DPhil candidate.

Oxford, March 2019

The cover page shows a shallow water simulation entirely computed with 16bit posits.

Abstract

Weather and climate models are based on computations of real numbers, which are represented on a computer in 64 bits with a finite numerical precision. The standard are so-called floating-point numbers (floats), that encode a real number in terms of sign, exponent and significant bits. Many bits, especially of the significand, do not contain real information, which promotes the use of reduced precision floats with less than 64 bits. However, floats might not be the best bit-wise representation for real numbers in numerical models of weather and climate. Posit numbers are a recently proposed alternative, which extend the floating-point standard by the concept of regime bits. Posits therefore have a higher numerical precision around one, yet a wide dynamic range of representable numbers. We study two weather and climate models of low and medium complexity, the Lorenz 1963 system and a shallow water model, to present benefits of posits compared to floats at 16 bit. As a standardised posit processor does not exist yet, we use posit arithmetic on a conventional CPU via a Julia-based emulator. The fractal dimension of the Lorenz attractor is clearly improved with posits compared to floats. The finite difference algorithms of the shallow water model are written in a way to overcome the limitations of 16bit numbers, which greatly improves the model's resilience to rounding errors and arithmetic overflows of both number formats. Forecasts with the shallow water model based on 16bit posits with 1 or 2 exponent bits are clearly more accurate than half precision floats and the error due to the rounding errors remains much lower when compared to the discretisation error. Although this study focuses on idealised simulations, the results show potential for weather and climate modelling with dynamical cores that are largely based on 16bit computations. Especially 16bit posits with 2 exponent bits provide a great potential for many weather and climate models, due to its 32 orders of magnitude-wide range of representable numbers. Together with a 32bit posit format, a posit processor based on these two formats could greatly support the transition of models to be rewritten for less than 32 bit. We believe that high performance computing for Earth System modelling would benefit greatly from a processor that would support both 16 and 32bit posit formats.

Contents

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 [Palmer, 2015, 2012]:

(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

(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 64bit 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 (32bit) 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].

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 64bits 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 (16bit) floats, is an active field of

research, but remains challenging [Düben, 2018; Düben & Palmer, 2014; Hatfield *et al.*, 2018; Thornes *et al.*, 2017]. 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 field programmable gate arrays (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 16bit 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 [Chaurasiya *et al.*, 2018; Chen *et al.*, 2018; Glaser *et al.*, 2017; van Dam, 2018].

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.

This study has recently been accepted for publication [Klöwer *et al.*, 2019].

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 (16bit 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 arise from the idea to project the real axis onto a circle (Fig. 2.1). Although probably useless for real applications, the simplest posit format in 2bit illustrates some main properties of the posit number format: Posits do not have any redundant bit patterns. There is one pattern for zero, one for (complex) infinity and no bit pattern for Not-a-Number (NaN, complex infinity serves as a replacement). This is in contract to half precision floats, that have 2046 bit patterns that describe NaN, or 3% of all representable numbers. All posit formats have as many numbers between 0 and 1 as between 1 and infinity. The circle is split into *regimes*, determined by a constant *useed*, which is always found in the north-west on the posit circle (Fig. 2.1b) and will be defined shortly. Further regimes are defined by $useed^{\pm 2}$, $useed^{\pm 3}$, $useed^{\pm 4}$, etc. (Fig. 2.2). 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 [Gustafson, 2017; Gustafson & Yonemoto, 2017] (negative posit numbers are converted first to their two's complement, see Eq. 2.3)

$$p = (-1)^{sign\ bit} \cdot useed^k \cdot 2^e \cdot (1 + f) \tag{2.1}$$

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

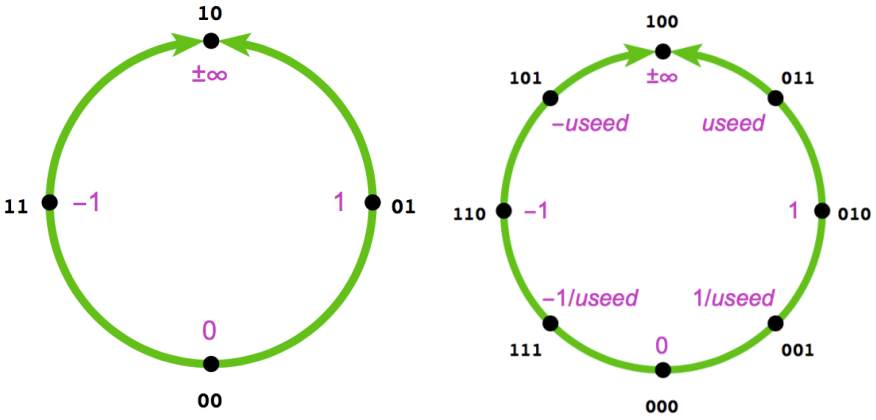


Figure 2.1: The two simplest posit number systems: (left) 2bit and (right) 3bit; obtained by projecting the real axis onto a circle. The bit patterns are marked on the outside in black and the respective values in purple on the inside of each circle. The value of *useed* depends on the number of exponent bits as explained in the text. Reproduced from Gustafson [2017].

the value of 2^e in Eq. 2.1. 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 [Chen *et al.*, 2018; Gustafson & Yonemoto, 2017]. Throughout this article 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. $useed = 4$)

$$\begin{aligned} 57 &\approx 01110111_{\text{Posit(8,1)}} \\ &= (-1)^0 \cdot 4^2 \cdot 2^1 \cdot (1 + 2^{-1} + 2^{-2}) = 56 \end{aligned} \tag{2.2}$$

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 inverting all bits and adding 1,

$$\begin{aligned} -0.28 \approx 11011110_{\text{Posit(8,1)}} &= 10100010\_ \\ &= (-1)^1 \cdot 4^{-1} \cdot 2^0 \cdot (1 + 2^{-3}) = -0.28125. \end{aligned} \tag{2.3}$$

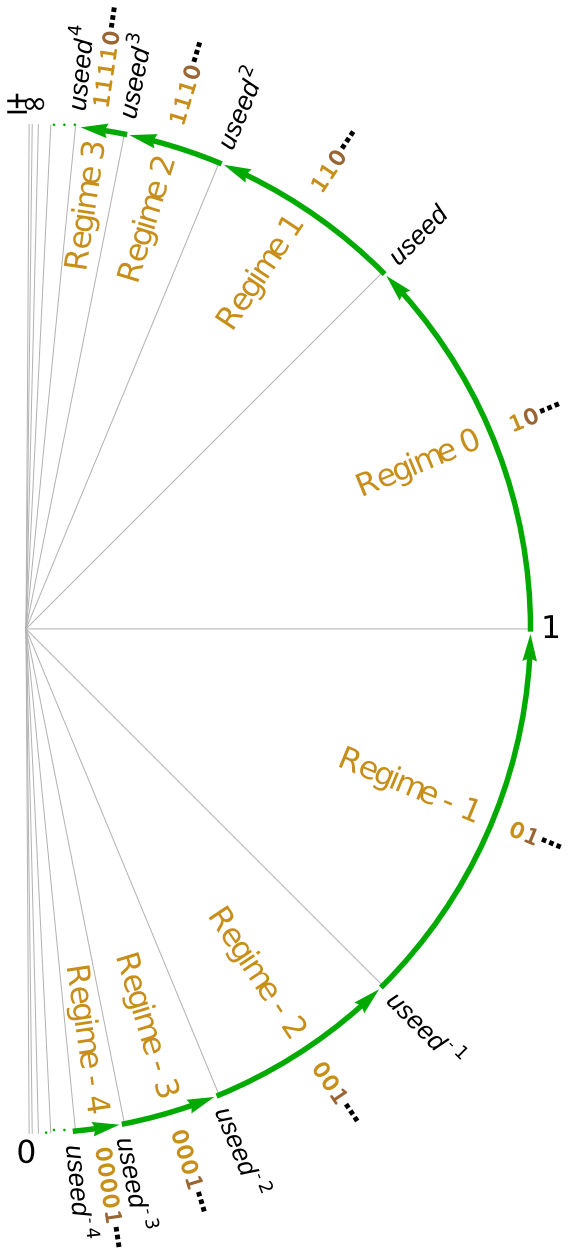


Figure 2.2: Posit regimes visualised on the posit circle. Typical values for *useed* are $2, 4$ or 16, depending on the number of exponent bits in the respective posit format. The angles that are spanned by the regimes on the unit circle is proportional to the amount of representable numbers within. Bit patterns on the outside denote the regime bits including the terminating bit. Regimes are numbered by their k-value. Reproduced from Gustafson [2017].

After the conversion to the two's complement, the bits are interpreted in the same way as in Eq. 2.2.

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 A.3 includes a short discussion on the benefits of quires. The posit number format is explained in more detail in Gustafson [2017].

```
julia> # define posit environment
julia> using SigmoidNumbers
julia> Posit161 = Posit{16,1};
julia> # convert float to 16bit posit, add
julia> a = Posit161(12.3);
julia> c = a+a;
julia> # bits split in sign, regime, exponent and fraction
julia> bits(c," ")
"0 1110 0 1000100110"
julia> Float64(c)   # convert back to double
24.59375
```

Figure 2.3: Example use of the posit emulator *SigmoidNumbers* in the Julia shell.

In order to use posits on a conventional CPU we use the posit emulator *SigmoidNumbers* written by Isaac Yonemoto in Julia Bezanson *et al.* [2014]. This emulator defines conversion to and from floats and arithmetic operations with posits (see Fig. 2.3 for an example). Consequently, posit arithmetic can be used for the numerical integration of the Lorenz equations (section 3) and the shallow water model (section 4).

2.2 Decimal precision

The decimal precision is defined as [Gustafson, 2017; Gustafson & Yonemoto, 2017]

$$\text{decimal precision} = -\log_{10}|\log_{10}(\frac{x_{\text{repr}}}{x_{\text{exact}}})| \tag{2.4}$$

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 round-to-nearest rounding mode, the decimal precision approaches infinity when the exact result approaches the representable number and has a minimum in between two representable numbers. This minimum defines the *worst-case* decimal precision, i.e. the decimal precision when the rounding error is maximised. The worst-case decimal precision is the number of decimal places that are at least correct after rounding.

Fig. 2.4a compares the worst-case decimal precision for various 16bit number formats: Half precision floats, 16bit posits with various number of exponent bits, 16bit integers and the fixed-point format Q6.10 (6 integer bits, 10 fraction bits). Floats have a nearly constant decimal precision of almost 4 decimal places, which decreases for the subnormal numbers towards the smallest representable number *minpos*. Posits, on the other hand, show an increased decimal precision for numbers around 1. Posits with 1 or 2 exponent bits also have a wider dynamic range than half precision floats, in exchange for less precision for numbers on the order of 10^4 as well as 10^{-4}. Due to the no overflow/no underflow-rounding mode, the decimal precision is slightly above zero outside the dynamic range.

The decimal precision of 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. 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).

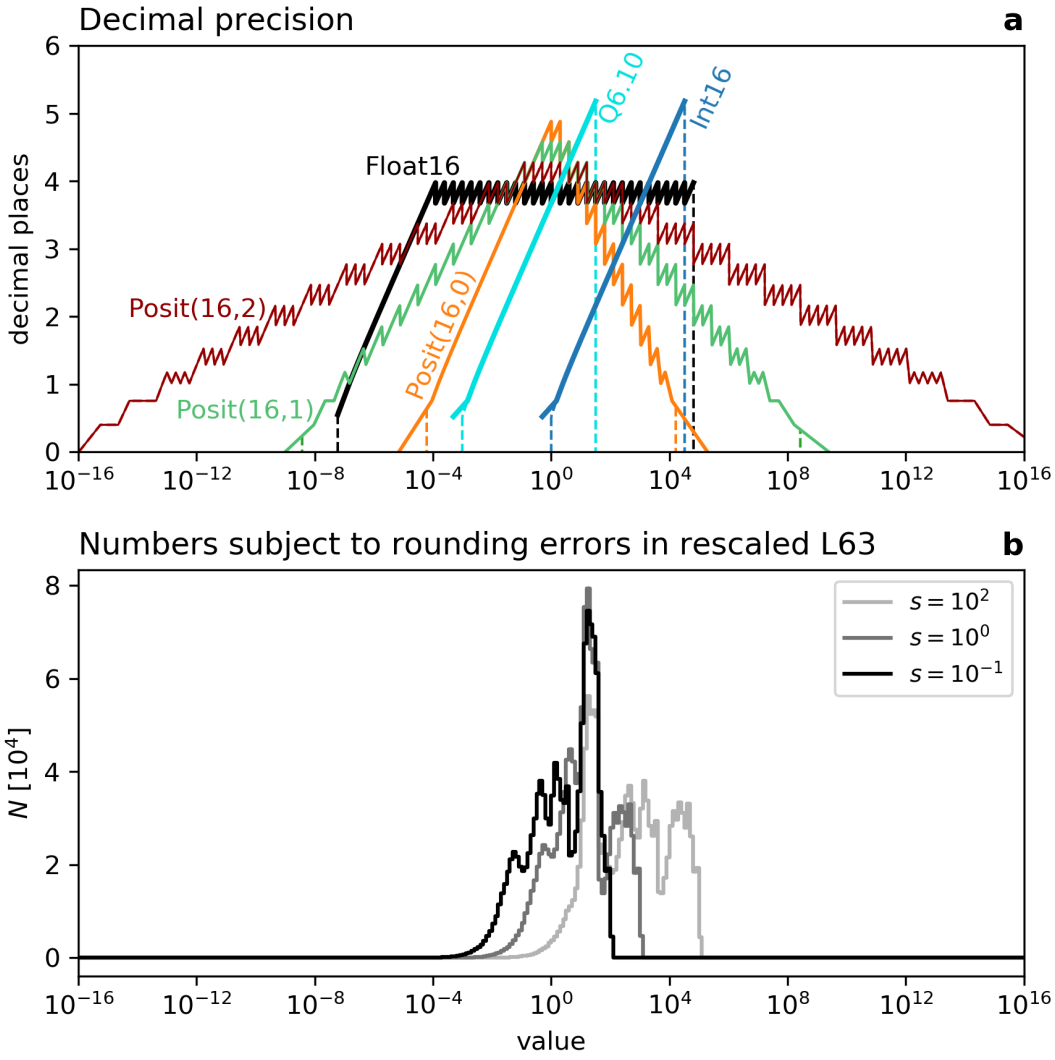


Figure 2.4: (a) Decimal precision of various 16bit number formats. Dashed vertical lines indicate the dynamic range of representable numbers for each format. (b) Histogram of results of all arithmetic operations in the rescaled Lorenz system, that are subject to rounding errors, considering absolute values.

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 [Jeffress *et al.*, 2017; Kwasniok, 2014; Lorenz, 1963; 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

$$\frac{dx}{dt} = \sigma(y - x) \tag{3.1a}$$

$$\frac{dy}{dt} = x(\rho - z) - y \tag{3.1b}$$

$$\frac{dz}{dt} = xy - \beta z \tag{3.1c}$$

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

To find the optimal number format to solve Eq. 3.1 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. 2.4b shows histograms for all numbers that are used to solve the Lorenz system (including all intermediate calculations). A comparison to the decimal precision in Fig. 2.4a 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, 2008].

Each substep in the time integration can be written as

$$\tilde{x}^{n+1} = \tilde{x}^n + RK_x\,(\tilde{y}^n - \tilde{x}^n) \tag{3.2a}$$

$$\tilde{y}^{n+1} = \tilde{y}^n + RK_y\left(\tilde{x}^n(\rho - \frac{\tilde{z}^n}{s}) - \tilde{y}^n\right) \tag{3.2b}$$

$$\tilde{z}^{n+1} = \tilde{z}^n + RK_z\left(\tilde{x}^n\frac{\tilde{y}^n}{s} - \beta\tilde{z}^n\right) \tag{3.2c}$$

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. 3.2 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. 2.4b), 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/rewrite 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. 3.1a). 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, for the discretised model with finite precision variables, 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. 3.1a and b). However, when using posits and a rescaling factor of $s = 0.1$ the representation of the attractor is improved significantly (Fig. 3.1c). The results for posits look similar to the results with half precision floats (16bit) if no rescaling was used

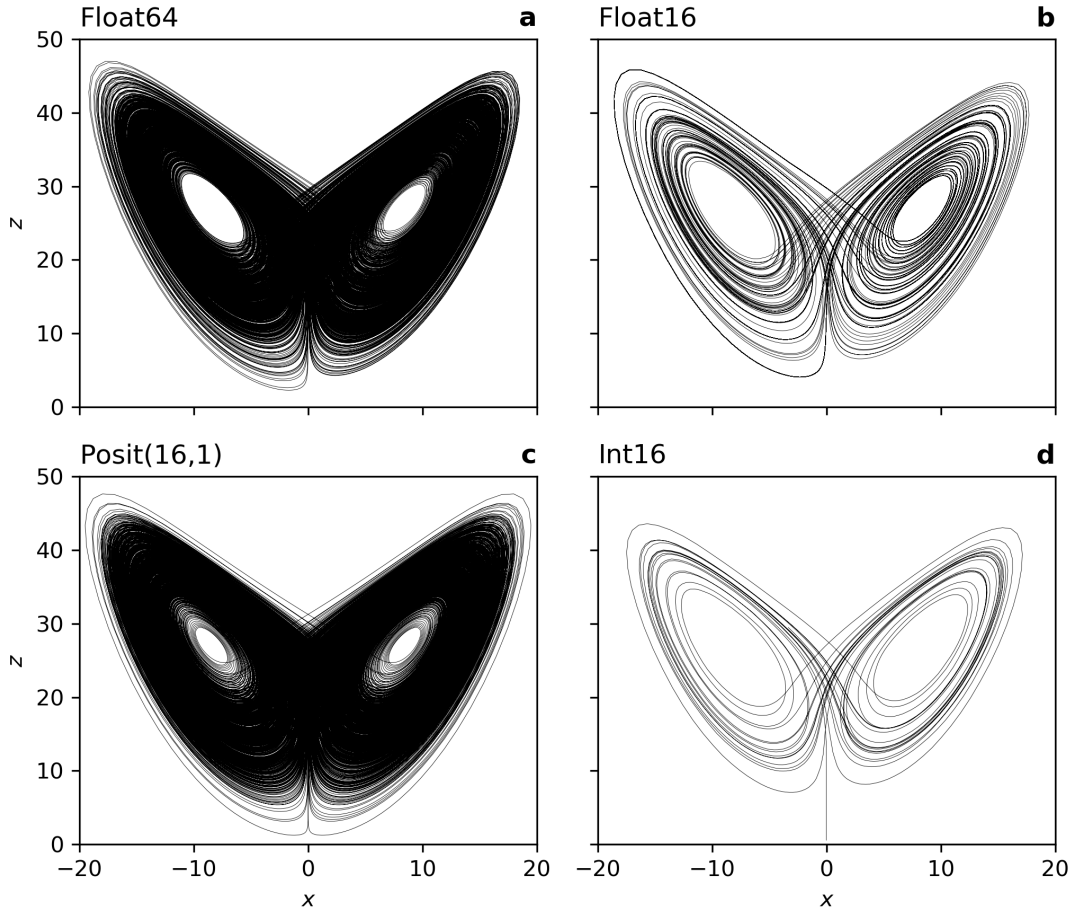


Figure 3.1: The Lorenz attractor computed with different arithmetics and precision. (a) 64bit double precision floats, (b) 16-bit half precision floats, (c) 16-bit posits with 1 exponent bit and (d) 16-bit integers. The scaling of the Lorenz equations (Eq. 3.2) is (a,b) $s = 1$, (c) $s = 0.1$ and (d) $s = 100$. All curves are integrated for the same number of time-steps.

(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. 3.1d).

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 Shallow water model

4.1 Methods

This section will evaluate the different number formats (16bit half precision floats, 16bit 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

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} + f\hat{\mathbf{z}} \times \mathbf{u} = -g\nabla\eta + \mathbf{D} + \mathbf{F} \tag{4.1a}$$

$$\frac{\partial \eta}{\partial t} + \nabla \cdot (\mathbf{u}h) = 0. \tag{4.1b}$$

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. 4.1b 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 $\times$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. 4.1, is presented in Appendix A.1. The shallow water equations are discretized 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, 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

$$\frac{\partial q}{\partial t} + \mathbf{u} \cdot \nabla q = 0. \tag{4.2}$$

We discretise Eq. 4.2 with a semi-Lagrangian advection scheme [Diamantakis, 2013; 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. 4.2 is therefore turned into an interpolation problem. Further details concerning the semi-Lagrangian advection scheme and its reformulation into a non-dimensional relative coordinate form to work with 16bit arithmetics is discussed in Appendix A.2.

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. 2.4a). 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. 4.1 and 4.2 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 being large for geophysical flows, we need to use dimensionless Nabla operators $\tilde{\nabla} = \Delta\nabla$. The continuity equation Eq. 4.1b, for example, is discretised with an explicit time integration method as

$$\eta^{n+1} = \eta^n + RK_\eta \left(-\tilde{\nabla} \cdot (\mathbf{u}h)^n \right) \tag{4.3}$$

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 A.1 for further details and a discussion of the dissipation term $\mathbf{D}$). The entire numerical integration is performed using the various 16bit number formats. However, posits are converted back to single precision floats for model output. 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. Further details of the numerical integration of the shallow water

equations with respect to 16bit arithmetics are provided in Appendix A.1.

4.2 Results

The results will be discussed in four parts: Snapshots of single forecasts are used to qualitatively assess the rounding errors of 16bit floats and posits. Ensemble forecasts yield a quantitative measure for the expected contribution of rounding errors to the forecast error. The impact of rounding errors on the climatological mean and variability is discussed subsequently and finally histograms yield an insight in the problematic terms that are prone to rounding errors.

Single forecasts 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. 4.1) stirring and mixing can be well simulated with half precision floats and with 16bit posits (2 exponent bits). However, the half precision simulation (Fig. 4.1c) deviates much faster than the posit simulation (Fig. 4.1b) from the double precision reference (Fig. 4.1a). 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. 2.4a).

To provide evidence that the approaches taken here to allow 16bit simulations also work on larger grids, we ran a single forecast on an 800x400 grid again at a resolution of $\Delta = 10$km. The domain covered therefore spans 8000km in zonal and 4000km in meridional direction and resembles a highly idealized circumpolar current. Although the wind forcing is kept the same, we change the bottom topography slightly to create more shear instabilities in the current. Additionally to the meridional ridge at $x = \frac{L_x}{2}$ three additional ridges are placed at $x = 0, \frac{L_x}{4}$ and $\frac{3L_x}{4}$. The zonal positions of these ridges are displaced a few grid points to inhibit a solution that is periodic with respect to the meridional ridges. The simulation is spun-up to statistical equilibrium without solving the tracer equation. After the spin-up phase the initial condition of the tracer concentration (which we call temperature, although still a passive tracer) is prescribed by a hyperbolic tangent in the meridional direction. The initially zonally constant temperature profile is therefore immediately stirred by an already turbulent underlying flow field. This single forecast is performed three times, using (i) 64 bit double precision arithmetic, (ii) 16bit half precision floats and (iii) 16bit posits with 2 exponent bits.

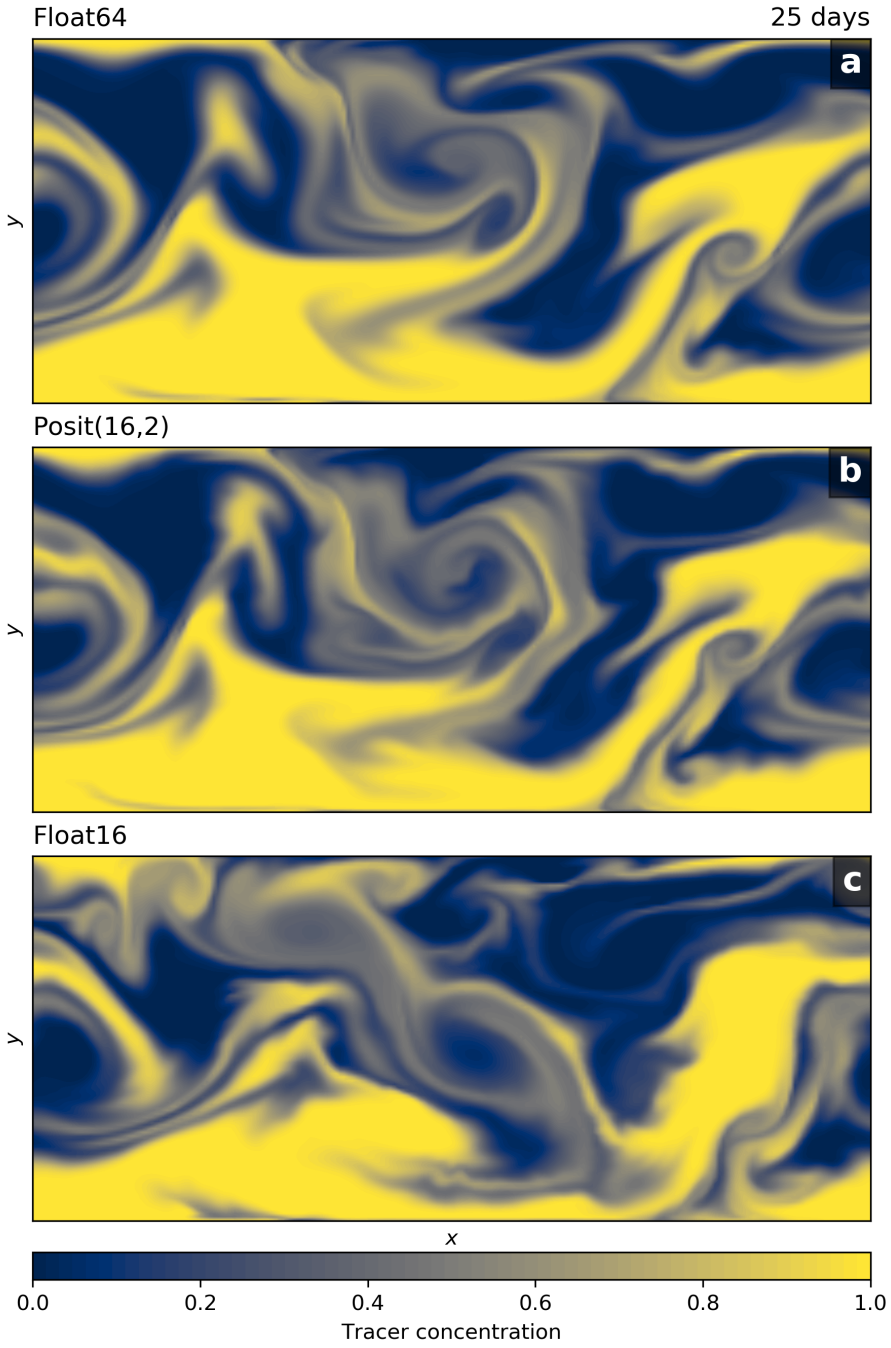


Figure 4.1: Snapshot of tracer concentration simulated by the shallow water model, based on (a) double precision floats and (b) posit arithmetic (16 bit with 2 exponent bits) and (c) half precision floats. The tracer was injected uniformly in the left half of the domain 25 days before. This simulation was run at a resolution of $\Delta = 10$km (200x100 grid points). The corresponding video can be found at http://milank.de/videos/swm_posit_tracer.mp4

Snapshots of temperature at different time steps after initialisation (Fig. 4.2 and Fig. 4.3) as well as Fig. 4.4 indicate that 16bit simulations can provide useful forecasts. 16bit posits with 2 exponent bits clearly outperform half precision floats and closely resemble the forecast based on 64bit double precision floats. However, some spurious features, that are likely instabilities occurring due to rounding errors, are visible for 16bit posits (Fig. 4.3b). These instabilities are not caused by the semi-Lagrangian advection scheme but are also apparent in the underlying flow field. We assume that they can be traced back to rounding errors associated with the sum of tendencies as outlined in Appendix A.3, but a thorough analysis is needed to support or disprove this assumption. These simulations provide evidence, that 16bit numbers are indeed promising to calculate the semi-Lagrangian advection scheme once written in a non-dimensional and relative coordinate formulation (see Appendix A.2 for details).

Ensemble forecasts Limited by the computational requirements to run an ensemble of forecasts with software-emulated posits we consider again a shallow water model of a small domain (2000 km $\times$1000 km), discretised with only 100x50 grid points. The forecast error in the shallow water model 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 errors 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 & 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. 4.4), with a small accumulation of rounding errors even for lead times of 100 days. The forecast error for 16bit posits without exponent bit increases quickly (Fig. 4.4), 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

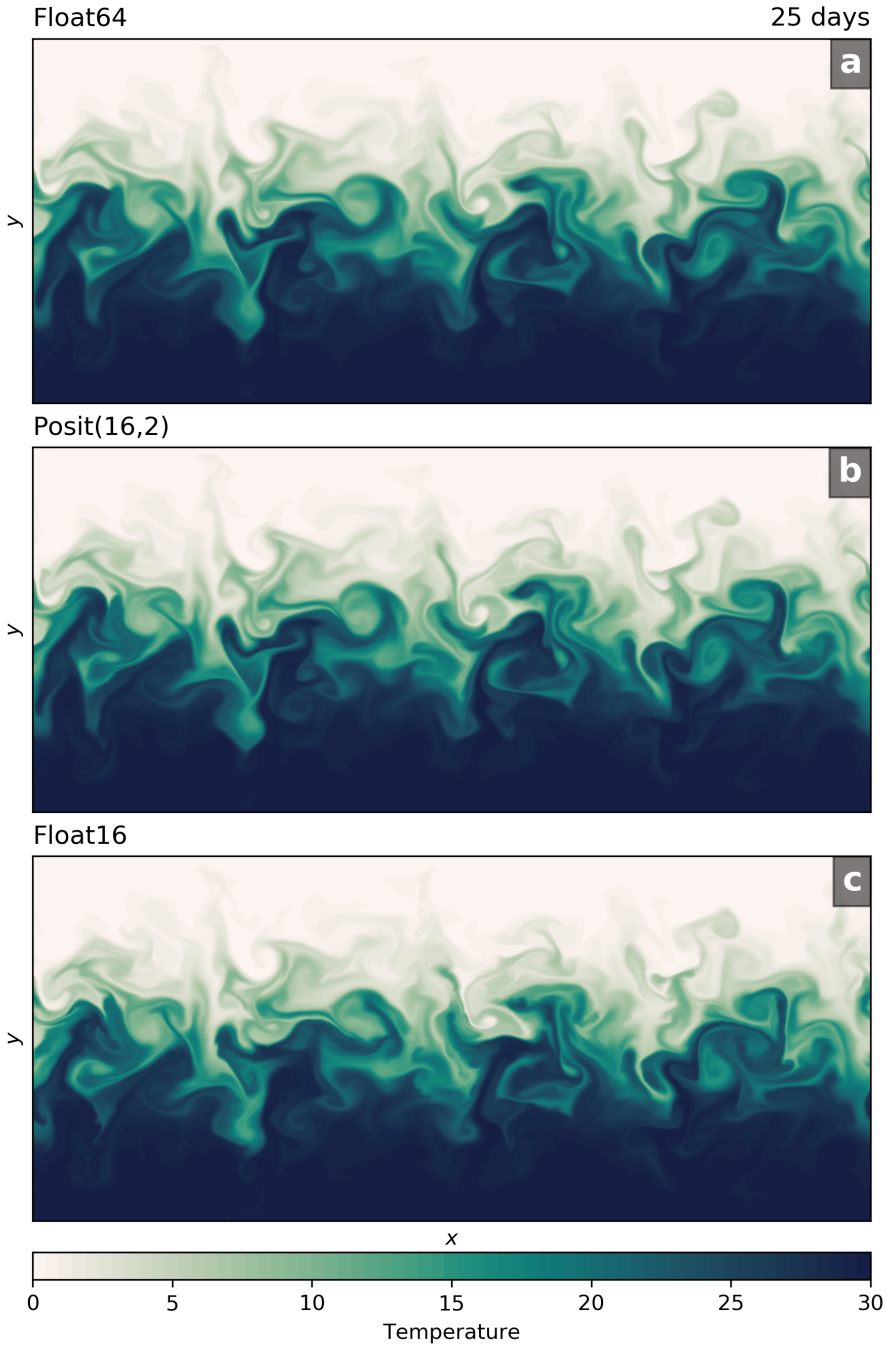


Figure 4.2: Snapshot of temperature (used as passive tracer) simulated by the shallow water model, based on (a) double precision floats and (b) posit arithmetic (16 bit with 2 exponent bits) and (c) half precision floats. This simulation was run at a resolution of $\Delta = 10$km (800x400 grid points). The corresponding video can be found at http://milank.de/videos/tracer_posit_hr.mp4

Figure 4.3: As Fig. 4.2 but 50 days after model initialisation.

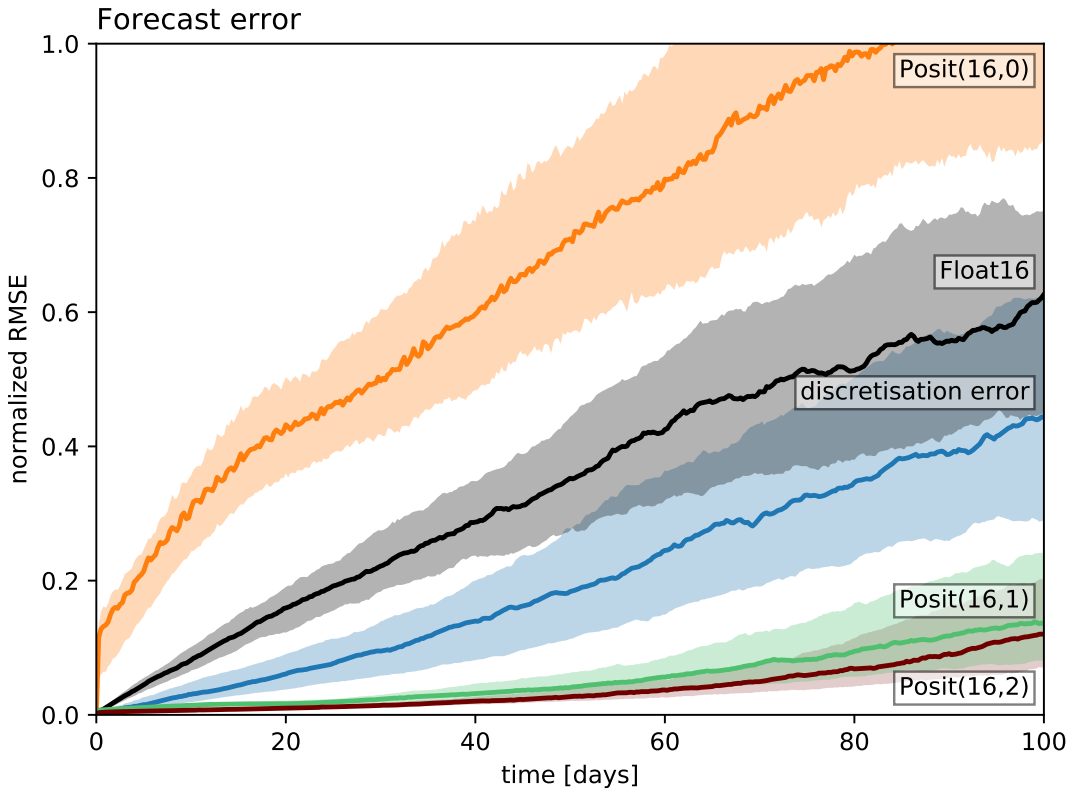


Figure 4.4: Forecast error measured as the root mean square error (RMSE) of sea surface height taking the double precision forecast as reference. The RMSE is normalised by a mean forecast error at very long lead times. Solid lines represent the median of 280 forecasts per number format. The shaded areas denote the interquartile range.

16bit 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. 2.4a). The forecast error of half precision floats is larger than the discretisation error.

Climatological mean & variability A useful forecast model does not necessarily have an accurate representation of the reference mean state. A systematic error in the model integration can lead to a bias in the mean state or the variability around the mean state, although of minor importance for short-range forecasts. In order to investigate the effect of reduced precision arithmetics on mean and variability, we average the sea surface height of the forecast ensemble over all time steps and similarly for variability, which is computed as the standard deviation of sea surface height. Again, the integration with double precision floats is regarded as reference truth, which shows a zonal current that is deviated to the south when it passes over the ridge in the bottom topography (Fig. 4.5a). The error, computed as difference to double precision floats, with 16bit posit arithmetic is clearly negligible (Fig. 4.5d and e), whereas the simulation based on half precision floats shows a less pronounced trough and ridge in the sea surface height, indicative of a weaker mean current (Fig. 4.5b). A similar conclusion can be drawn for variability (Fig. 4.6): 16bit posits (1 or 2 exponent bits) do not alter the mean nor the variability of the 64bit double precision simulation. However, large forecast errors of half precision floats coincide with errors in mean and variability and are on the same order of magnitude as the discretisation error. We assume that 16bit posits without exponent bits suffer from the no overflow no underflow rounding mode, as tendencies that are smaller than the smallest representable number are round away from zero and therefore increase in magnitude. An increase in the variability follow as shown in Fig. 4.6f.

Histograms To understand the limitations of the respective number systems to solve the shallow water equations we compare histograms of numbers that occur in various terms of the model integration with the decimal precision of floats and posits (Fig. 4.7). As mentioned earlier, the prognostic variables u, v, η are all $\mathcal{O}(1)$, which is optimal for the posit number system. The sum of the tendencies is approximately 4 orders or magnitude smaller, but still sufficiently represented by the 16bit number systems (except for 16bit posits without exponent bits, which likely explains their poor performance). We conclude that especially the sum of the tendencies and the diffusive terms (Biharmonic diffusion and bottom friction) deserve attention when further optimizing the model code to avoid intermediate results that are poorly representable by 16bit number formats. Departure points that were calculated by the semi-Lagrangian advection scheme in its

non-dimensional and relative coordinate formulation (see Appendix A.2) are well centred around 1, supporting the this formulation for low precision numbers. The momentum advection and Coriolis terms are hidden behind the histogram of the pressure gradient term, pointing towards the dominance of the geostrophic balance in the shallow water model.

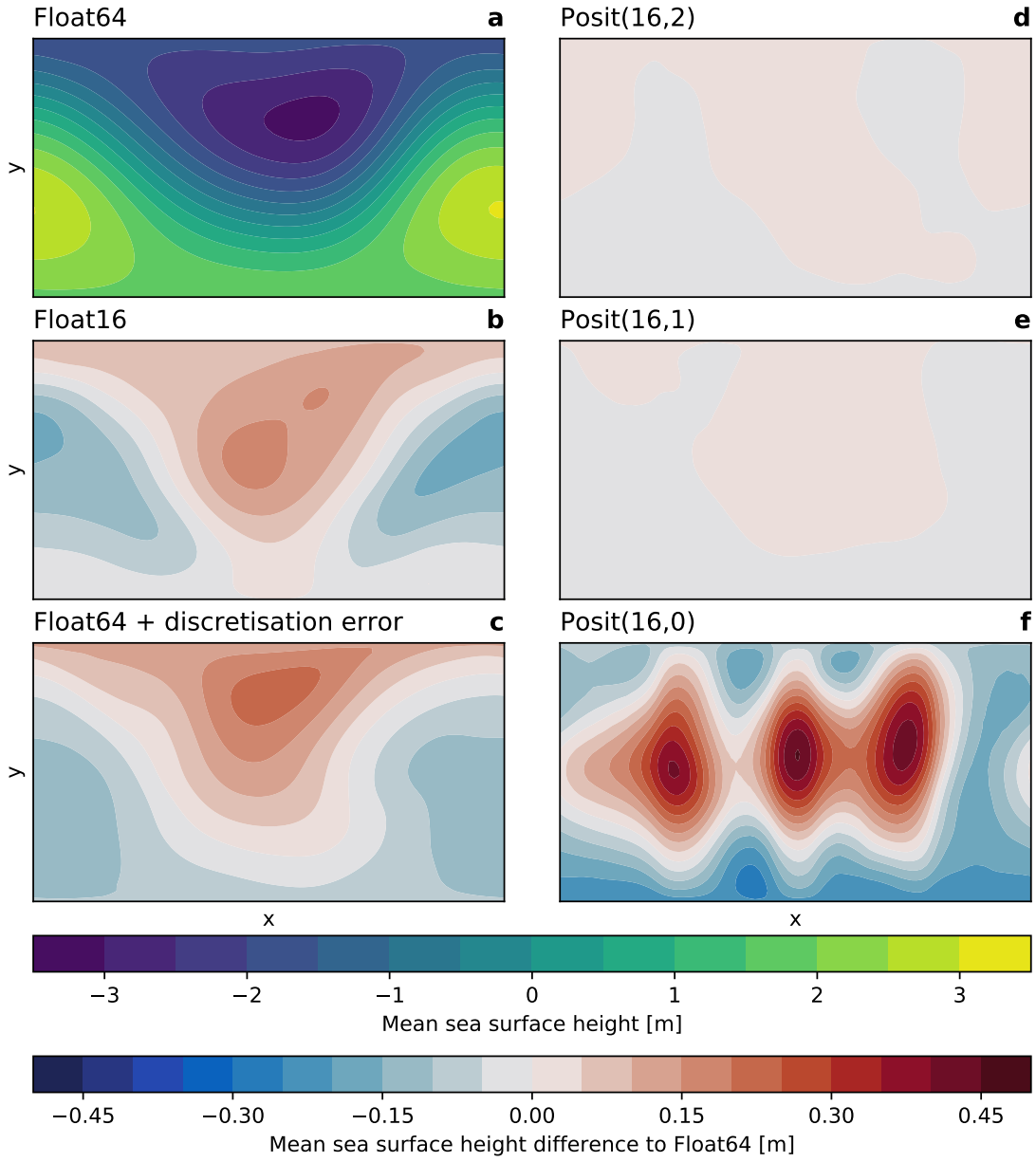


Figure 4.5: Climatological mean circulation as indicated by the mean sea surface height simulated by the shallow water model. (a) Reference mean field computed with double precision floats; (b–f) difference to the reference. (b) half precision floats; (c) double precision floats plus discretisation error; (d) 16bit posits with 2 exponent bits; (e) 16bit posits with 1 exponent bit and (f) 16bit posits without exponent bits. The difference to the reference (b–f) - (a) is also indicated by a different colour map.

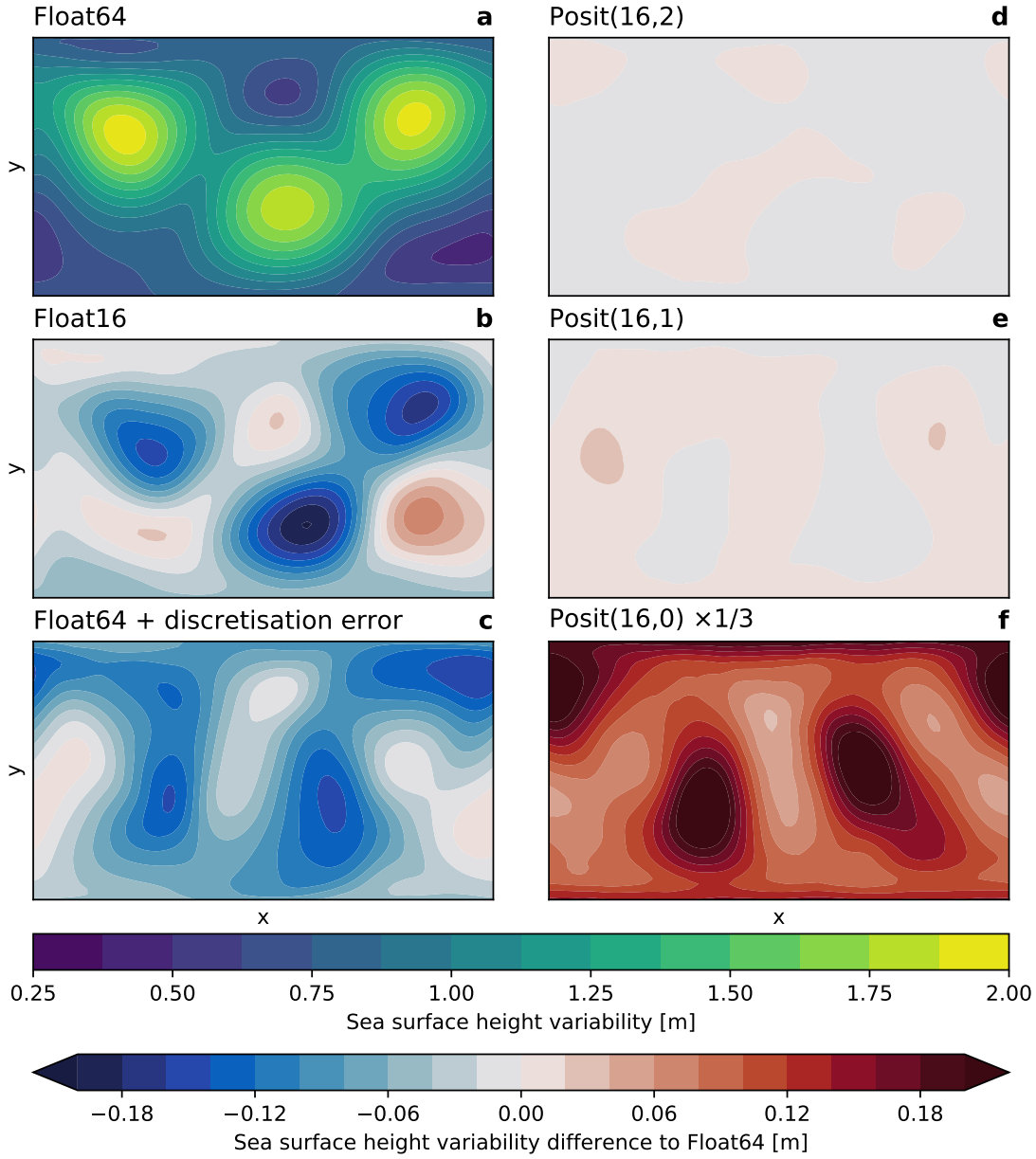


Figure 4.6: Same as Fig. 4.5 but for climatological variability computed as standard deviation from the mean. Note that (f) was scaled down by a factor of $\frac{1}{3}$ to match the colour range.

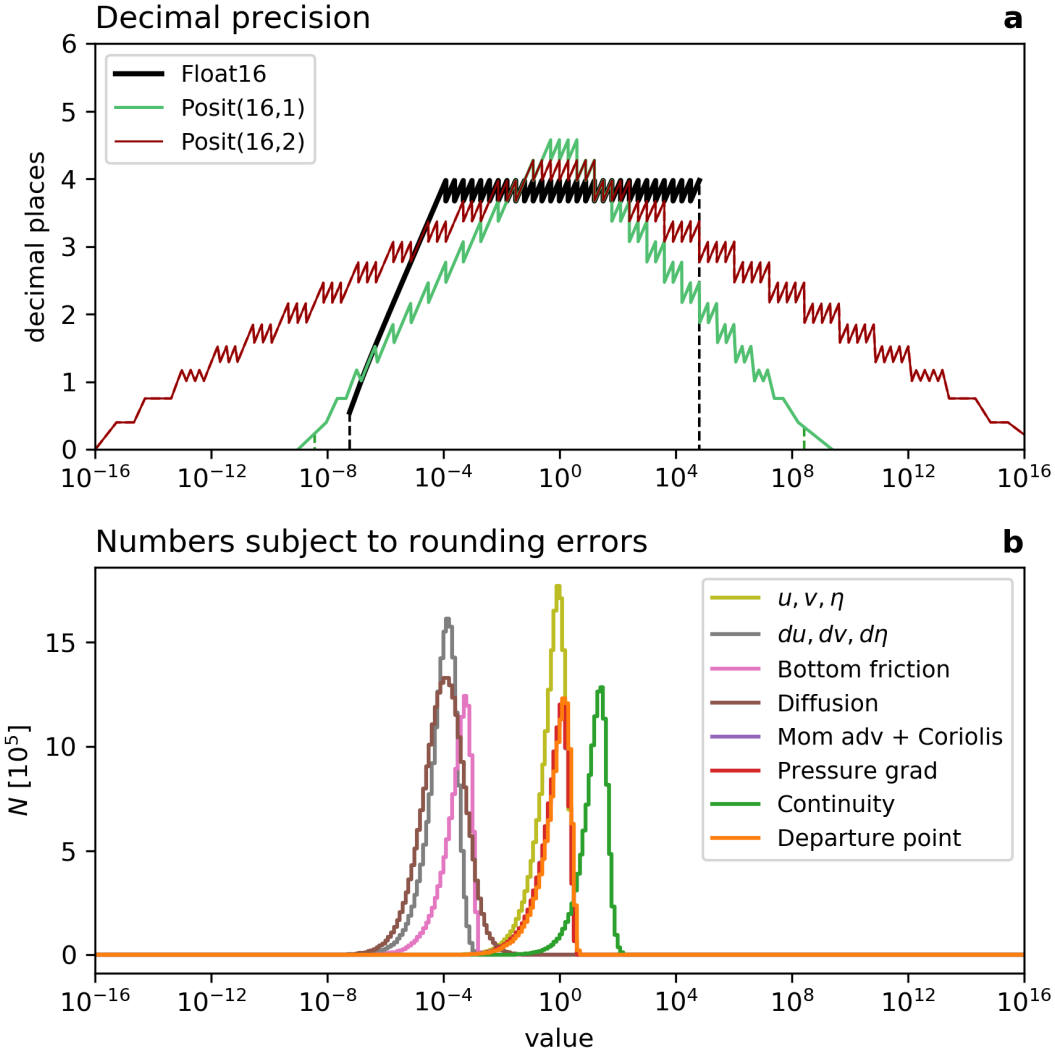


Figure 4.7: Histogram showing the numbers occurring in the shallow water model simulation. (a) The decimal precision of Fig. 2.4a for comparison (b) Numbers subject to rounding errors in various parts of the model integration: u, v, η are the prognostic variables, $du, dv, d\eta$ are the sum of the tendencies including the time step. Various other terms as indicated in the legend. Departure points are computed by the semi-Lagrangian advection scheme in the non-dimensional relative coordinate formulation. The momentum advection and Coriolis term histogram is hidden behind the pressure gradient term histogram.

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 16bit posits with one exponent bit when compared to 16bit 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, 16bit posits with 1 or 2 exponents clearly outperform floats and appear very promising for application in high performance computing for Earth System modelling. Especially 16bit 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 16bit 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 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 16bit arithmetic would be dwarfed by errors in initial conditions or discretisation errors in many applications. As the forecast error with 16bit posits (1 or 2 exponent bits) are still considerably lower than the discretisation error, this suggest 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 16bit 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 32bit 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 32bit posit formats with 2 exponent bits.

6 Outlook

In the following projects for future work are discussed. Preferences and expected time investments are discussed in the individual subsections.

6.1 Hardware emulation of posits with FPGAs

Section 3 and 4 are entirely based on the software emulation of posits via the Julia-based emulator. Software emulation provides a precise estimate of the numerical precision that can be expected in a simulation. Emulating posit arithmetic in software comes with the advantage that the same model code can be used with only small adjustments to specify the desired number system. The shallow water model here can be executed on a conventional CPU with either floating-point arithmetic or posit arithmetic at various configurations by changing a single flag in the model parameters. This is a convenient approach to test the effect of different binary number formats on metrics like forecast accuracy or mean state. However, this approach comes with the disadvantage that every number system that is not supported by the hardware runs at drastically reduced speed. The Julia-based emulator for 16bit posits is approximately 150x slower than 64bit double precision floats and 300x slower than 32bit single precision floats (which are both supported by the CPU). We can therefore only assume that 16bit posits will be faster than 32bit single precision floats, but we cannot provide any evidence for this. The assumption is justified, as 16bit arithmetics require less data to be transferred on hardware. The microprocessor architecture is simplified for posits compared to floats [Chaurasiya *et al.*, 2018; Chen *et al.*, 2018; Glaser *et al.*, 2017; van Dam, 2018], which suggests reduced area microprocessors and lower energy consumption. For posits, many arithmetic operations can be performed at reduced clock cycles, which provides further potential to an increased speed of posit arithmetic compared to floating-point arithmetic. Unfortunately, no posit processor (also called posit numerical unit, PNU) with a comparable performance exists yet, as most PNUs are currently based on field-programmable gate arrays (FPGA). An FPGA is a device that serves as an integrated circuit which can be modified by the user after manufacturing. A hardware specific programming language allows to programme the desired circuits into hardware, which can serve as a microprocessor prototype. FPGAs are very versatile but in general do not match the speed of a CPU, unless very specific algorithms are entirely moved onto an FPGA. Nevertheless, a PNU simulated by an FPGA could serve as a great testing environment for posit arithmetics. This form

of hardware acceleration would provide more experience with the details of hardware supported posit arithmetics and could also yield insights into quires (see section A.3). An FPGA PNU could support the fusion of certain arithmetic operations, such that their impact on the model fidelity can be investigated. It is not clear yet which set of operations can be fused into a single quire, but for more complicated combinations of operations also a two-step quire computation could be feasible.

This research project is in collaboration with the groups of Zaid Al-Ars and Peter Hofstee at TU Delft and currently in the planning phase. We are going to provide the software and Delft is going to work on the hardware. Therefore, computer scientists with expertise in FPGAs and processor design are going to implement the performance crucial parts of the algorithms into hardware and we will work on the algorithm that is best suited to be implemented on an FPGA. Collaborators at Delft have already implemented posit arithmetic onto FPGAs for matrix-matrix multiplies, with promising results [Chen *et al.*, 2018; van Dam, 2018]. Currently, the shallow water model is formulated with element-wise matrix-matrix operations, but large parts could also be written as sparse matrix-vector multiplications. The exact algorithm will be developed in collaboration with Delft to allow an efficient posit hardware emulator. This project is expected to be less time consuming from our side, and will likely be a Master's thesis for a student at Delft.

6.2 Increasing model complexity and parallelization

The following project aims to increase the complexity of the circulation model to test posit arithmetic in a more realistic weather or climate model application. This can either be conducted by (i) starting with the shallow water model presented in section 4 and subsequently increase the model's complexity by adding features, or (ii) using an already existing global atmospheric model that ideally is written in the Julia language to facilitate the transition of the posit software emulator.

A layered primitive equation model

The 3D Navier-Stokes equations describe the temporal evolution of the atmosphere and ocean. However, valid approximations for atmospheric and oceanic circulation simplify this set of equations into layered 2D primitive equations with hydrostatic approximation that are currently used for operational weather forecast. There are several competing approaches to solve the 2D primitive equations on a sphere, such as finite differences, finite

volume, finite element and discontinuous Galerkin as well as spectral methods, which are combined with explicit or implicit time stepping schemes. Although the underlying equations are the same, these various numerical techniques require different algorithms and pose a different problem for computing architecture to compute the dynamical core of a weather or climate model. The shallow water model presented in section 4 serves as a simplified 2D finite difference dynamical cores of some state-of-the-art weather and climate models. The approaches presented here to reformulate the discretised version of the equations of motions to be executable with 16bit float or posit arithmetics are therefore not necessarily transferable to other algorithms, despite the similarity of the underlying equations.

To increase the complexity of the shallow water model it is therefore suggested to add additional layers and to turn the shallow water equations into primitive equations. From a physical point of view this includes processes like baroclinic instability, but also an active tracer advection, which relaxes the assumption of a homogeneous density and allows the tracers to influence the flow field by introducing pressure gradients from density gradients (e.g. thermal wind). From a computational point of view, the coupling of layers involves a vertical integral of quantities, which orders of magnitude could considerably change from layer to layer. This would complicate the rescaling of the variables and it is a priori not clear, whether these algorithms cause problems to be executed with 16bit posit or float arithmetics. One advantage of the physical set-up of the here considered shallow water model is, that the prognostic variables are all $\mathcal{O}(1)$. Regarding a 2-layer stacked shallow water model the surface variables are again $\mathcal{O}(1)$, however, velocities in the lower layer are usually $\mathcal{O}(10^{-2}\,\mathrm{ms}^{-1})$ and the interface displacement $\mathcal{O}(10^{2}\,\mathrm{m})$. An interesting continuation of the present study would be therefore to explore the scaling possibilities in such a system, where different variables occupy different ranges in the provided range of representable numbers of the given number system. Ideally, each layer could be scaled to shift every prognostic variable back into a range of $\mathcal{O}(1)$. However, due to the non-linear terms, some of these scalings need to be undone when evaluating those terms. Additionally, different scalings of different layers need to be carefully considered when coupling the layers. Whether this can be done efficiently to yield a useful layered primitive equation model in 16bit, that performs well in forecast metrics as presented here for the shallow water model, remains an open question.

Converting the single layer shallow water model of section 4 into a 2-layer model is a project with an estimated time investment of a few months. Converting the shallow water equations into primitive equations is likely a similar time effort. These projects will be explored in the near future but not at high priority.

Distributed memory communication in 16bit with MPI

The shallow water model presented in section 4 runs on a single processor, i.e. we have not parallelized its integration among several processors. In general, grid point models are parallelized with a technique called *domain decomposition*. Instead of a single processor computing the right-hand side of the equations for the entire domain, the domain is decomposed into several subdomains, each computed by one processor. As long as the domain is large enough, this would in general allow a strong scalability, i.e. the runtime is inversely proportional to the number of processors used. However, the subdomains are coupled such that the boundary conditions computed by one domain have to be communicated with the neighbouring subdomains. This introduces overhead and the scalability depends on many details of the implementation. The communication of boundary conditions guarantees that information can propagate through the domain as if a single processor was used for the entire domain. In fact, a domain decomposition does not inhibit a bit-wise reproducibility. The communication of boundary conditions is usually implemented via ghost points. Ghost points extend every subdomain by a few surrounding grid points (depending on the stencil size), such that there is a small overlap between neighbouring subdomains. At the beginning of each time step each processor needs to copy the values at the boundaries into the arrays of the prognostic variables owned by the neighbouring processes. This communication is done via MPI (Message Passing Interface), a widely used standard to send data from one processor with dedicated memory to another with independent memory allocated. MPI is therefore used for the parallelization with a distributed memory paradigm, which is in contrast to shared memory communication, usually done via the OpenMP protocol. In most global high resolution weather and climate models, both MPI and OpenMP are used together for optimal performance on large supercomputers.

MPI communication can add a considerable amount of time to the overall runtime of a weather and climate model [Müller *et al.*, 2019]. Based on the assumption that MPI communication can be executed faster when less data is communicated it is therefore an interesting project to investigate whether a model simulation can be performed without decrease in fidelity when data is not communicated in 64bit numbers but in 16bit numbers (or even 8bit). The rounding therefore does not occur on the computation of the right-hand side, but every processor only receives a reduced precision version of the boundary conditions from the neighbouring subdomains. A great potential speed-up can be expected, when weather or climate models are communication-bound, i.e. the limiting factor of increasing the model performance is the communication between all

processors. Reducing the data communicated could therefore facilitate a solution to this limiting factor.

The shallow water model presented in section 4 is formulated in a way that the boundary conditions are implemented as ghost points. On every time step, the respective boundary conditions are therefore copied into the ghost points. The change towards a domain decomposition is therefore minimal as this copying process needs to be extended to allow for MPI communication, such that not one processor copies entries from one array to the other but from one array on one process to another array owned by another process. The Julia language supports the MPI standard via the MPI.jl wrapper. Unfortunately, the MPI standard does not yet allow the communication to be performed in 16bit, but we could emulate the effect of reduced precision communication via down and upcasting just before (or after) the communication is performed. We assume that the precision reduction potential is even greater as only the precision of prognostic variables is reduced that are typically shared between processors. This reduction does not generate any complication with intermediate results, that occur when reducing the precision for all computations of the right-hand side. When the range of numbers of prognostic variables that are communicated via MPI is limited, it might be even feasible to communicate 8bit numbers, but this remains to be investigated.

As a first step a low-precision MPI-communication can be implemented in the shallow water model presented in section 4, which is expected to be a minor project of a month. As Julia is indeed a competitive alternative to established high performance computing languages such as Fortran, it is worthwhile investigating whether Julia code can run efficiently in parallel without much programming effort. As a second step the same low-precision MPI-communication idea can be applied to global atmospheric models such as the Integrated Forecast System (IFS), although this will likely rule out the possibility for a posit-encoded communication, as no posit emulator is available for Fortran. Furthermore, this is a bigger project, expected to take a few months.

6.3 Information theory approach to reduced precision

Developing a 16bit shallow water model therefore followed partly a try-and-error approach, where various changes in the model code were tested to work with 16bit and eventually rewritten or improved. This progress was largely supported by solving the equation with SI units and with finite differences. Both provided the advantage that it was comparably easy to estimate roughly the order of magnitude of intermediate calculations. Consequently, bottlenecks with respect to 16bit arithmetics could be identified and

improved. Weather and climate models of higher complexity, especially in terms of their discretisation methods, likely complicate this approach. Identifying precision bottlenecks might become the limiting factor of low precision model development with increasing model complexity. A systematic diagnostic to automate the identification of precision-crucial parts is needed. Based on such a diagnostic, necessary precision levels for various calculations could be identified and in general a more flexible potentially heterogeneous precision design suggested.

Research based on information theory approaches to reduced precision [Jeffress *et al.*, 2017] could provide diagnostics for the bit-wise information content of variables. Ideally, such a diagnostic could estimate the necessary precision for algorithms and reduce the use of a time-consuming try-and-error approach. These diagnostic could help to identify the bottlenecks in the model code towards 16bit in more complex models, and also assess how much precision an algorithm requires – solely based on a single model run, with output from various prognostic and diagnostic variables. For dynamical core development this information theory approach, however, could be limited, as illustrated in the following example: Mathematically, the variables $x = (a + b) + c$ and $y = a + (b + c)$ are identical, however, with finite precision arithmetics the computation of $(a+b)$ introduces a rounding error and worst case an overflow, such that x and y can not expected to be identical. Estimating the bit-wise information content based on x therefore might be misleading, as an intermediate computation might require a higher precision or a larger dynamic range than x itself. Although promising for data storage, this illustrated issue might limit information theory approaches to identify reduced precision potential for dynamical cores of weather and climate models. It is therefore suggested to start this project with an observational data set and investigate the bit-wise information content of a given variable to predict another variable at another location. This approach can yield insights in the information content of geophysical data and hopefully identifies bits that do not contain any real information. In general we expect the information content to change depending on dominant time scale and the instabilities in the flow field, but this remains to be investigated. Preliminary work was able to reproduce the study by Jeffress *et al.* [2017], such that the next step would be to extend their methods for datasets of higher dimensionality. The project is assumed to require a time investment of a few months for major results.

Appendix

A.1 A 16bit 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 30°N 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 partial slip at the northern and southern boundary. The layer thickness is $h = \eta + H(x)$, with

$$H(x) = H_0 - H_1 \exp\left(-H_\sigma^{-2}(x - \tfrac{L_x}{2})^2\right) \tag{A.1}$$

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, travelling 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

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

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

$$\mathbf{D} = -r\mathbf{u} - \nu\nabla^4\mathbf{u} \tag{A.3}$$

of a linear bottom drag with timescale $r^{-1} = 300\,\mathrm{days} \approx 2.6 \times 10^7\,\mathrm{s}$ 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].

16bit formulation To avoid division and subsequent multiplication with large numbers throughout the numerical model integration, we reformulate the shallow water equations

(Eq. 4.1), such that every substep of the Runge-Kutta time scheme reads as

$$\mathbf{u}^{n+1} = \mathbf{u}^n + RK_{\mathbf{u}} \left(-(\mathbf{u}^n \cdot \tilde{\nabla})\mathbf{u}^n - \tilde{f}\hat{\mathbf{z}} \times \mathbf{u}^n - g\tilde{\nabla}\eta^n + \tilde{\mathbf{D}} + \tilde{\mathbf{F}} \right) \tag{A.4a}$$

$$\eta^{n+1} = \eta^n + RK_{\eta} \left(-\tilde{\nabla} \cdot (\mathbf{u}h)^n \right) \tag{A.4b}$$

with $RK_{\mathbf{u}}, RK_{\eta}$ being the Runge-Kutta coefficients times $\frac{\Delta t}{\Delta}$; the rescaled Coriolis parameter is $\tilde{f} = \Delta f$; and the rescaled wind stress forcing $\tilde{\mathbf{F}} = \Delta \mathbf{F}$, all precomputed at high precision in the model initialisation. The rescaled dissipation term $\tilde{\mathbf{F}}$ is

$$\tilde{\mathbf{D}} = -\tilde{r}\mathbf{u} - \tilde{\nu}\tilde{\nabla}^4\mathbf{u} \tag{A.5}$$

with $\tilde{r} = r\Delta \approx 0.0008\,\mathrm{ms}^{-1}$, 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.

The following time-split approach is in this study only used for the simulations of Fig. 4.2 and 4.3, but we want to emphasize the importance of time-split approaches. They can be important to increase the size of the tendencies, to decrease the rounding error. Instead of solving Eq. A.4 fully explicit with the same time step for each term, we can split the time steps for various terms depending on their numerical stability Shchepetkin & McWilliams [2005]. Fast gravity waves should be resolved, which requires the pressure gradient term and the continuity equation to be solved at the time step Δt. The diffusive terms $\mathbf{D}$, i.e. biharmonic diffusion and bottom friction, vary slower in time and do not require such a short time step for stability. Therefore they can be solved with a larger time step. Motivated by widely applied ocean models like NEMO [Madec, 2016] and ROMS [Shchepetkin & McWilliams, 2005], a possible approach is to solve the diffusive terms similar to the semi-implicit Euler method, which means that only after every mth time step diffusion will be evaluated with the current state of the prognostic variables $\mathbf{u}_*^{n+1}$, adding the diffusive terms then results in an updated prognostic state $\mathbf{u}^{n+1}$. We first step the non-diffusive equations forward m time steps with Δt, such that Eq. A.4a simplifies to

$$\mathbf{u}_*^{n+1} = \mathbf{u}^n + RK_{\mathbf{u}} \left(-(\mathbf{u}^n \cdot \tilde{\nabla})\mathbf{u}^n - \tilde{f}\hat{\mathbf{z}} \times \mathbf{u}^n - g\tilde{\nabla}\eta^n + \tilde{\mathbf{F}} \right) \tag{A.6}$$

After every mth time step we solve

$$\mathbf{u}^{n+1} = \mathbf{u}_*^{n+1} + \frac{\Delta t_{\text{diff}}}{\Delta}\left(-\tilde{r}\mathbf{u}_*^{n+1} - \tilde{\nu}\tilde{\nabla}^4\mathbf{u}_*^{n+1}\right) \tag{A.7}$$

with $\Delta t_{\text{diff}} = m\Delta t$ being the enlarged time step for diffusive terms. We use $m = 5$ for our physical setting, however, re-tuning might be necessary for different physical parameter values. A semi-implicit scheme for the diffusive terms comes with the advantage of decreased runtime, due to fewer evaluations of these terms. With respect to 16bit arithmetics, a semi-implicit scheme increases the size of the diffusive tendencies by a factor of m to $6m$ (the smallest Runge-Kutta coefficient is $1/6$ for the fourth order scheme). As the diffusive tendencies are several orders of magnitude smaller than the prognostic variables (Fig. 4.7), this is advantageous to avoid rounding errors (see Appendix A.3 for a further discussion).

A.2 A semi-Lagrangian advection scheme for 16bit

The semi-Lagrangian advection scheme is based on the idea to solve the advection equation with respect to its Lagrangian formulation [Diamantakis, 2013; Smolarkiewicz & Pudykiewicz, 1992]. In the absence of sources and sinks, the Lagrangian point-of-view states that the tracer concentration q does not change following its trajectory. The concentration q at a given departure point $\mathbf{x}_d$ is therefore the same as the concentration at the arrival point $\mathbf{x}_a$ later, which is reached by integrating the flow field over time along the trajectory of infinitesimal arrival points

$$\mathbf{x}_a = \mathbf{x}_d + \int_t^{t+\Delta t} \mathbf{u}(\mathbf{x}_a, t)\,\mathrm{dt} \tag{A.8}$$

For a turbulent flow the trajectories of a given set of departure points cross and therefore complicate the geometry of the underlying grid considerably. To avoid a change in the grid geometry, we set the arrival points to be identical to the centre points of the Arakawa C-grid of the discretised shallow water equations. The departure points however, will in general not coincide with that same grid, and we make use of interpolation to find the tracer concentration at arbitrary departure points. Hence, the semi-Lagrangian advection scheme can be split into two parts, (i) find for every arrival point the corresponding departure point and (ii) interpolate the tracer concentration onto the coordinate of the departure point (which is going to be the same concentration

at the arrival point). A discrete form of Eq. A.8 is

$$\mathbf{x}_d = \mathbf{x}_a - \mathbf{u}(\mathbf{x}_a, t + \Delta t_{\text{adv}})\Delta t_{\text{adv}} \tag{A.9}$$

where we used the velocity $\mathbf{u}$ at the arrival point and at the arrival time. The time step is denoted as Δt_{adv} as it is in general different from the time step Δt that is used to integrate the shallow water equations. The semi-Lagrangian advection scheme is unconditionally stable and therefore allows for very large time steps. Large time steps are desired as (i) the computational cost is reduced and (ii) the lower total number of interpolations reduces the numerical tracer diffusion. We can increase the accuracy with an iterative method that uses several steps with interpolations of the velocity field onto intermediate points of the trajectory. In fact, we use a two-step method that can, with a mid-point $\mathbf{x}_m$, be written as

$$\mathbf{x}_m = \mathbf{x}_a - \mathbf{u}(\mathbf{x}_a, t)\frac{\Delta t_{\text{adv}}}{2} \tag{A.10a}$$

$$\mathbf{x}_d = \mathbf{x}_a - \mathbf{u}(\mathbf{x}_m, t - \frac{\Delta t_{\text{adv}}}{2})\Delta t_{\text{adv}} \tag{A.10b}$$

For very long time steps Δt_{adv} it is crucial to have an accurate estimate of the departure point, as a rapidly changing turbulent flow field can otherwise lead to widely deviated departure points. Please note, that also an interpolation of the velocity $\mathbf{u}$ onto the arrival point (or intermediate points for the iterative method) is necessary due to the staggered Arakawa C-grid. This interpolation is done bilinearly, as discussed in the following for the interpolation of the tracer concentration.

Once the departure point $\mathbf{x}_d$ is found, a bilinear interpolation of the surrounding grid points onto the departure point is performed. In the unit square we define the tracer concentrations $q(x, y)$ with $x, y \in [0, 1] \times [0, 1]$ of the corner points as $q_{00} = q(0, 0), q_{01} = q(0, 1)$, etc. The bilinear interpolation follows then as

$$q(x, y) = q_{00}(1 - x)(1 - y) + q_{10}x(1 - y) + q_{01}(1 - x)y + q_{11}xy \tag{A.11}$$

such that for a given departure point $\mathbf{x}_d$ we have to deduce the relative coordinates x, y in the grid cell surrounded by the four closest tracer concentration nodes. This procedure is simplified in our case of an equidistant grid, but can be generalized to arbitrary grids.

16bit formulation We now aim to rewrite the semi-Lagrangian advection scheme to avoid computations that are problematic with 16bit arithmetics. For geophysical

simulations the departure point can be $\mathcal{O}(10^7\,\mathrm{m})$ or larger when computed in meters, which is beyond the range of representable numbers in half precision float arithmetic. To avoid this, we use instead a non-dimensional coordinate $\tilde{\mathbf{x}} = \mathbf{x}\Delta^{-1}$ to transform Eq. A.9 into

$$\tilde{\mathbf{x}}_d = \tilde{\mathbf{x}}_a - \mathbf{u}(\mathbf{x}_a, t + \Delta t_{\mathrm{adv}})\frac{\Delta t_{\mathrm{adv}}}{\Delta}. \tag{A.12}$$

Note that in practice, $\tilde{\mathbf{x}}_a$ is a set of integers describing the array indices. For the simple case of an equi-distant grid, this turns the coordinates $\mathbf{x}$ into indices $\tilde{\mathbf{x}}$ that can be readily used to access elements in the arrays of $\mathbf{u}$ and q. For a non-equidistant grid, the grid spacing Δ here could be replaced by a typical grid spacing scale, the exact value is of minor importance. 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. 4.1 ($\Delta = 10\,\mathrm{km}$) the rescaled time step is $\frac{\Delta t_{\mathrm{adv}}}{\Delta} \approx \frac{2\cdot 10^4\,\mathrm{s}}{10^4\,\mathrm{m}} = 2\,\mathrm{sm}^{-1}$ and therefore precomputed at higher precision in the model initialisation.

To use non-dimensional coordinates is sufficient for small grids. However, for a grid with 1000 grid points in one direction also $\tilde{\mathbf{x}}_d$ will approach values of $\mathcal{O}(1000)$. 16bit arithmetics have a poor resolution at these orders of magnitude: A half precision float, for example, can only represent the number 1000.5 between 1000.0 and 1001.0, which introduces a big rounding error in the departure point computation. We therefore propose to use relative non-dimensional coordinates instead. Here, the departure point coordinate is computed relative to the arrival point. We therefore set $\tilde{\mathbf{x}}_a = 0$ in Eq. A.12

$$\tilde{\mathbf{x}}_{d,rel} = -\mathbf{u}(\mathbf{x}_a, t + \Delta t_{\mathrm{adv}})\frac{\Delta t_{\mathrm{adv}}}{\Delta}. \tag{A.13}$$

and obtain an equation with all terms being $\mathcal{O}(1)$ and therefore representable with 16bit arithmetics without problems. In practice, when converting the relative departure point $\tilde{\mathbf{x}}_{d,rel}$ to an array index, the problematic computation in Eq. A.12 is executed with integer arithmetics, which can be performed without rounding errors. Once the bilinear interpolation is computed with respect to the unit square, all terms in Eq. A.11 are also representable with 16bit arithmetics, given a 16bit-conform range of values for the tracer concentration q.

The iterative two-step departure point computation from Eq. A.10 reads in a non-

dimensional relative coordinate formulation

$$\tilde{\mathbf{x}}_{m,rel} = -\mathbf{u}(\mathbf{x}_a, t)\frac{\Delta t_{\text{adv}}}{2\Delta} \tag{A.14a}$$

$$\mathbf{x}_m = \mathcal{I}(\tilde{\mathbf{x}}_{m,rel}) \tag{A.14b}$$

$$\tilde{\mathbf{x}}_{d,rel} = -\mathbf{u}(\mathbf{x}_m, t - \frac{\Delta t_{\text{adv}}}{2})\frac{\Delta t_{\text{adv}}}{\Delta} \tag{A.14c}$$

where the function $\mathcal{I}(\tilde{\mathbf{x}}_{rel})$ converts a relative coordinate into the relative grid cell index of the surrounding grid points (which can then be used with integer arithmetics) and the relative coordinate within the respective grid cell, which is needed to compute the bilinear interpolation of the velocity field onto the departure point. Note that $\mathcal{I}(\tilde{\mathbf{x}}_{rel})$ essentially separates a computation with reals into two parts. One that can be computed with integers without rounding errors, and a calculation with reals, with a removed offset to avoid rounding errors for floats or posits. For further technical details, the reader is referred to the repository www.github.com/milankl/juls. This non-dimensional relative coordinate formulation of the semi-Lagrangian advection scheme is implemented to solve the tracer advection equation in the shallow water model and is shown to work efficiently with 16bit arithmetics.

A.3 Perspectives for quires

Quires are an additional register on hardware to store intermediate results without rounding. A fused operation like multiply-add can therefore be executed with the single rounding error that applies when storing the final result. Let $\mathcal{R}(x)$ be a rounding function that rounds a result of an operation to the nearest representable number in a given number system. A dot product like $r = ca + db$ is approximated with finite precision arithmetic on a conventional CPU as

$$r = \mathcal{R}(\mathcal{R}(ca) + \mathcal{R}(db)), \tag{A.15}$$

that means after each arithmetic operation the rounding function is applied and introduces a rounding error. In contrast, quires would remove the rounding of intermediate results. The quire register is large enough to store the result of an arithmetic operation exactly and no rounding is therefore applied. Storing the final result r means that it has to be representable in the given number system, which requires rounding

$$r = \mathcal{R}(ca + db). \tag{A.16}$$

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. 4.1a involves computations like

$$u^{n+1} = u^n + RK_u \left(Qhv + \partial_x p + D_x + F_x\right) \tag{A.17}$$

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 reduce the rounding error. Quires may allow to perform the sum over different terms to calculate the right-hand side of the equations with a single rounding error when the final result of the new velocity value is stored.

Acknowledgements

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

I gratefully acknowledge funding from the European Research Council under grant number 741112 *An Information Theoretic Approach to Improving the Reliability of Weather and Climate Simulations* and from the UK National Environmental Researc Council (NERC) under grant number NE/L002612/1.

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

References

Arakawa, A. & Hsu, Y.J.G. (1990). Energy Conserving and Potential-Enstrophy Dissipating Schemes for the Shallow Water Equations. 13, 17

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. 13

Arbic, B.K. & Scott, R.B. (2008). On Quadratic Bottom Drag, Geostrophic Turbulence, and Oceanic Mesoscale Eddies. *Journal of Physical Oceanography*, **38**, 84–103. 34

Bezanson, J., Edelman, A., Karpinski, S. & Shah, V.B. (2014). Julia: A Fresh Approach to Numerical Computing. **59**, 65–98. 6

Butcher, J.C. (2008). *Numerical Methods for Ordinary Differential Equations*. Wiley, 2nd edn. 9, 13

Chaurasiya, R., Gustafson, J., Shrestha, R., Neudorfer, J., Nambiar, S. & Niyogi, K. (2018). Parameterized Posit Arithmetic Hardware Generator. 9. 2, 28

Chen, J., Al-Ars, Z. & Hofstee, H.P. (2018). A Matrix-Multiply Unit for Posits in Reconfigurable Logic Leveraging (Open) CAPI. 1–5. 2, 4, 28, 29

Dawson, A. & Düben, P.D. (2017). Rpe v5: An emulator for reduced floating-point precision in large numerical simulations. *Geoscientific Model Development*, **10**, 2221–2230. 2

Diamantakis, M. (2013). The semi-Lagrangian technique in atmospheric modelling: current status and future challenges. In *ECMWF Seminar in numerical methods for atmosphere and ocean modelling*, September, 183–200. 14, 36

Düben, P.D. (2018). A new number format for ensemble simulations. *Journal of Advances in Modeling Earth Systems*. 2

Düben, P.D. & Palmer, T.N. (2014). Benchmark Tests for Numerical Weather Forecasts on Inexact Hardware. *Monthly Weather Review*, **142**, 3809–3829. 2

Gill, A.E. (1982). *Atmosphere-Ocean Dynamics*. Academic Press. 13

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. 2, 28

Grassberger, P. & Procaccia, I. (1983). Measuring the strangeness of strange attractors. *Physica D: Nonlinear Phenomena*, **9**, 189–208. 12

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**, 2935–2946. 14, 34

Gustafson, J.L. (2017). Posit Arithmetic. 3, 4, 5, 6, 7

Gustafson, J.L. & Yonemoto, I. (2017). Beating Floating Point at its Own Game: Posit Arithmetic. *Supercomputing Frontiers and Innovations*, **4**, 71–86. 2, 3, 4, 7

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*. 2

IEEE (2008). IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2008*, 1–70. 1, 3

Jeffress, S., Düben, P. & Palmer, T. (2017). Bitwise efficiency in chaotic models. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Science*, **473**, 20170144. 1, 9, 33

Klöwer, M., Düben, P.D. & Palmer, T.N. (2019). Posits as an alternative to floats for weather and climate models. In *Conference for Next Generation Arithmetic 2019 (CoNGA'19)*, 8. 2

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**. 9

Lorenz, E.N. (1963). Deterministic Nonperiodic Flow. *Journal of the Atmospheric Sciences*, **20**, 130–141. 9

Madec, G. (2016). NEMO ocean engine. *Note du Pole de modelisation de l'Institut Pierre-Simon Laplace*, **27**. 35

McGuinness, M.J. (1983). The fractal dimension of the Lorenz attractor. *Physics Letters A*, **99**, 5–9. 12

Müller, A., Deconinck, W., Kühnlein, C., Mengaldo, G., Lange, M., Wedi, N., Bauer, P., Smolarkiewicz, P.K., Diamantakis, M., Lock, S.J., Hamrud, M., Saarinen, S., Mozdzynski, G., Thiemert, D., Glinton, M., Bénard, P., Voitus, F., Colavolpe, C., Marguinaud, P., Zheng, Y., Van, J., Degrauwe, D., Smet, G., Termonia, P., Nielsen, K.P., Sass, B.H., Poulsen, W., Berg, P., Osuna, C., Fuhrer, O., Clement, V., Baldauf, M., Gillard, M., Szmelter, J., Brien, E.O., Mckinstry, A., Robinson, O., Kurowski, K., Procyk, M. & Spychala, P. (2019). The ESCAPE project : Energy-efficient Scalable Algorithms for Weather Prediction at Exascale. 31

Palmer, T. (2015). Build imprecise supercomputers. *Nature*, **526**, 2–3. 1

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**, 841–861. 1

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

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. 2, 7

Sadourny, R. (1975). The Dynamics of Finite-Difference Models of the Shallow-Water Equations. 17

Shchepetkin, A.F. & McWilliams, J.C. (2005). The regional oceanic modeling system (ROMS): A split-explicit, free-surface, topography-following-coordinate oceanic model. *Ocean Modelling*, **9**, 347–404. 35

Smolarkiewicz, P.K. & Pudykiewicz, J.A. (1992). A Class of Semi-Lagrangian Approximations for Fluids. *Journal of the atmospheric sciences*, **49**, 2082–2096. 14, 36

Tantet, A., Lucarini, V. & Dijkstra, H.A. (2018). Resonances in a Chaotic Attractor Crisis of the Lorenz Flow. *Journal of Statistical Physics*, **170**, 584–616. 9

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**, 897–908. 2

Vallis, G.K. (2006). *Atmospheric and Ocean Fluid Dynamics*. Cambridge University Press. 13, 34

van Dam, L. (2018). *Enabling High Performance Posit Arithmetic Applications Using Hardware Acceleration*. Ph.D. thesis, Delft University of Technology. 2, 28, 29

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**, 495–502. 1