Klöwer et al. 2021. *Fluid simulations accelerated with 16 bit: Approaching 4x speedup on A64FX*, submitted.

Fluid simulations accelerated with 16 bit: Approaching 4x speedup on A64FX by squeezing ShallowWaters.jl into Float16

Milan Klöwer[1,*], Sam Hatfield[2], Matteo Croci[3], Peter D. Düben[2] and Tim N. Palmer[1]

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

[*]Corresponding author: milan.kloewer@physics.ox.ac.uk

Most Earth-system simulations run on conventional CPUs in 64-bit double precision floating-point numbers Float64, although the need for high-precision calculations in the presence of large uncertainties has been questioned. Fugaku, currently the world's fastest supercomputer, is based on A64FX microprocessors, which also support the 16-bit low-precision format Float16. We investigate the Float16 performance on A64FX with ShallowWaters.jl, the first fluid circulation model that runs entirely with 16-bit arithmetic. The model implements techniques that address precision and dynamic range issues in 16 bit. The precision-critical time integration is augmented to include compensated summation to minimize rounding errors. Such a compensated time integration is as precise but faster than mixed-precision with 16 and 32-bit floats. As subnormals are inefficiently supported on A64FX the very limited range available in Float16 is $6 \cdot 10^{-5}$ to 65504. We develop the analysis-number format Sherlogs.jl to log the arithmetic results during the simulation. The equations in ShallowWaters.jl are then systematically rescaled to fit into Float16, using 97% of the available representable numbers. Consequently, we benchmark speedups of 3.8x on A64FX with Float16. Adding a compensated time integration the speedup is 3.6x. Although ShallowWaters.jl is simplified compared to large Earth-system models, it shares essential algorithms and therefore shows that 16-bit calculations are indeed a competitive way to accelerate Earth-system simulations on available hardware.

Plain Language Summary

Computational performance is a major limitation to improved weather and climate forecasts. Most Earth-system simulations run on conventional computers with every calculation being performed with 64 bit at very high precision, although the need for high-precision calculations in the presence of large uncertainties of the climate system has

been questioned. We present results with ShallowWaters.jl, the first fluid circulation model that runs entirely with 16-bit precision, essentially making every calculation only to 4 digits accurate. Furthermore, only numbers between $6 \cdot 10^{-5}$ to 65,504 are representable and we systemically rescale all calculations to not exceed this range, making use of 97% of all representable numbers within. Simulations with ShallowWaters.jl performed on modern hardware are almost 4x faster than the conventional high-precision calculations. Although ShallowWaters.jl is simplified compared to large Earth-system models, it shares essential algorithms and therefore shows that 16-bit calculations are indeed a competitive way to accelerate Earth-system simulations on available hardware.

Key points

- The first fluid circulation model entirely based on 16-bit instead of conventional 64-bit calculations approaches 4x speedups on hardware.
- Systematically rescaling squeezes all calculations into the very limited range of Float16, making use of 97% of the available numbers.
- Compensated summation in the precision-critical time integration minimizes rounding errors from Float16 and is faster than mixed-precision.

1. Introduction

The first numerical weather prediction models have recently moved away from 64-bit double precision floating-point numbers for higher computational efficiency in lower precision (Govett et al., 2017; Nakano et al., 2018; Rüdisühli et al., 2013; Váňa et al., 2017). While both Float32 and Float64 formats are widely available for high-performance computing, support for 16-bit arithmetic is only available on mainstream hardware for a few years, due to the demand for low precision by deep learning. The transition for an existing application towards 16 bit is challenging: Rounding errors from low precision have to be controlled and a limited range of representable numbers cannot be exceeded without causing often catastrophic under and overflows. But the potential performance gains are promising, with 4x speedups compared to 64-bit calculations, not to mention the reduced energy consumption.

The current boom in machine learning applications is supported by advances in microprocessors. Instead of conventional central processing units (CPU), graphic and tensor processing units GPU, TPU (N. Jouppi et al., 2018; N. P. Jouppi et al., 2017; Steinkraus et al., 2005) are used, which are better suited for the workloads of machine learning. While

most supercomputers are based on Intel CPUs with the x86-64 architecture (Dongarra & Luszczek, 2011), many new installations transition towards GPUs or alternative microprocessor architectures (Zheng, 2020). The trend is towards heterogeneous computing with specialised hardware, which is both a challenge and an opportunity for weather and climate models (Bauer et al., 2021). Fugaku, the world's fastest supercomputer as of 2020, is based on Fujitsu's A64FX processors with ARM architecture (Odajima et al., 2020; Sato et al., 2020). The A64FX also implements the Float16 format (1 sign, 5 exponent and 10 mantissa bits) and Fujitsu promises a four-fold increase in the number of floating-point operations per second.

Float16 is the 16-bit variant of Float32 and Float64 and is defined in the 2008 revision of the IEEE-754 standard on floating-point arithmetic ('IEEE Standard for Binary Floating-Point Arithmetic', 1985; 'IEEE Standard for Floating-Point Arithmetic', 2008). Alternatives such as bfloat16 (Burgess et al., 2019; Kalamkar et al., 2019), minifloats (Fox et al., 2020), logarithmic fixed-point numbers (Johnson, 2020; Sun et al., 2020), posits (Gustafson & Yonemoto, 2017; Klöwer et al., 2019; Langroudi et al., 2019; Zhang & Ko, 2020) and stochastic rounding (Croci & Giles, 2020; Hopkins et al., 2020; Mikaitis, 2020; Paxton et al., 2021) have been investigated, most of these are not available on standard supercomputing hardware. Currently only floats (and integers) enjoy a widely available support in terms of hardware, libraries and compilers that effectively make it possible to execute complex computational applications.

The use of low-precision number formats is motivated as in the presence of large uncertainties in the climate system rounding errors are masked by other sources of error (Palmer, 2015). Typical rounding errors from high-precision calculations are many orders of magnitude smaller than errors in the observations, from coarse resolution or underrepresented physical processes. Low-precision calculations are therefore, at least in theory, sufficient without a loss in accuracy for a weather forecast or a climate prediction. Emulated in parts of weather and climate models, 16-bit half precision has been shown to be a potential route to accelerated simulations (Chantry et al., 2019; Hatfield et al., 2019; Klöwer et al., 2020).

Although weather and climate model data often comes with large uncertainties, many intermediate calculations inside a model simulation require a higher precision. Time integration is often a precision-critical part of numerical simulations of dynamical systems. Stability constraints require small time steps such that tendencies are often several times smaller than the prognostic variables (Courant et al., 1967). Adding the two yields a loss of

precision from the tendency as small increments can only be poorly resolved in low precision (S. Gill, 1951; Kahan, 1965; Møller, 1965). In extreme cases this can lead to a model stagnation (Croci & Giles, 2020), and is often dealt with using mixed-precision approaches (Dawson et al., 2018; Klöwer et al., 2020; Tintó Prims et al., 2019), where the tendencies are computed in low-precision, but converted to a high-precision format before addition. This is beneficial as a large share of computing time is accelerated with low precision, while precision-critical operations are kept in high precision.

Precision loss in calculations can be analysed with a variety of available tools, like FPBench (Damouche et al., 2017), CADNA (Jézéquel & Chesneaux, 2008), Verrou (Fevotte & Lathuilière, 2019), and Verificarlo (Denis et al., 2016). Such tools are often either based on interval arithmetic, providing rigid rounding error bounds, or on stochastic arithmetic to assess the rounding error growth. While these can be useful to identify the minimal decimal precision for simulating chaotic systems, analysing the limited dynamic range of low precision number formats is largely unaddressed in these tools.

Here, we present, to our knowledge, the first fluid circulation model that runs entirely in hardware-accelerated 16-bit floats on the ARM architecture-based microprocessor A64FX. Strategies are presented to solve precision and range issues with 16-bit arithmetic: In section 2 we scale the shallow water equations, and an appropriate scale is found with the newly-developed analysis-number format Sherlogs.jl, which is introduced in section 3. A compensated time integration is presented in section 4 to minimize precision issues. Section 5 analyses the rounding errors of Float16 in ShallowWaters.jl and benchmarks the performance compared to Float64. Section 6 discusses the results.

2. Scaling the shallow water equations

The shallow water equations describe atmospheric or oceanic flow idealised to two horizontal dimensions. They result from a vertical integration of the Navier-Stokes equations (A. E. Gill, 1982; Vallis, 2006) and are simplified but representative of many weather and climate models, which are usually solved with many coupled horizontal layers. They describe the time evolution of the prognostic variables velocity $\mathbf{u} = (u, v)$, and interface height η in the following form

$$\begin{aligned}
&\partial_t \mathbf{u} + \mathbf{u} \cdot \nabla \mathbf{u} + f\mathbf{z} \times \mathbf{u} = -g\nabla\eta + \nu_B \nabla^4 \mathbf{u} - r\mathbf{u} + \mathbf{F} \\
&\partial_t \eta + \nabla \cdot (\mathbf{u}h) = 0 \\
&\partial_t q + \mathbf{u} \cdot \nabla q = -\tau(q - q_0)
\end{aligned} \tag{1}$$

defined over a rectangular domain with zonal and meridional coordinates x, y of size L_x = 8000km, L_y = 4000km, respectively. The domain is a zonal channel with boundary conditions being periodic in x. The channel setup is motivated by zonal flows like the Antarctic Circumpolar Current but highly idealised (Jansen et al., 2015).

The non-linear momentum advection is $\mathbf{u} \cdot \nabla \mathbf{u}$. The Coriolis force is $f\mathbf{z} \times \mathbf{u} = (-fv, fu)$ with the Coriolis parameter f using a β-plane approximation at 45˚N. The pressure gradient $-g\nabla\eta$ scales with a reduced gravitational acceleration $g = 0.01\ \mathrm{ms}^{-2}$ to represent baroclinic ocean/atmosphere dynamics (A. E. Gill, 1982). The zonal wind forcing $\mathbf{F} = (F_x, 0)$ is a meridional shear $F_x = F_0 \sin(\omega t) \tanh(2\pi(yL_y^{-1} - \frac{1}{2}))$ which reverses seasonally ($\omega^{-1} = 365\ \mathrm{days}$). Lateral diffusion of momentum is described by $\nu_B \nabla^4 \mathbf{u}$, with biharmonic viscosity coefficient ν_B. Linear bottom friction is represented by $-r\mathbf{u}$ which decelerates the flow at a time scale of $r^{-1} = 300\ \mathrm{days}$. The equation for interface height η is the shallow water-variant of the continuity equation, ensuring conservation of volume. The layer thickness is $h = \eta + H$ of a fluid with depth H at rest. Several meridional ridges on the seafloor trigger instabilities in the zonal flow, but they are small compared to the fluid depth. The shallow water equations are complemented with an advection for the passive tracer q, which is stirred by the flow and slowly ($\tau^{-1} = 100\ \mathrm{days}$) relaxed back to a reference q_0. For further details and parameter choices see the Appendix.

In order to control the range of numbers occurring in the simulation, the shallow water equations are scaled with a multiplicative constant. The evaluation of linear terms is not affected, but the non-linear terms involve an unscaling. The same constant s is chosen for zonal velocity u and meridional velocity v, such that $\hat{u} = su$ and $\hat{v} = sv$. Additionally, we use dimensionless spatial gradients $\hat{\partial}_x = \Delta x \partial_x, \hat{\nabla} = \Delta x \nabla$, etc. by scaling the equations with the grid spacing Δx. For simplicity, we use the same Δx in x and y-direction but generalisation to less regular grids is possible. The grid spacing Δx is then combined with the time step $\widehat{\Delta t} = \frac{\Delta t}{\Delta x}$ and $\hat{\partial}_t = \Delta x \partial_t$. Due to the 4th-order gradient in the viscosity, we scale its coefficient as $\hat{\nu_B} = \Delta x^{-3} \nu_B$. Using the potential vorticity $h^{-1}(f + \zeta)$, with the relative vorticity $\zeta = \partial_x v - \partial_y u$, and the Bernoulli potential $\frac{1}{2}(u^2 + v^2) + g\eta$, the shallow water equations can be written into a scaled form as

$$\begin{aligned}
\hat{\partial}_t\hat{u} &= \frac{[s\Delta x f]+\hat{\zeta}}{\hat{h}}\frac{\hat{v}\hat{h}}{s} - \hat{\partial}_x\left([\frac{1}{2s}](\hat{u}^2+\hat{v}^2)+[\frac{sg}{s_\eta}]\hat{\eta}\right) + \hat{\nu_B}\hat{\nabla}^4\hat{u} - [r\Delta x]\hat{u} + [s\Delta x F_x] \\
\hat{\partial}_t\hat{v} &= -\frac{[s\Delta x f]+\hat{\zeta}}{\hat{h}}\frac{\hat{u}\hat{h}}{s} - \hat{\partial}_y\left([\frac{1}{2s}](\hat{u}^2+\hat{v}^2)+[\frac{sg}{s_\eta}]\hat{\eta}\right) + \hat{\nu_B}\hat{\nabla}^4\hat{v} - [r\Delta x]\hat{v} + [s\Delta x F_y]
\end{aligned} \tag{2}$$

Square brackets denote pre-computed constants and only the volume fluxes uh, vh have to be unscaled on every time step. As the volume fluxes are quadratic terms, the evaluation of $\hat{u}\hat{h}$ scales as s^2, which therefore has to be partly unscaled with s^{-1}. The continuity equation is rescaled with s_η, i.e. $\hat{\eta} = s_\eta\eta$ as well as $\hat{h} = \hat{\eta} + s_\eta H$, and the tracer advection equation is rescaled with s_q, so that $\hat{q} = s_q q$

$$\begin{aligned}
\hat{\partial}_t\hat{\eta} &= -\hat{\partial}_x(\frac{\hat{u}\hat{h}}{s}) - \hat{\partial}_y(\frac{\hat{v}\hat{h}}{s}) \\
[s\hat{\partial}_t]\hat{q} &= \left(-\hat{u}\hat{\partial}_x\hat{q} - \hat{v}\hat{\partial}_y\hat{q}\right) - [\tau\Delta x](\hat{q}-\hat{q}_0)
\end{aligned} \tag{3}$$

ShallowWaters.jl solves these scaled shallow water equations with 2nd order finite differencing on a regular, but staggered Arakawa C-grid (Arakawa & Lamb, 1977). The advection of potential vorticity uses the energy and enstrophy-conserving scheme of Arakawa and Hsu (Arakawa & Hsu, 1990). The tracer advection equation for q is solved with a semi-Lagrangian advection scheme (Diamantakis, 2013; Smolarkiewicz & Pudykiewicz, 1992). This scheme calculates a departure point for every arrival grid point one time step ago. The tracer field is then interpolated onto the departure point, which is used as the tracer concentration at the arrival point for the next time step. More details on the implementation of the semi-Lagrangian advection scheme is described in Klöwer et al. 2020. The time integration of ShallowWaters.jl is discussed in section 4.

3. Choosing a scale with Sherlogs.jl

The scaling of equations has to be implemented carefully when using number formats with a limited dynamic range, such as Float16 (Fig. 1). Subnormals for Float16 are in the range of $6{\cdot}10^{-8}$ to $6{\cdot}10^{-5}$ (see the appendix). Subnormals are inefficiently supported on some hardware, such that their occurrence causes large performance penalties. This reduces the available range of Float16 even further and a simulation has to fit as best as possible in the remaining 9 orders of magnitude between $6.104 \cdot 10^{-5}$ and 65,504. A single overflow, i.e. a result above 65,504, will abort the simulation. Understanding the range of numbers that

occur in all operations and ideally in which lines of the code is therefore very important. For most algorithms this is very difficult to achieve unless the numbers are directly measured within the simulation.

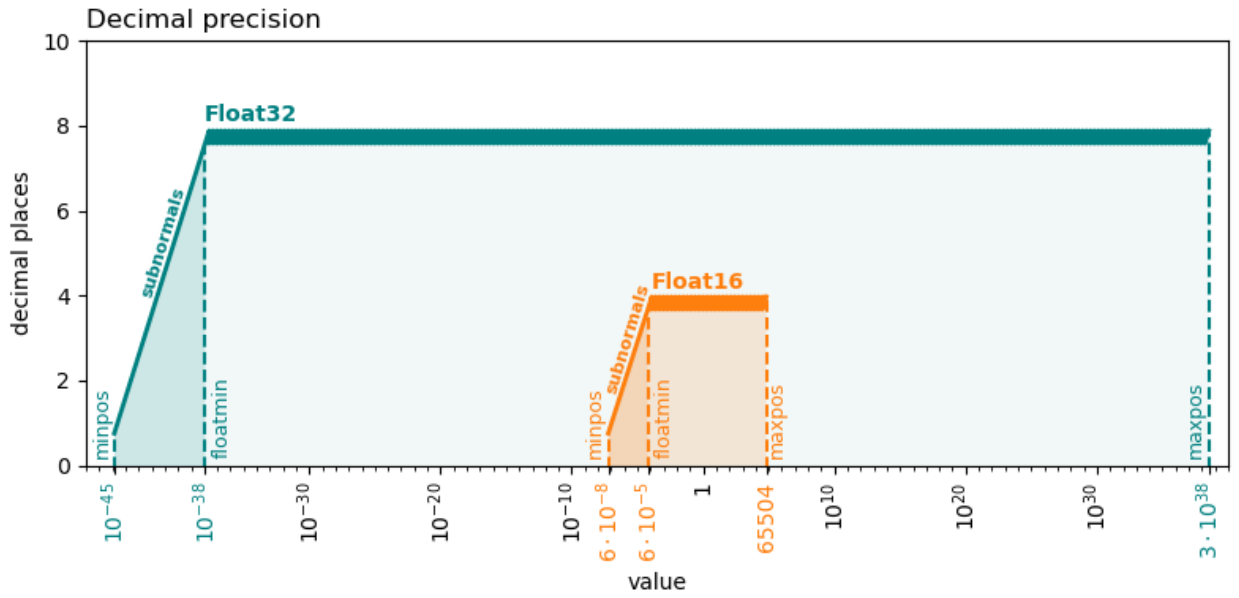

Figure 1 | Decimal precision of Float16 and Float32 over the range of representable numbers. The decimal precision is worst-case, i.e. given in terms of decimal places that are at least correct after rounding (see Appendix). The smallest representable number (minpos), the smallest normal number (floatmin) and the largest representable number (maxpos) are denoted with vertical dashed lines. The subnormal range is between minpos and floatmin respectively.

We therefore developed the analysis-number format Sherlogs. Sherlog16, for example, uses Float16 to compute, but after every arithmetic operation the result is also logged into a bitpattern histogram. Running a simulation with Sherlogs will take considerably longer due to the overhead from logging the arithmetic results, which can be obtained in the form of a bitpattern histogram upon completion. The bitpattern histogram will reveal information such as the smallest and largest occurring numbers or how well an algorithm fits into a smaller dynamic range. An example usage of Sherlogs is given in Fig. 2.

```
julia> using ShallowWaters, Sherlogs          # load packages
julia> # run ShallowWaters with Sherlog16 which logs all arithmetic results
julia> run_model(Sherlog16)                   # use Sherlog16 as number format

julia> get_logbook(1)                         # retrieve the bitpattern histogram
65536-element LogBook(1112720887, 1484631, 1378491, 1024411, … , 0, 0, 0, 0, 0)

julia> # run ShallowWaters with DrWatson16 recording a stack trace when f=true
julia> f(x) = 0 < abs(x) < floatmin(Float16)  # true for subnormals
julia> run_model(DrWatson16{f})               # use DrWatson16 as number format

julia> get_stacktrace(1)                      # retrieve the first stack trace
3-element Vector{Base.StackTraces.StackFrame}:
* at DrWatson16.jl:52 [inlined]               # subnormal occurred in *
caxb!(...) at time_integration.jl:320         # inside this function
time_integration(...) at time_integration.jl:82 # called from here
```

Figure 2 | Example usage and output of Sherlogs.jl, a package for Sherlogs and DrWatson, two analysis-number formats that can be combined with type-flexible functions in Julia. Using `Sherlog16` as the first argument of `run_model` runs ShallowWaters.jl with Float16 but also logs the bitpattern of every arithmetic result into a *logbook* of length $2^{16} = 65536$ to create a bitpattern histogram. `DrWatson16{f}` uses Float16 but also records a stack trace (a list of calling functions and respective lines of code) every time the function `f(x)` evaluates to true with the arithmetic result `x`. Here, a subnormal arises in a multiplication (`*` in line 14 here) in line 320 of the code in script time_integration.jl.

Sherlogs are implemented in the package Sherlogs.jl, which makes use of the type-flexible programming paradigm in Julia (Bezanson et al., 2017). A function is written in an abstract form, which is then dynamically dispatched to the number format provided and compiled just-in-time. Such a number format can therefore be, for example, Float64 or Float16, but also any user-defined number format such as Sherlogs.

An appropriate scaling $^{S}, ^{S_\eta}, ^{S_q}$ has to be chosen for a given set of parameters. The bitpattern histogram of the entirely unscaled shallow water equations simulated with Float32 reveals range issues that would arise with Float16 (Fig. 3a). A large share (10%) of the arithmetic results would be below the representable range of Float16. Consequently, running the model without any scaling modifications in Float16 would round many numbers to 0, causing so-called underflows that deteriorate the simulated dynamics (Klöwer et al., 2020). Most of these underflows occur in the calculation of gradients, which consequently have to be non-dimensionalised as previously suggested (Klöwer et al., 2019). This also largely removes a resolution-dependence of the bitpattern histograms, such that

Float16 simulations are possible across a wide range of resolutions. Dimensionless gradients are a major improvement to fit ShallowWaters.jl into the available range with Float16, yet 3% of the arithmetic results are subnormals (Fig. 3b). On A64FX a flag can be set to avoid the performance penalty from subnormals by flushing every occurring subnormal to zero. The smallest representable number is therefore $6.104 \cdot 10^{-5}$.

Using the DrWatson number format from Sherlogs.jl identifies the addition of the tendencies to the prognostic variables u, v, η as prone to produce subnormals (Fig. 2). We therefore increase the scales s, s_η to scale up the prognostic variables and consequently their tendencies. Choosing $s = 2^6, s_\eta = 1$ reduces the amount of subnormals to 0.04%, while leaving about a factor two headspace between the largest occurring numbers (about 30,000) to avoid overflows beyond 65,504 (Fig. 3b). The compensated time integration (section 4) increases this share to about 0.2%.

The idealised tracer in ShallowWaters.jl takes values in (-1,1), so we scale this variable by $s_q = 2^{15}$ in order to use most of the Float16 range. This is to allow as many bitpatterns as possible for the interpolation in the semi-Lagrangian advection scheme, which uses non-dimensional departure points on a locally relative grid for 16-bit arithmetic, as described in Klöwer et al., 2020.

Consequently, the fully scaled shallow water equations are squeezed well into Float16, making near-optimal use of the available bitpatterns, of which only 3% are unused (NaNs excluded). In contrast, a simulation with Float32 does not make use of at least 81% of available bitpatterns (Fig. 3), assuming that for a simulation run long enough all bitpatterns within the used range occur eventually. Extrapolating this to Float64 with a representable range of $5 \cdot 10^{-324}$ to $2 \cdot 10^{308}$ the share of unused bitpatterns is at least 97.5%. This computational inefficiency can be overcome with 16-bit number formats and systematic scaling as presented here. However, scaling leaves the precision issues with low-precision formats unaddressed, for which we present a technique in the next section.

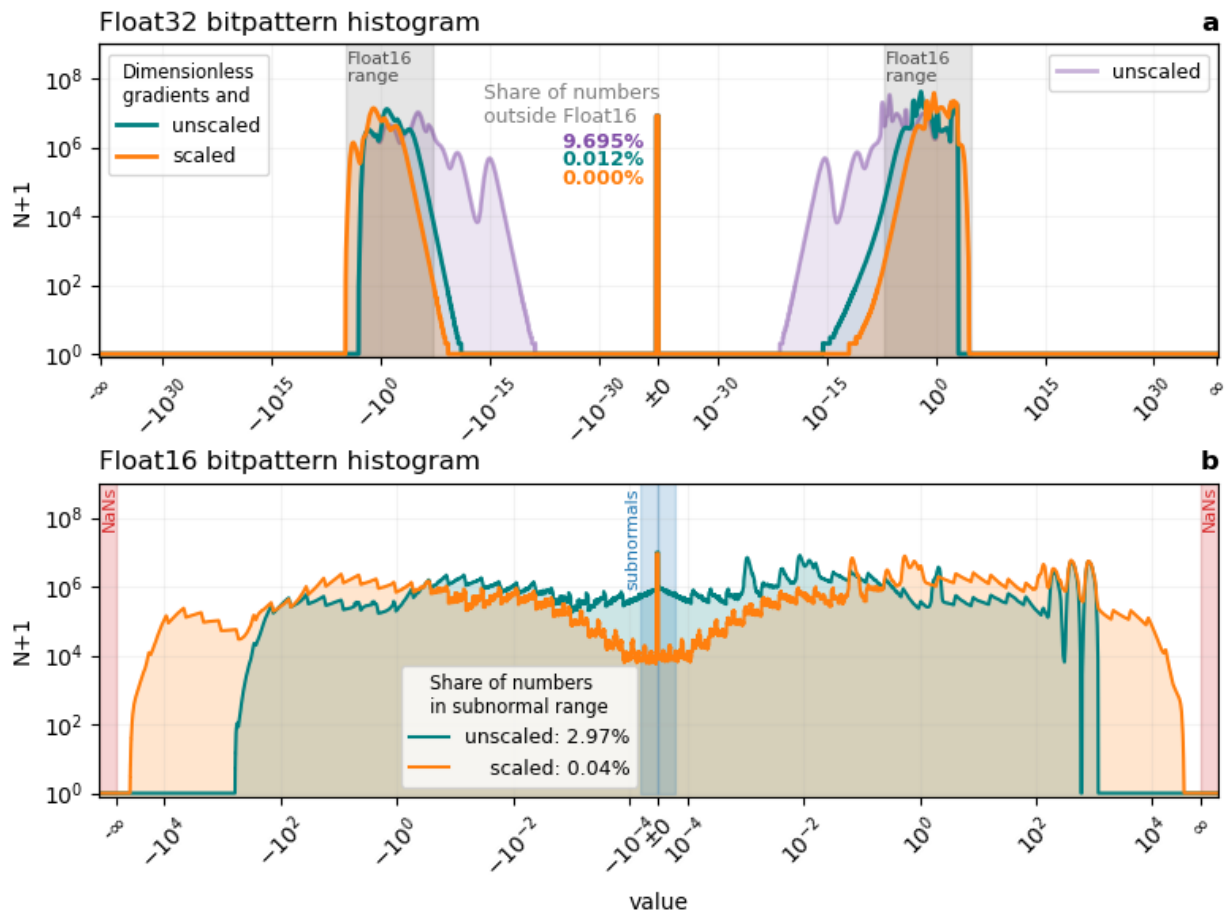

Figure 3 | Bitpattern histogram of all arithmetic results in ShallowWaters.jl. a 200-day simulation at Δx = 20km based on Float32 arithmetic. The share of numbers outside the Float16 range (grey shading) are colour-coded to the respective histograms. **b** as **a** but based on Float16. Bitpattern histograms are created with Sherlogs.jl. The logarithmic y-axis denotes the number of occurrences N of the respective bitpattern during the simulation. The histograms span all available bitpatterns in the respective formats evenly but are sorted and relabelled with the corresponding values for readability. The range of bitpattern that are subnormals or interpreted as Not-A-Number (NaN) are marked. Bitpatterns histograms are without compensated time integration (section 4).

4. A compensated time integration

To minimize the precision loss in the time-integration, we adopt compensated summation as an alternative approach to mixing precision. Compensated summation is a simple, yet powerful technique that prevents the accumulation of rounding errors in the computation of large sums. Since the addition of multiple terms is ubiquitous in scientific computing, compensated summation can be used to improve the accuracy of many algorithms such as numerical linear algebra operations, integration or optimization. Here we use compensated

summation to augment the resilience to rounding errors of our half-precision time-stepping method.

The first version of compensated summation was used by Gill in 1951 in a Runge-Kutta integrator scheme in fixed-point arithmetic (S. Gill, 1951), and the idea was subsequently extended to floating-point arithmetic by Kahan (Kahan, 1965), Møller (Møller, 1965) and others (Higham, 1993; Linnainmaa, 1974; Vitasek, 1969). That we are aware of, our paper is the first work in which compensated summation is used in a fluid circulation model with 16-bit arithmetic.

To understand compensated summation, consider the following naïve algorithm for the summation of all the entries of a length-n vector $a_i, i = 1, ..., n$

```
sum = 0                  # variable to store the sum
for aᵢ in a              # loop over all elements of a
    sum = sum + aᵢ       # accumulate each element into sum
end
return sum
```

This algorithm is prone to rounding errors, which accumulate at a rate proportional to n (Higham, 1993). Furthermore, the algorithm might cause stagnation, a phenomenon for which the partial sum becomes too large, causing each subsequent addition to be neglected due to rounding. Compensated summation offers a much better alternative at the cost of introducing an additional compensation variable `c`:

```
c = 0                    # compensation, initially 0
sum = 0                  # variable to store the sum
for aᵢ in a              # loop over all elements of a
    aᵢc = aᵢ - c         # compensate rounding error from previous iteration
    temp = sum + aᵢc     # add next element of a, but store in temp
    c = (temp-sum) - aᵢc # rounding error from sum+aᵢc
    sum = temp           # copy addition back to sum
end
return sum
```

At infinite precision, the compensation `c` will remain 0. At finite precision, however, calculating `c = (temp-sum) - aᵢc` will estimate the rounding error in the addition `sum + aᵢc` and subsequently attempt to compensate for it in the next iteration through `aᵢc = aᵢ - c`. For base-2 floating point arithmetic we have exactly `sum + aᵢ = temp + c`, i.e. the compensation variable `c` correctly captures the rounding errors in the addition.

Compensated summation prevents the rounding errors from accumulating, and the overall summation error will stay a mere multiple of machine precision (Higham, 1993). Overall, the compensation `c` can be interpreted as a storage variable for rounding errors and effectively prevents rounding errors in the summation from growing beyond machine precision accuracy.

Compensated summation is especially useful in settings in which the order of summation cannot be manipulated to prevent rounding error growth. Time integration schemes, for which the state variables are updated sequentially, are especially amenable to augmentation by compensated summation. Over a time period T the number of terms to be added scales as $T\Delta t^{-1}$, proportional to one for each time step. The naïve algorithm would cause rounding errors to grow like $\mathcal{O}(T\Delta t^{-1}\varepsilon)$, causing errors to counter-intuitively grow as the time-step is refined. With compensated summation the rounding errors will stay $\mathcal{O}(\varepsilon)$.

ShallowWaters.jl uses the 4-th order Runge-Kutta scheme (Butcher, 2008) to integrate the non-dissipative terms in time: The momentum advection $\mathbf{u}\cdot\nabla\mathbf{u}$; the Coriolis force $f\mathbf{z}\times\mathbf{u}$; the pressure gradient $-g\nabla\eta$; the wind forcing $\mathbf{F}$; and the conservation of volume $-\nabla\cdot(\mathbf{u}h)$ are summarized as the right-hand side function rhs. The time integration is now augmented with compensated summation. The rounding error c_u that occurs in the addition of the total tendency du to the previous time step u_n is calculated and stored. On the next time step, this rounding error is subtracted from the total tendency du in an attempt to compensate for the rounding error from the previous time step. This is illustrated here for the zonal velocity u in isolation, although in practice time integration has to update the prognostic variables u, v, η simultaneously. A compensated time integration for u with RK4 can be written as

$$\begin{aligned}
k_1 &= \mathrm{rhs}(u)\\
k_2 &= \mathrm{rhs}(u + \tfrac{\Delta t}{2}k_1)\\
k_3 &= \mathrm{rhs}(u + \tfrac{\Delta t}{2}k_2)\\
k_4 &= \mathrm{rhs}(u + \Delta t k_3)\\
du &= \tfrac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4) - c_u\\
u^*_{n+1} &= u_n + du\\
c_u &= (u^*_{n+1} - u_n) - du
\end{aligned} \tag{4}$$

with $c_u = 0$ as initial condition. The addition $u_n + du$ usually suffers from rounding errors as described above. The loss of precision in du is calculated in c_u (which is only 0 in exact arithmetic). The compensation is analogously implemented with c_v, c_η for the other prognostic variables.

The dissipative terms, i.e. biharmonic diffusion of momentum $\nu_B \nabla^4 \mathbf{u}$ and bottom friction $-r\mathbf{u}$, are integrated with a single forward step after the Runge-Kutta integration in ShallowWaters.jl and summarized as $\mathrm{rhs}_\mathrm{diss}$. To compensate for rounding errors for both the dissipative and non-dissipative terms simultaneously, c_u from Eq. (4) is subtracted from the total dissipative tendency du_diss. In that sense, the rounding error from Eq. (4) is attempted to be compensated subsequently in Eq. (5), and vice versa.

$$\begin{aligned} du_\mathrm{diss} &= \Delta t\ \mathrm{rhs}_\mathrm{diss}(u^*_{n+1}) - c_u \\ u_{n+1} &= u^*_{n+1} + du_\mathrm{diss} \\ c_u &= (u_{n+1} - u^*_{n+1}) - du_\mathrm{diss} \end{aligned} \tag{5}$$

Only the addition of the total tendency is compensated here to minimize the amount of additional calculations, which increases when also compensating the 3 sub steps in RK4.

The compensated time integration is an alternative to mixed-precision approaches. While those aim to keep the precision high in the precision-critical calculations, the compensated time integration introduces a new variable to compensate for the rounding errors in one precision-critical calculation. With compensated time integration all variables can be kept in 16 bit, and no conversions between number formats are necessary.

5. A fluid simulation calculated entirely in 16 bit

The accumulated rounding error from mixing precision and compensated time integration is now assessed. ShallowWaters.jl is started from identical, in Float16 perfectly representable, initial conditions in a domain of 8000km by 4000km. The model is spun-up to reach a turbulent flow domain-wide, while the tracer starts from an idealised checkerboard pattern to better highlight the turbulence everywhere in the domain. The grid consists of 3000x1500 points at about 2.7km grid-spacing (see appendix for the physical parameters). With Float16 and without compensated time integration, the accumulated rounding error for zonal velocity u compared to Float64 exponentially increases 100-fold in the first 150 days (Fig. 4a). With mixed-precision, using Float16 for the tendencies and Float32 for the prognostic variables, this rounding error growth is strongly

reduced. Errors after a few time steps without mixed-precision are reached after about 100 days (~25,000 time steps) of integration. After that the error growth accelerates and chaos removes the information of the initial conditions.

Using a compensated time integration, the rounding error from Float16 is strongly reduced and matches well with the error growth of mixed-precision. From the perspective of rounding errors the two methods are therefore equivalently suited to reduce rounding errors with 16-bit arithmetic. The rounding error growth of the other prognostic variables is similar. The positive effect of compensated time integration is well illustrated in snapshots of tracer mixing where even after 100 days of simulation only a very slight deviation from the Float64 reference is observable (Fig. 4b, c and d).

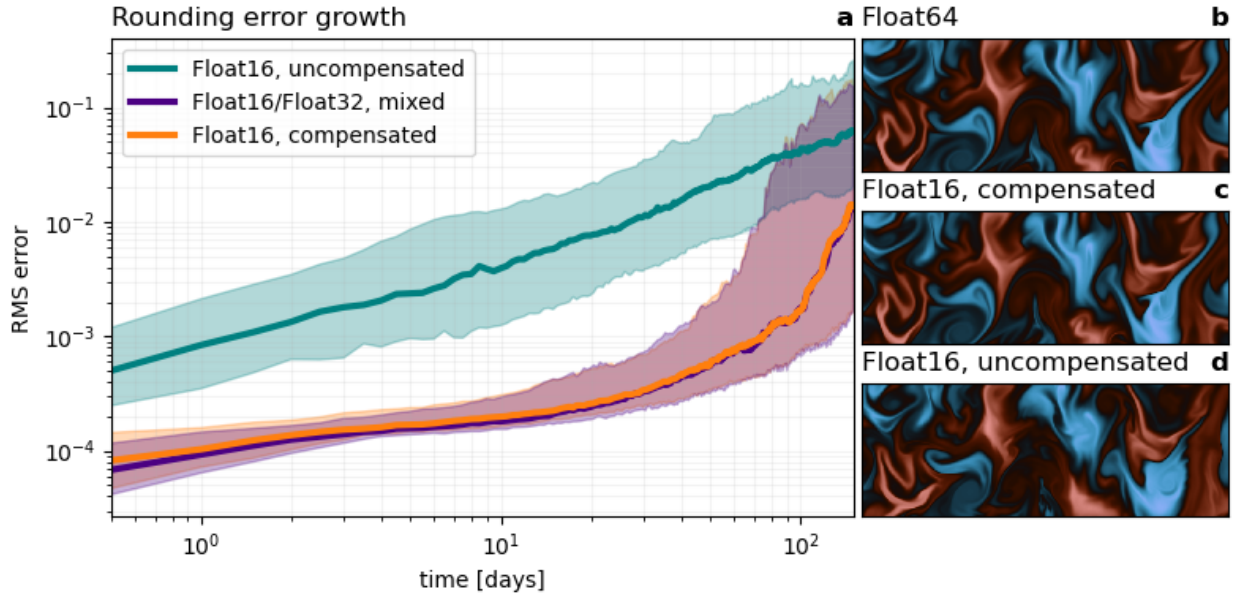

Figure 4 | Rounding error growth with Float16 in ShallowWaters.jl using compensated time integration or mixed-precision. a Errors are root-mean square (RMS) errors of zonal velocity u relative to Float64. Solid lines denote the median and shadings the interdecile confidence interval. **b**,**c** Snapshots of tracer q from a zoom into Fig. 5 after 100 days of simulation and **d** as **c** but without compensated time integration.

Even after 100 days of simulation a large simulation (3000x1500 grid points) with Float16 shows minimal errors in the tracer mixing compared to Float64 (Fig. 5). Only at regions near the boundaries, where the mixing is enhanced, a difference is visible. The remaining rounding error is small and will be masked in a more realistic setup by model or discretization errors. To better understand the simulated timescales, an animated version of Fig. 5 is available in the supplement.

Reducing the precision in calculations raises concerns about the numerical conservation of physically conserved quantities like mass. The compensated time integration conserves the mass in the shallow water equations with Float16 (<0.002% change within 500 days compared to Float64), similar to mixed-precision (Fig. S1). Without compensated time integration for Float16 the conservation is with 0.05% change over 500 days less accurate. Similar results were obtained for the conservation of the tracer. We will now assess the speedups with Float16 compared to Float64 on A64FX.

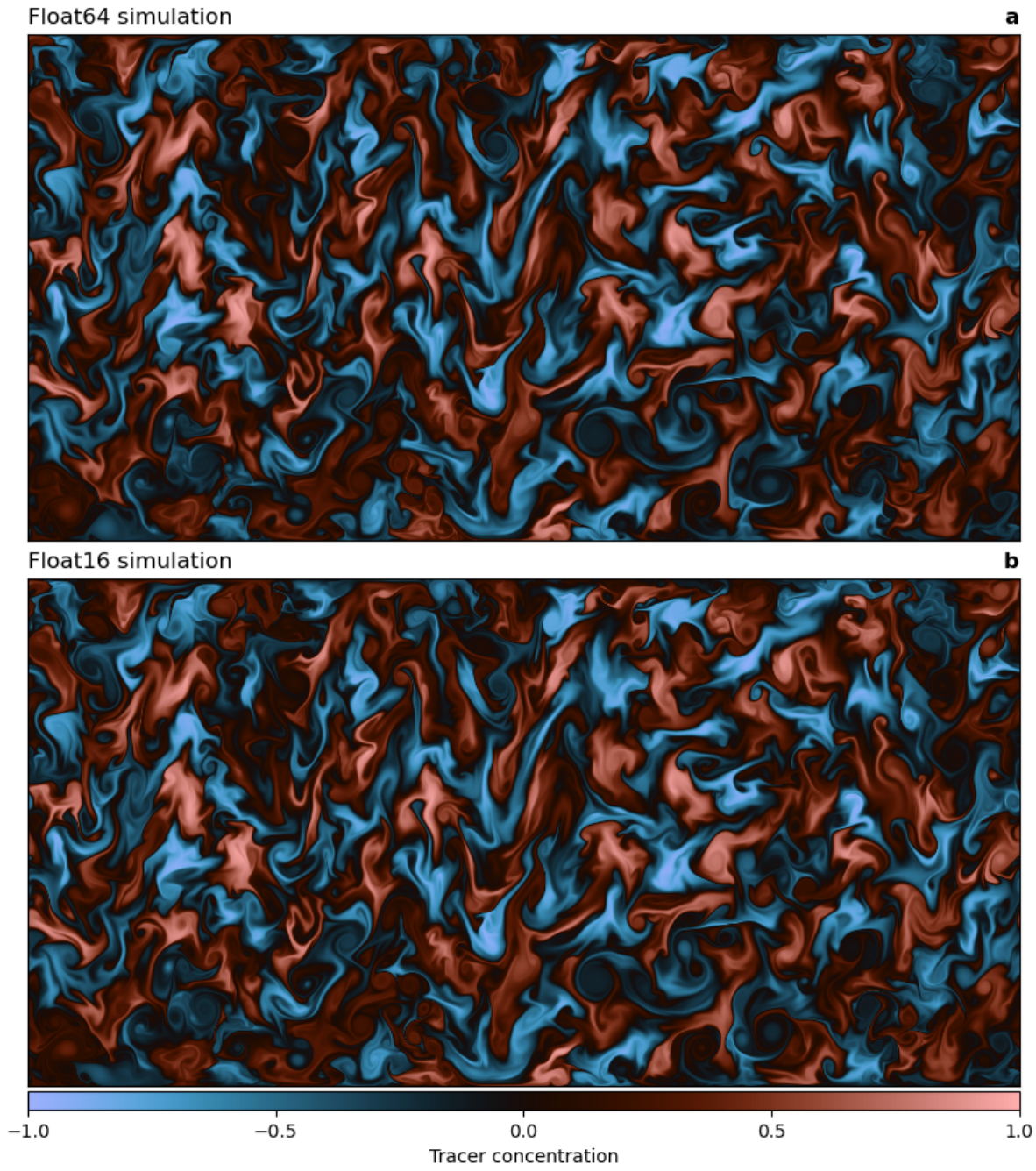

Figure 5 | Turbulent tracer mixing as simulated by ShallowWaters.jl. a Simulation based on Float64 arithmetic and **b** Float16 with compensated time integration. Snapshot is taken after 100 days of simulation (~25,000 time steps) with 3000x1500 grid points starting from identical initial conditions. Remaining errors between **a** and **b** from low precision Float16 are tolerable and will be masked by other sources of error in a less idealised model setup.

The A64FX is a microprocessor developed by Fujitsu based on the ARM-architecture. It powers not just the fastest supercomputer in the world as of June 2021 (measured by TOP500.org, Dongarra & Luszczek, 2011), Fugaku, but also a number of smaller systems around the world, including Isambard 2 which we use here. The A64FX has a number of features intended to accelerate machine learning applications. Notably, it allows not just Float32 and Float64 arithmetic but also Float16. Official benchmarks of the A64FX demonstrate a cost increase which is linear with the number of bits. In that sense, Float32 can be twice as fast in applications than Float64, while Float16 can be four times as fast, when optimized well. In practice, speedups in complex applications are due to a mix of factors: In compute-bound applications, the wall-clock time is largely given by the clock rate of the processor and the vectorization of arithmetic operations (such that small sets of them are performed in parallel on a single processor core). Using Float16 instead of Float64 allows to put four times as many numbers through the vectorization, theoretically allowing for 4x speedups. The performance of memory-bound applications, on the other hand, is largely determined by the data transfer rate between the processor and its various levels of caches that increase in size but decrease in bandwidth. Using Float16 instead of Float64 allows to load four times as many numbers from memory, which theoretically translates to 4x speedup as well.

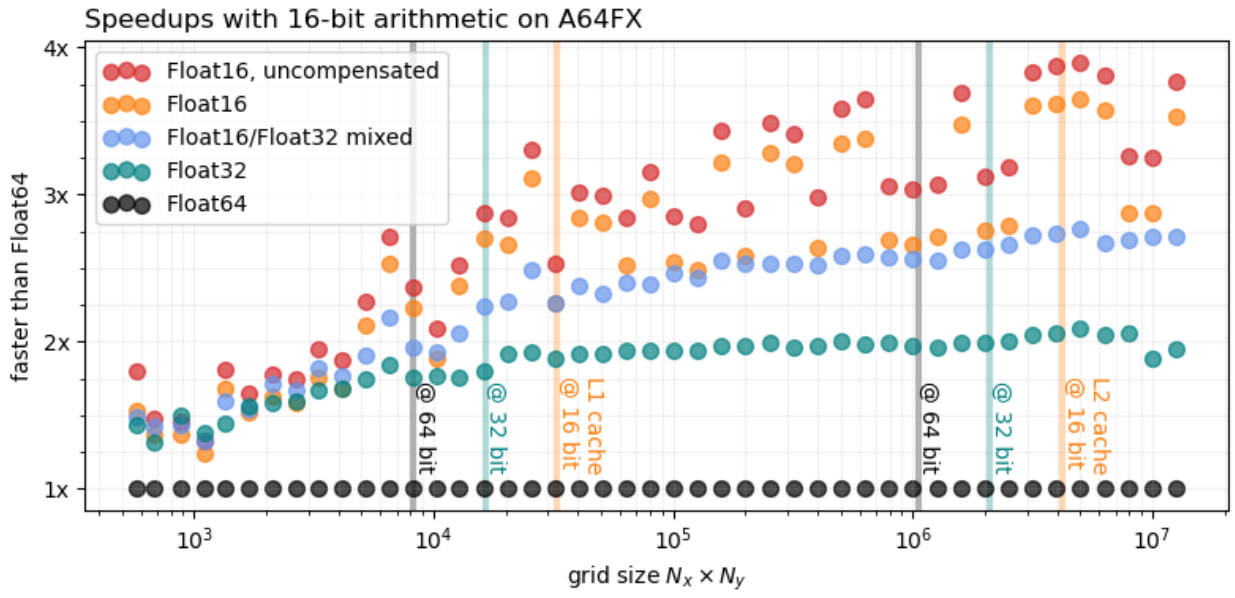

Figure 6 | Performance increase from Float16 when running ShallowWaters.jl at varying grid sizes. The grid size is the total number of grid points $N_x \times N_y$. All timings are single-threaded median wall clock times relative to Float64, excluding compilation, model initialisation and memory pre-allocation. The corresponding size of the L1 and L2 cache (64KiB, 8 MiB) of A64FX is given as vertical lines for arrays of 16, 32 and 64-bit floats.

ShallowWaters.jl is a memory-bound application for which the biggest benefit from Float16 will be the reduction of the size of the arrays by a factor of four when compared to Float64. The arrays can therefore be read faster from memory with a potential speedup of 4x. We benchmark ShallowWaters.jl at varying grid sizes, excluding compilation, model initialisation and memory pre-allocation. With grid sizes of 10^5 (about 450x225 grid points) and larger, there is a clear improvement from using Float32 instead of Float64 which approaches 2x speedups (Fig. 6). Using Float16 these speedups reach up to 3.8x for grid sizes beyond $3{\cdot}10^6$ (about 2450x1225 grid points). The dependency of the speedup on the grid size is complicated: While larger grids usually experience more acceleration on A64FX in Float16, there are ranges where the speedup drops to 3-3.25x. This is likely due to peculiarities in the memory and cache hierarchy of the A64FX, such that the performance benefit of Float16 cannot always be fully realised. A detailed assessment of these peculiarities is beyond the scope of this study, but it is nevertheless reassuring that, even in the worst case, Float16 is still at least three times faster than Float64 for these large grids.

As discussed in previous sections, using a compensated time integration can be used to minimize the rounding errors, which comes with a small additional computational cost: Using the compensated time integration the speedups drop to about 3.6x for large grids. Nevertheless, a compensated time integration yields higher performances than mixing the precision of Float16 and Float32, which approaches only 2.75x here. Consequently, a compensated time integration for Float16 is, although as precise, faster than mixed-precision.

6. Conclusions

Low-precision calculations for weather and climate simulations are a potential that is not yet fully exploited. While the first weather forecast models are moving towards Float32, 16-bit arithmetics will likely find increasing support on future supercomputers. We present, to our knowledge, the first fluid simulation that runs entirely in hardware-accelerated 16 bit with minimal rounding errors but at almost 4x the speed. The simulations were performed on A64FX, the microprocessor that is used in Fugaku, the fastest supercomputer as of November 2020.

The complex partial differential equations underlying weather and climate simulations are difficult to fit into the limited range of Float16, but here we have presented a method to do this more systematically. We present Sherlogs.jl to analyse number formats. Sherlogs.jl allows to assess any changes to the scaling of the equations to minimize underflows while

making the most of the available representable numbers. In our case, subnormal floating-point numbers had to be avoided and scaling of the equations dropped the amount of subnormals occurring below 0.2%.

Using 16-bit floats will likely cause precision issues in fluid simulations. While mixed-precision has been used to minimize rounding errors in precision-critical calculations, we have presented here an approach that compensates for rounding errors to allow for simulations entirely within 16-bit arithmetic. The compensated time integration minimizes rounding errors from this precision-critical part of a simulation at a slightly higher cost. Benchmarking in comparison to mixed-precision shows that the compensated time integration is faster in ShallowWaters.jl while being as precise as mixed-precision.

Alternatives to floats have been discussed for weather and climate simulations previously (Klöwer et al., 2020, 2019). Although posit numbers (Gustafson & Yonemoto, 2017) are more precise in these applications, the improvement from floats to posits is smaller than using mixed-precision and therefore also smaller than the compensated time integration. In that sense, algorithms that are low-precision resilient are far more important than the actual choice of the number format, especially given that only floats are widely hardware-supported.

The work here shows that a naive translation of the mathematical equations into code will likely fail with 16-bit arithmetic. However, this does not mean that 16-bit arithmetic is unsuited for the numerical solution of complex partial differential equations such as the shallow water equations. But it means that both precision and range issues have to be addressed in the design of the algorithms used. A compensated time integration is a low-precision resilient algorithm, and scaling is essential to fit the very limited range of Float16.

While 16-bit hardware is largely designed for machine learning, its potential to increase computational efficiency extends to weather and climate applications too. 16-bit calculations are indeed a competitive way to also accelerate Earth-system simulations on available hardware.

Appendix

Model setup The domain $L_x \times L_y$ of the model setup of ShallowWaters.jl uses periodic boundary conditions in x and no-slip at the Northern and Southern boundary. The standard depth is H_0 = 500m, and several meridional mountain ridges are placed at irregular distances on the seafloor to trigger instabilities in the flow. The layer thickness at rest is

$$H(x) = H_0 - \sum_i H_1 \exp\left(-2H_\sigma^{-2}\left(x - H_{x_i}L_x\right)^2\right) \tag{6}$$

for $i = 1,2,3,4$ with $H_{x_i} = (0.05, 0.25, 0.45, 0.9)$ four relative x-positions between 0 and L_x. The height of the mountain ridges is H_1= 100m and the characteristic width H_σ = 300km. The fluid density is ρ = 1000 kgm^{-3}, which influences the wind forcing coefficient $F_0 = (\rho H_0)^{-1}F_c$ with F_c = 0.12Pa. The biharmonic viscosity $\nu_B = \Delta x^2 \nu_A$ is derived from the harmonic viscosity ν_A, which itself scales with the squared grid spacing Δx^2, and uses $\nu_{A,0}$ = 500m^{2}s^{-1} at Δx_0 = 30km.

$$\nu_B = \Delta x^2 \nu_A = \frac{\Delta x^4}{\Delta x_0^2}\nu_{A,0} \tag{7}$$

The linear bottom drag is disabled $r = 0$. The time step Δt is chosen to resolve gravity waves, which propagate approximately at phase speed $c_{ph} = \sqrt{gH_0} \approx$ 2.2ms^{-1}. With the Courant-Friedrichs-Lewy number (Courant et al., 1967) $C_{FL} = 0.9$, we use approximately $\Delta t = C_{FL}\Delta x c_{ph}^{-1}$ = 17.9min for Δx = 2.7km. To reduce the diffusion in the tracer advection and for computational efficiency we increase the time step for the semi-Lagrangian scheme to $12\Delta t$. For more details see Klöwer, 2021.

Floating-point numbers and subnormals The 16-bit floating-point number format Float16 is with 1 sign bit, $n_e = 5$ exponent bits and $n_m = 10$ mantissa bits defined as

$$x = \begin{cases} (-1)^s \cdot 2^{e-bias} \cdot (1+f) & \text{if } e > 0, \quad \text{(normals)} \\ (-1)^s \cdot 2^{1-bias} \cdot f & \text{if } e = 0 \text{ and } f > 0, \quad \text{(subnormals)} \\ (-1)^s \cdot 0 & \text{if } e = 0 \text{ and } f = 0, \quad (\pm 0) \\ (-1)^s \cdot \infty & \text{if } e = 31 \text{ and } f = 0, \quad (\pm\text{Inf}) \\ \text{NaN} & \text{if } e = 31 \text{ and } f > 0, \quad \text{(Not-A-Number)} \end{cases} \tag{8}$$

The 5 exponent bits are interpreted as an unsigned integer $e \in \{0, 1, ..., 30, 31\}$ and the subtraction with $bias = 2^{n_e-1} - 1 = 15$ converts them effectively into signed integers, representing positive or negative exponents. The mantissa bits form the fraction $f \in [0, 1)$

$$f = \sum_{j=1}^{n_m} m_j 2^{-j} = \tfrac{m_1}{2} + \tfrac{m_2}{4} + \tfrac{m_3}{8} + ... \tag{9}$$

such that the state ($m_1 = 0$ or 1) of the first mantissa bit determines whether $\frac{1}{2}$ is added to the sum, the second mantissa bit m_2 determines whether $\frac{1}{4}$ is added, etc. The smallest subnormal number, which is also the smallest representable number, often called *minpos*, occurs for all bits being 0 except the last mantissa bit $m_{10} = 1$, then $x = 2^{1-bias} \cdot 2^{-10} = 2^{-24} \approx 6 \cdot 10^{-8}$. The 1023 positive subnormal numbers are linearly distributed between 0 and the smallest normal number, also called *floatmin*, $x = 2^{-14} \approx 6 \cdot 10^{-5}$. The subnormals therefore fill the range between 0 and floatmin and were introduced to avoid underflows in some arithmetic operations with the smallest normal numbers. However, as they are an exception to the definition of normal floats (line 1 in Eq. 8), they also require special treatment on hardware, which reduces performance compared to the normal floats on some processors, e.g. on ARM-based A64FX. The precision of Float16 throughout the range of representable numbers can be analysed with the decimal precision p (Gustafson & Yonemoto, 2017; Klöwer et al., 2020, 2019)

$$p(x_{\mathrm{repr}}, x_{\mathrm{exact}}) = -\log_{10}(\mathrm{abs}(\log_{10}(\frac{x_{\mathrm{repr}}}{x_{\mathrm{exact}}}))) \tag{10}$$

Where x_{exact} is the exact result of an arithmetic operation and x_{repr} the closest representable number in a given format. An arithmetic result that lies halfway between two representable numbers experiences the worst-case rounding error, which determines the worst-case decimal precision for that number. An overview of decimal precisions for Float16 and Float32 is given in Figure 1.

Acknowledgements

This work used the Isambard UK National Tier-2 HPC Service operated by GW4 and the UK Met Office, and funded by the Engineering and Physical Sciences Research Council EPSRC. Valentin Churavy is greatly acknowledged for A64FX-related bug fixes in Julia v1.6. Phil Ridley is acknowledged for fruitful discussion on A64FX. MK gratefully acknowledges

funding from the Natural Environmental Research Council NERC under grant number NE/L002612/1. MK and TNP gratefully acknowledge funding from the European Research Council under grant number 74112. PDD gratefully acknowledges funding from the Royal Society for his University Research Fellowship. SH and PDD have received funding from the ESIWACE2 project (EU Horizon 2020 under grant number 823988). MC acknowledges funding from the ICONIC EPSRC Programme Grant EP/P020720/1.

Author contributions

Conceptualization: MK, SH, MC. Data curation: MK. Formal Analysis: MK. Methodology: MK. Visualization: MK. Writing – original draft: MK. Writing – review & editing: MK, SH, MC, PDD, TNP.

Conflict of interest

The authors declare no conflict of interest.

Data and software availability

ShallowWaters.jl (currently v0.5) is available at https://github.com/milankl/ShallowWaters.jl, Sherlogs.jl (currently v0.2) is available at https://github.com/milankl/Sherlogs.jl. Further software to reproduce the analysis is available at https://github.com/milankl/Isambard (will be converted to a DOI upon acceptance).

References

Arakawa, A., & Hsu, Y.-J. G. (1990). Energy Conserving and Potential-Enstrophy Dissipating Schemes for the Shallow Water Equations. *Monthly Weather Review*, *118*(10), 1960–1969. https://doi.org/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. In J. Chang (Ed.), *Methods in Computational Physics: Advances in Research and Applications* (Vol. 17, pp. 173–265). Elsevier. https://doi.org/10.1016/B978-0-12-460817-7.50009-4

Bauer, P., Dueben, P. D., Hoefler, T., Quintino, T., Schulthess, T. C., & Wedi, N. P. (2021). The digital revolution of Earth-system science. *Nature Computational Science*, *1*(2), 104–113. https://doi.org/10.1038/s43588-021-00023-0

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

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

Butcher, J. C. (2008). Runge–Kutta Methods. In *Numerical Methods for Ordinary Differential Equations* (pp. 137–316). John Wiley & Sons, Ltd. https://doi.org/10.1002/9780470753767.ch3

Chantry, M., Thornes, T., Palmer, T., & Düben, P. (2019). Scale-Selective Precision for Weather and Climate Forecasting. *Monthly Weather Review*, *147*(2), 645–655. https://doi.org/10.1175/MWR-D-18-0308.1

Courant, R., Friedrichs, K., & Lewy, H. (1967). On the Partial Difference Equations of Mathematical Physics. *IBM Journal of Research and Development*, *11*(2), 215–234. https://doi.org/10.1147/rd.112.0215

Croci, M., & Giles, M. B. (2020). Effects of round-to-nearest and stochastic rounding in the numerical solution of the heat equation in low precision. *ArXiv:2010.16225 [Cs, Math]*. http://arxiv.org/abs/2010.16225

Damouche, N., Martel, M., Panchekha, P., Qiu, C., Sanchez-Stern, A., & Tatlock, Z. (2017). Toward a Standard Benchmark Format and Suite for Floating-Point Analysis. In S. Bogomolov, M. Martel, & P. Prabhakar (Eds.), *Numerical Software Verification* (pp. 63–77). Springer International Publishing. https://doi.org/10.1007/978-3-319-54292-8_6

Dawson, A., Düben, P. D., MacLeod, D. A., & Palmer, T. N. (2018). Reliable low precision simulations in land surface models. *Climate Dynamics*, *51*(7), 2657–2666. https://doi.org/10.1007/s00382-017-4034-x

Denis, C., De Oliveira Castro, P., & Petit, E. (2016). Verificarlo: Checking Floating Point Accuracy through Monte Carlo Arithmetic. *2016 IEEE 23nd Symposium on Computer Arithmetic (ARITH)*, 55–62. https://doi.org/10.1109/ARITH.2016.31

Diamantakis, M. (2013). *The semi-Lagrangian technique in atmospheric modelling: Current status and future challenges*. 18.

Dongarra, J., & Luszczek, P. (2011). TOP500. In D. Padua (Ed.), *Encyclopedia of Parallel Computing* (pp. 2055–2057). Springer US. https://doi.org/10.1007/978-0-387-09766-4_157

Fevotte, F., & Lathuilière, B. (2019). Debugging and Optimization of HPC Programs with the Verrou Tool. *2019 IEEE/ACM 3rd International Workshop on Software Correctness for HPC Applications (Correctness)*, 1–10. https://doi.org/10.1109/Correctness49594.2019.00006

Fox, S., Rasoulinezhad, S., Faraone, J., Boland, D., & Leong, P. (2020, September 28). *A Block Minifloat Representation for Training Deep Neural Networks*. International Conference on Learning Representations. https://openreview.net/forum?id=6zaTwpNSsQ2

Gill, A. E. (1982). *Atmosphere-ocean dynamics*. Acad. Press.

Gill, S. (1951). A process for the step-by-step integration of differential equations in an automatic digital computing machine. *Mathematical Proceedings of the Cambridge Philosophical Society*, *47*(1), 96–108. https://doi.org/10.1017/S0305004100026414

Govett, M., Rosinski, J., Middlecoff, J., Henderson, T., Lee, J., MacDonald, A., Wang, N., Madden, P., Schramm, J., & Duarte, A. (2017). Parallelization and Performance of the NIM Weather Model on CPU, GPU, and MIC Processors. *Bulletin of the American Meteorological Society*, *98*(10), 2201–2213. https://doi.org/10.1175/BAMS-D-15-00278.1

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

Hatfield, S., Chantry, M., Düben, P., & Palmer, T. (2019). Accelerating High-Resolution Weather Models with Deep-Learning Hardware. *Proceedings of the Platform for Advanced Scientific Computing Conference*, 1–11. https://doi.org/10.1145/3324989.3325711

Higham, N. J. (1993). The Accuracy of Floating Point Summation. *SIAM Journal on Scientific Computing*, *14*(4), 783–799. https://doi.org/10.1137/0914050

Hopkins, M., Mikaitis, M., Lester, D. R., & Furber, S. (2020). Stochastic rounding and reduced-precision fixed-point arithmetic for solving neural ordinary differential equations. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, *378*(2166), 20190052. https://doi.org/10.1098/rsta.2019.0052

IEEE Standard for Binary Floating-Point Arithmetic. (1985). *ANSI/IEEE Std 754-1985*, 1–20. https://doi.org/10.1109/IEEESTD.1985.82928

IEEE Standard for Floating-Point Arithmetic. (2008). *IEEE Std 754-2008*, 1–70. https://doi.org/10.1109/IEEESTD.2008.4610935

Jansen, M. F., Adcroft, A. J., Hallberg, R., & Held, I. M. (2015). Parameterization of eddy fluxes based on a mesoscale energy budget. *Ocean Modelling*, *92*, 28–41. https://doi.org/10.1016/j.ocemod.2015.05.007

Jézéquel, F., & Chesneaux, J.-M. (2008). CADNA: A library for estimating round-off error propagation. *Computer Physics Communications*, *178*(12), 933–955. https://doi.org/10.1016/j.cpc.2008.02.003

Johnson, J. (2020). Efficient, arbitrarily high precision hardware logarithmic arithmetic for linear algebra. *ArXiv:2004.09313 [Cs, Math]*. http://arxiv.org/abs/2004.09313

Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., Boyle, R., Cantin, P., Chao, C., Clark, C., Coriell, J., Daley, M., Dau, M., Dean, J., Gelb, B., … Yoon, D. H. (2017). In-Datacenter Performance Analysis of a Tensor Processing Unit. *Proceedings of the 44th Annual International Symposium on Computer Architecture*, 1–12. https://doi.org/10.1145/3079856.3080246

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

Kahan, W. (1965). Pracniques: Further remarks on reducing truncation errors. *Communications of the ACM*, *8*(1), 40. https://doi.org/10.1145/363707.363723

Kalamkar, D., Mudigere, D., Mellempudi, N., Das, D., Banerjee, K., Avancha, S., Vooturi, D. T., Jammalamadaka, N., Huang, J., Yuen, H., Yang, J., Park, J., Heinecke, A., Georganas, E., Srinivasan, S., Kundu, A., Smelyanskiy, M., Kaul, B., & Dubey, P. (2019). A Study of BFLOAT16 for Deep Learning Training. *ArXiv:1905.12322 [Cs, Stat]*. http://arxiv.org/abs/1905.12322

Klöwer, M. (2021). *milankl/ShallowWaters.jl: A type-flexible 16-bit shallow water model*. Zenodo. https://doi.org/10.5281/zenodo.4890413

Klöwer, M., Düben, P. D., & Palmer, T. N. (2020). Number Formats, Error Mitigation, and Scope for 16-Bit Arithmetics in Weather and Climate Modeling Analyzed With a Shallow Water Model. *Journal of Advances in Modeling Earth Systems*, *12*(10), e2020MS002246. https://doi.org/10.1029/2020MS002246

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

Langroudi, H. F., Carmichael, Z., & Kudithipudi, D. (2019). Deep Learning Training on the Edge with Low-Precision Posits. *ArXiv:1907.13216 [Cs, Stat]*. http://arxiv.org/abs/1907.13216

Linnainmaa, S. (1974). Analysis of some known methods of improving the accuracy of floating-point sums. *BIT Numerical Mathematics*, *14*(2), 167–202. https://doi.org/10.1007/BF01932946

Mikaitis, M. (2020). Stochastic Rounding: Algorithms and Hardware Accelerator. *ArXiv:2001.01501 [Cs, Math]*. http://arxiv.org/abs/2001.01501

Møller, O. (1965). Quasi double-precision in floating point addition. *BIT Numerical Mathematics*, *5*(1), 37–50. https://doi.org/10.1007/BF01975722

Nakano, M., Yashiro, H., Kodama, C., & Tomita, H. (2018). Single Precision in the Dynamical Core of a Nonhydrostatic Global Atmospheric Model: Evaluation Using a Baroclinic Wave Test Case. *Monthly Weather Review*, *146*(2), 409–416. https://doi.org/10.1175/MWR-D-17-0257.1

Odajima, T., Kodama, Y., Tsuji, M., Matsuda, M., Maruyama, Y., & Sato, M. (2020). Preliminary Performance Evaluation of the Fujitsu A64FX Using HPC Applications. *2020 IEEE International Conference on Cluster Computing (CLUSTER)*, 523–530. https://doi.org/10.1109/CLUSTER49012.2020.00075

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

Klöwer et al. 2021. *Fluid simulations accelerated with 16 bit: Approaching 4x speedup on A64FX*, submitted.

Paxton, E. A., Chantry, M., Klöwer, M., Saffin, L., & Palmer, T. (2021). Climate Modelling in Low-Precision: Effects of both Deterministic & Stochastic Rounding. *ArXiv*. http://arxiv.org/abs/2104.15076

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

Sato, M., Ishikawa, Y., Tomita, H., Kodama, Y., Odajima, T., Tsuji, M., Yashiro, H., Aoki, M., Shida, N., Miyoshi, I., Hirai, K., Furuya, A., Asato, A., Morita, K., & Shimizu, T. (2020). Co-Design for A64FX Manycore Processor and "Fugaku". *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, 1–15. https://doi.org/10.1109/SC41405.2020.00051

Smolarkiewicz, P. K., & Pudykiewicz, J. A. (1992). A Class of Semi-Lagrangian Approximations for Fluids. *Journal of the Atmospheric Sciences*, *49*(22), 2082–2096. https://doi.org/10.1175/1520-0469(1992)049<2082:ACOSLA>2.0.CO;2

Steinkraus, D., Buck, I., & Simard, P. Y. (2005). Using GPUs for machine learning algorithms. *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*, 1115-1120 Vol. 2. https://doi.org/10.1109/ICDAR.2005.251

Sun, X., Wang, N., Chen, C., & Ni, J. (2020). *Ultra-Low Precision 4-bit Training of Deep Neural Networks*. 12.

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

Vallis, G. K. (2006). *Atmospheric and Oceanic 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. https://doi.org/10.1175/MWR-D-16-0228.1

Vitasek, E. (1969). The numerical stability in solution of differential equations. In J. Li. Morris (Ed.), *Conference on the Numerical Solution of Differential Equations* (pp. 87–111). Springer. https://doi.org/10.1007/BFb0060017

Zhang, H., & Ko, S.-B. (2020). Design of Power Efficient Posit Multiplier. *IEEE Transactions on Circuits and Systems II: Express Briefs*, *67*(5), 861–865. https://doi.org/10.1109/TCSII.2020.2980531

Zheng, W. (2020). Research trend of large-scale supercomputers and applications from the TOP500 and Gordon Bell Prize. *Science China Information Sciences*, *63*(7), 171001. https://doi.org/10.1007/s11432-020-2861-0

Supplementary Figures

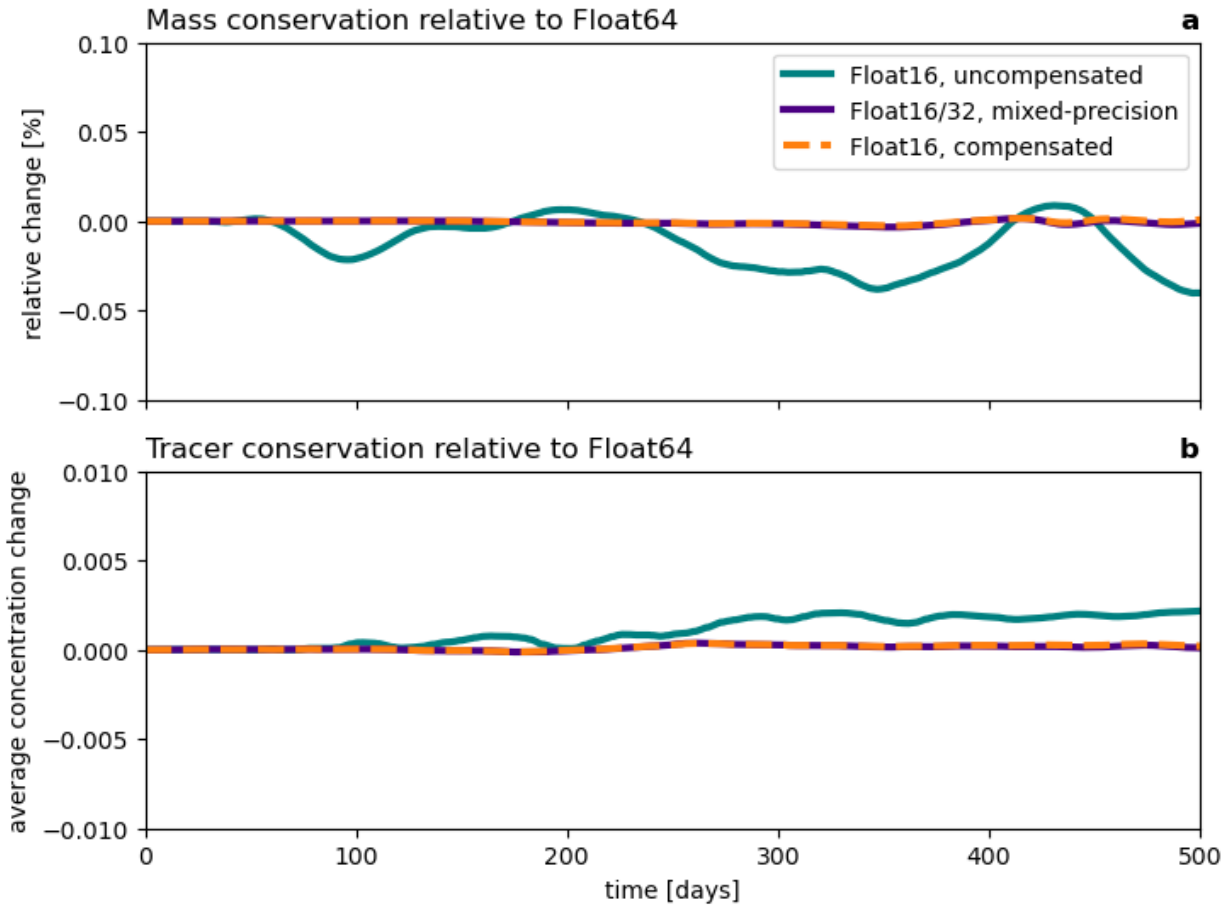

Figure S1 | Mass and tracer conservation with Float16 arithmetic. a Mass conservation relative to Float64. **b** Tracer conservation relative to Float64 in units of tracer concentration with initial conditions in (-1,1), see Fig. 5. Both mass and tracer are well conserved with Float16 arithmetic. Best conservations are obtained with compensated time integration or mixed-precision.