An Emulator for Reduced Floating-Point Precision in Large Numerical Simulations

Andrew Dawson and Peter Düben
Atmospheric, Oceanic & Planetary Physics, University of Oxford, Oxford UK

1. The Role of Inexact Computations in Numerical Simulations

Weather forecasting, and many other fields that rely on high performance computing, have enjoyed the benefits of continued supercomputing power increases, allowing model complexity and cost to increase steadily with time. However, the next generation of supercomputing technology, the exascale computer, is out of reach with current technology largely due to the anticipated excessive power demands. Continued improvement in supercomputing performance will require both making more efficient use of the current technologies and the design of new hardware architectures that can avoid the power issues.

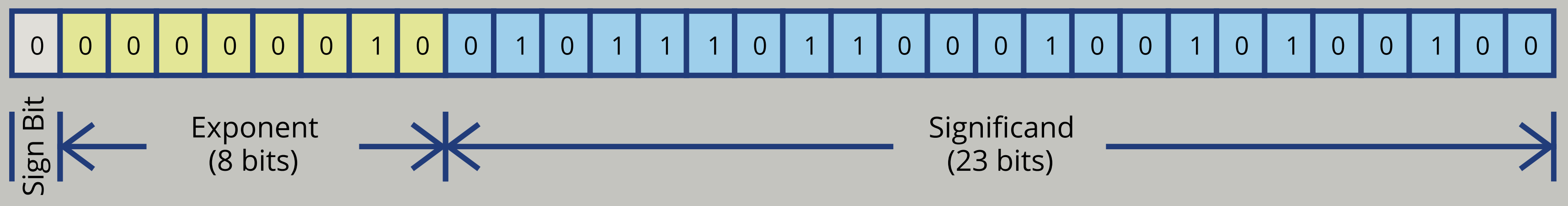
One option for next-generation hardware is to design energy-efficient domain-specific hardware or leverage existing technologies (e.g., GPUs) that can perform floating-point calculations with a reduced number of bits, or with some non-zero probability of errors. Many parts of our numerical models may be over-engineered in terms of floating-point precision, using 64-bit double precision for algorithms where the solution may be able to be computed to acceptable accuracy with many fewer bits. This makes reduced precision or inexact floating-point calculations an attractive option for weather forecast models. Savings made by reducing floating-point precision could be reinvested allowing models to run at higher resolution, include more Earth-system complexity, or use larger ensemble sizes.

2. The Representation of Floating-Point Numbers

Computers use floating-point numbers to represent arbitrary decimal numbers. A floating-point number is represented as a group of 32 bits (or 64 bits for double precision). The bits are divided into groups, each having a different meaning:

- **Sign Bit:** The left-most bit is the sign-bit which determines if the number is positive or negative. If the bit is set (has value 1) then the number is negative, if the bit is not set (has value 0) the number is positive.
- **Exponent:** The next 8 bits (11 for double precision) are called the exponent and determine the magnitude of the number.
- **Significand:** The remaining 23 bits (52 for double precision) are called the significand (also called the mantissa) and determine the digits that make up the number. It is the number of bits in the significand that determine the precision to which decimal numbers can be represented.

Single Precision Floating-Point in Binary

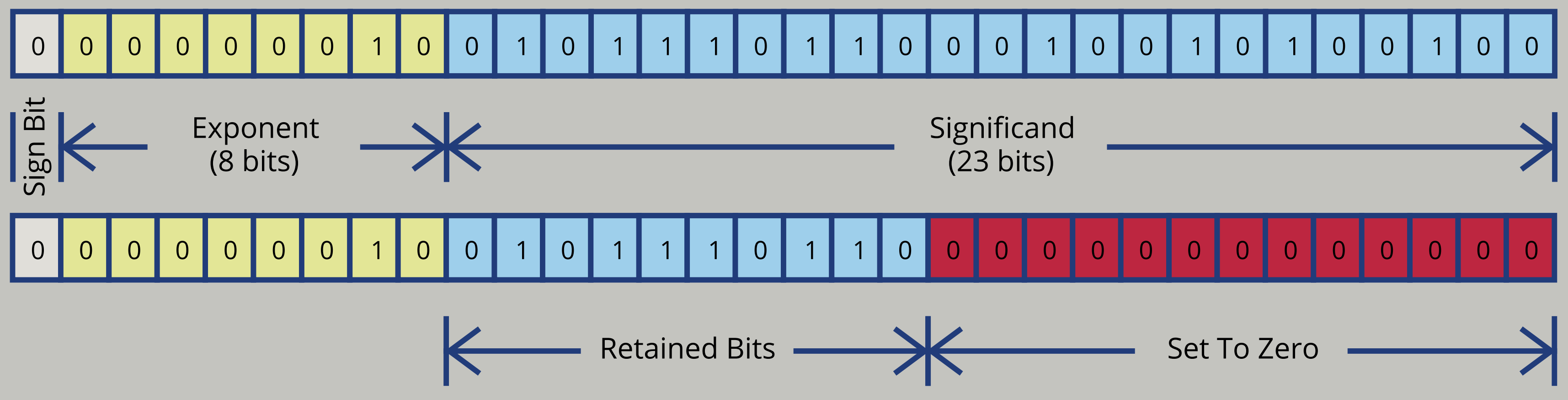


3. Reduced Floating-Point Precision on Standard Hardware

Reducing the precision of a floating-point number can be done simply by reducing the number of bits used to represent it. Conventional CPU hardware typically limits us to using either 32-bit or 64-bit floating-point numbers, meaning support for arbitrary representations is not readily available.

A relatively straightforward way to approach this problem is to emulate the reduced precision in software, whilst still using conventional 32- or 64-bit floating-point representations in hardware. The simplest emulation strategy is to truncate the number of bits in the significand of the floating-point number by setting a given number of the least significant bits to zero.

Truncating The Significand to 10 bits



Rounding is performed if the left-most bit to be zeroed is set. This is done to avoid introducing biases caused by always rounding in the same direction.

4. Emulating Reduced Precision in Fortran Programs

We have created a Fortran library ***rpe*** [1] that is designed to introduce reduced-precision floating-point arithmetic into existing Fortran programs with only limited changes to the code. The library defines a new data type `type(rpe_var)` that can be used in place of Fortran's `real` data type.

Example: A simple low-precision program

Consider a simple program that performs a few floating-point operations at single precision:

```
program full_precision
    real :: rho, g, h, P
    rho = 1.2041
    g = 9.80665
    h = 10
    P = rho * g * h
    write (*, *) P
end program
```

This program could be re-written making use of the ***rpe*** library, so that floating-point precision is reduced to only 10 bits in the significand for all computations:

```
program full_precision
    use rp_emulator                    ! load the rpe module
    type(rpe_var) :: rho, g, h, P     ! use type(rpe_var) instead of real
    RPE_DEFAULT_SBITS = 10             ! set the default working precision to 10 bits
    rho = 1.2041
    g = 9.80665
    h = 10
    P = rho * g * h
    write (*, *) P%val                 ! use the value attribute for printing
end program
```

The second program produces the result `118.0625`, whereas the full precision program produces `118.081879`.

The modifications required to work with reduced precision in an existing program are minimal, and the procedure for doing so scales nicely even to large programs with many thousands of lines of code.

5. How the Emulator Works

The emulator works by overloading the assignment (=) operator so that when a value is assigned to an `type(rpe_var)` instance the value is truncated to the required number of bits. ***rpe*** also provides implementations of all arithmetic and logic operators (+, −, *, /, **, ==, /=, >, <, >=, <=) for the `rpe_var` type that are interoperable with the existing built in types `integer` and `real`. In addition ***rpe*** provides versions of many Fortran intrinsic functions that operate on `type(rpe_var)` instances.

Overloaded definitions that would normally return a `real` type use overloaded assignments to return a reduced precision `type(rpe_var)` value. Compound expressions are handled implicitly, with each intermediate value being reduced in precision before computing the next.

Example: Precision of compound operations

Consider the following Fortran code using 10-bit precision:

```
type(rpe_var) :: p, x, y, z
RPE_DEFAULT_SBITS = 10
x = 1.2
y = 3.8
z = 2.1
p = x * y + z
```

The computations performed by this block of code are given below, where R is an operator that reduces the precision of a floating-point value to a 10-bit significand:

```
x = R( 1.2 )
y = R( 3.8 )
z = R( 2.1 )
p = R( R( x * y ) + z)
```

The precision can be controlled globally via `RPE_DEFAULT_SBITS` or by setting the `sbits` attribute of an `type(rpe_var)` instance. This allows the construction of programs that use different precision levels for different parts of the code.

6. A Geophysical Application

The emulator has been applied to a shallow-water equation model run in a Munk double-gyre test case [2, 3]. The model is run in serveral configurations: 64-bit floating-point arithmetic (control), with emulated 15-bit precision, and with emulated 10-bit precision.

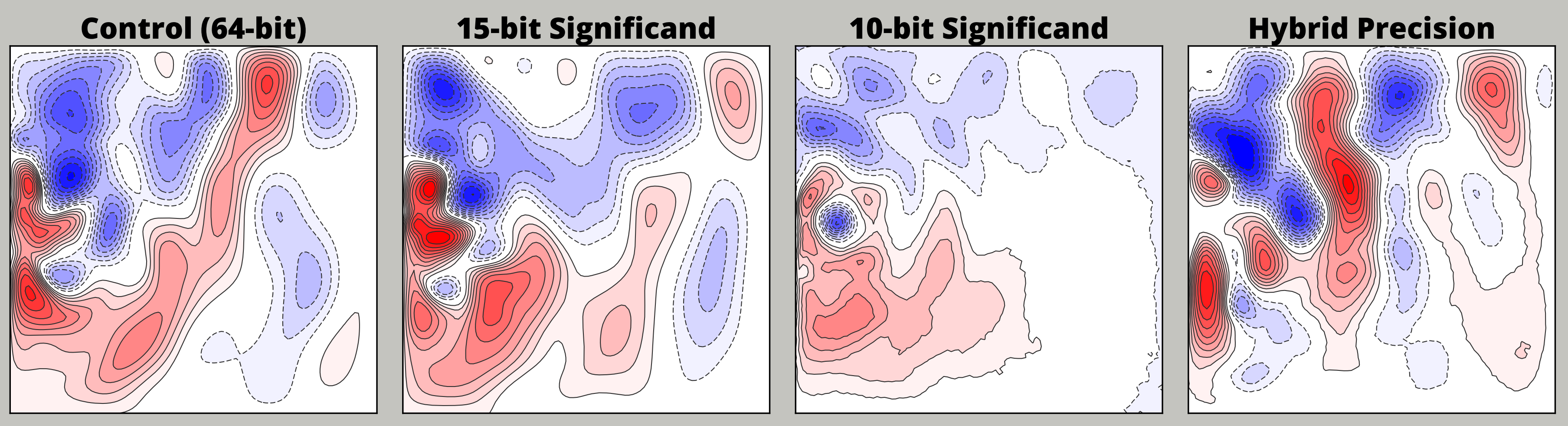


Figure : Snapshots of surface height some time after spin-up. The double-precision and 15-bit significand configurations have smooth height fields. The 10-bit configuration is behaving somewhat differently.

The model performs acceptably with 15-bit significand arithmetic, but 10-bits appears to be too low. The problem is that the increments required to update the height field are often too small to be added to the model state when run at low enough precision. We can get around this problem by computing the right-hand-sides of the discretized equations in half precision (10-bit significand, 5-bit exponent), but doing the update to the state in higher precision. This is referred to as hybrid precision.

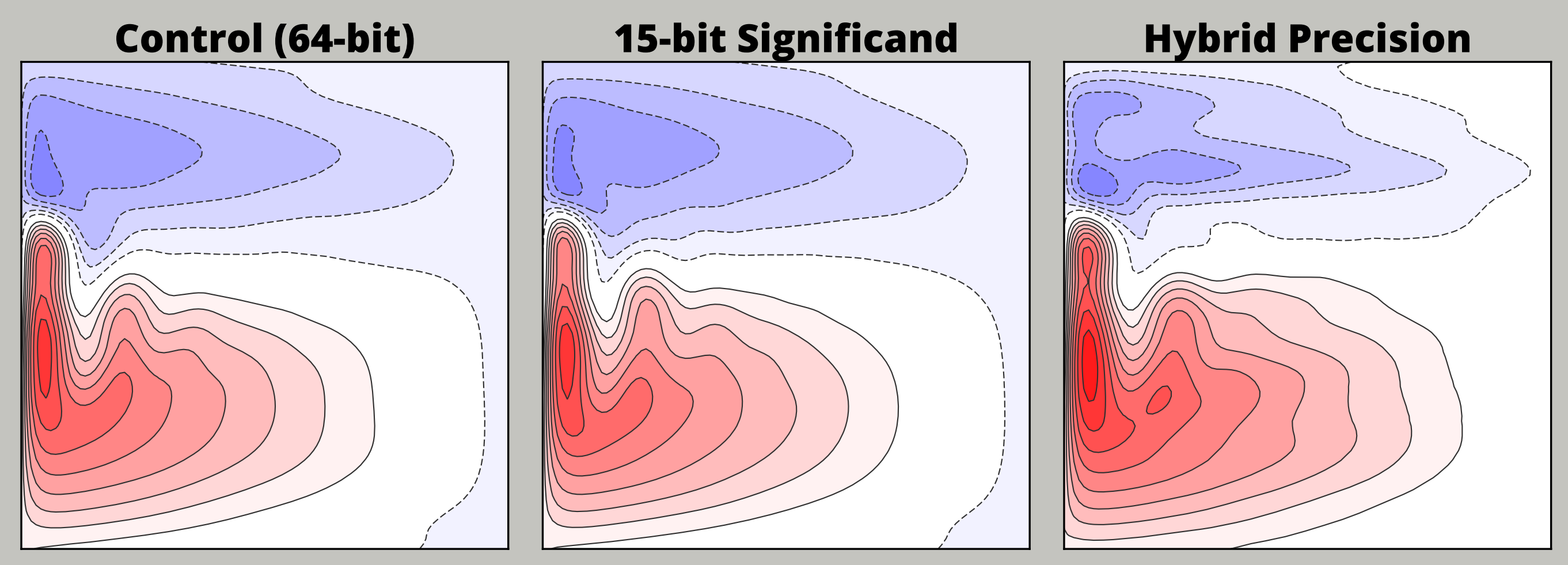


Figure : Time-mean surface height fields. The hybrid precision (IEEE half-precision for RHS, higher precision for state) performs very well compared to a reference double precision model, and a 15-bit significand simulation.

7. Summary and More Information

We have presented a software emulator ***rpe*** which can be used to perform numerical experiments with reduced floating-point precision, and demonstrated its efficacy using an idealised geophysical model. The tool can be applied to any numerical simulation written in Fortran90+.

The ***rpe*** software is free and open-source:

- the source code can be obtained from `http://github.com/aopp-pred/rpe`
- documentation including a user's guide is available online at `http://rpe.readthedocs.io`.

References

[1] A. Dawson and P. D. Düben. rpe v5: An emulator for reduced floating-point precision in large numerical simulations. *Geosci. Model Dev.*, 2016. In preparation.
[2] D. P. Marshall, B. Vogel, and Xiaoming Zhai. Rossby rip currents. *Geophys. Res. Let.*, 40:4333–4337, 2013. doi: 10.1002/grl.50842.
[3] F. C. Cooper and L. Zanna. Optimisation of an idealised ocean model, stochastic parameterisation of sub-grid eddies. *Ocean Model.*, 88:38–53, 2015. doi: 10.1016/j.ocemod.2014.12.014.

http://www2.physics.ox.ac.uk/contacts/people/dawson
andrew.dawson@physics.ox.ac.uk