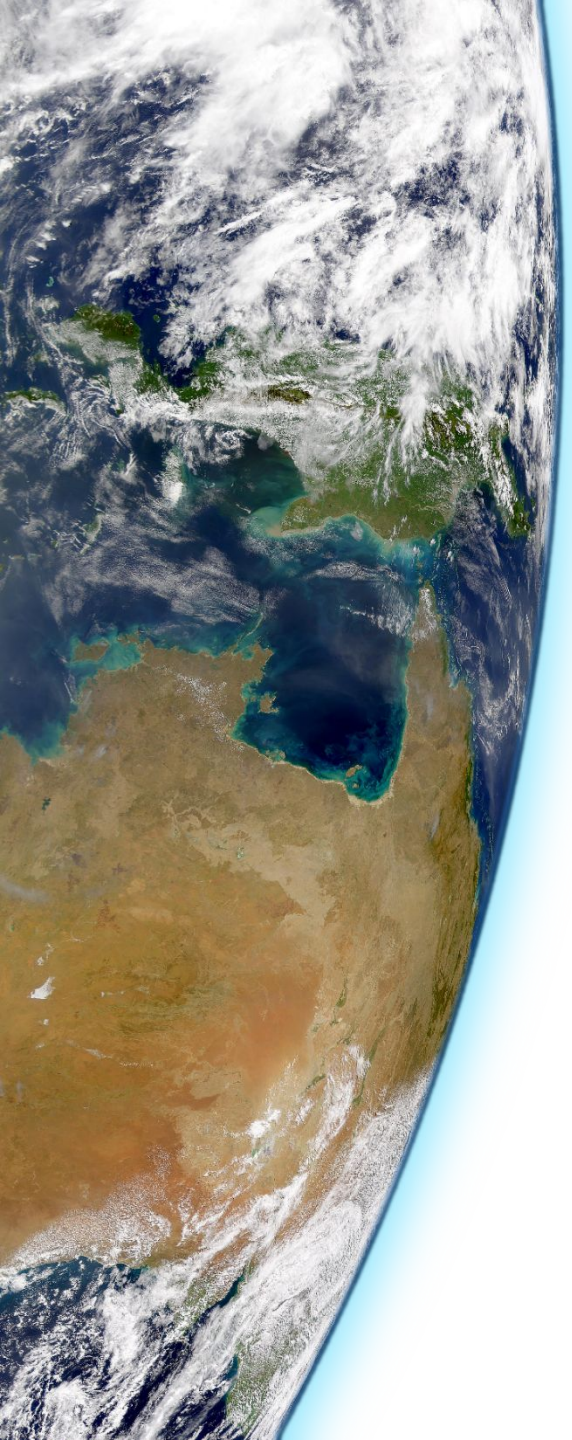


Advances on mixed precision and its impact on developers

NEMO HPC WG

28/07/2020

Oriol Tintó Prims

BSC Earth Sciences Department

esiwace2
CENTRE OF EXCELLENCE IN SIMULATION OF WEATHER AND CLIMATE IN EUROPE

is-enes
INFRASTRUCTURE FOR THE EUROPEAN NETWORK FOR EARTH SYSTEM MODELLING

In previous seasons...

- Optimizing numerical precision **improves computational performance**.
- We developed a method to **identify the precision required** for each real variable.
- Some prototypes demonstrated that **NEMO can achieve performance gains maintaining double-precision accuracy**.
- A development branch was created with the following implementation plan:
    - We are using a new **key_single** which will set the **working precision to single**. Compiling without this key will yield a version of the code in double precision that **shall pass all the sette tests**.

Advances on Mixed Precision

Some of the developments that were on the branch were merged in this mid-year merge party. At this stage the branch had:

- A lot of trivial changes (adding type specification to hardcoded reals **1.0 -> 1.0_wp** , ...).
- Communication and IO interfaces duplicated to work with both single and double.
- A few non trivial changes to avoid floating point exceptions, mainly in the sea-ice.

It compiles and runs **without crashing** in single and double precision. Passing all the sette tests (in double).

It might result sufficient for some applications.

What is missing?

- Setting sensitive variables to **double-precision**.

- Extensive testing.

What is missing?

- Setting sensitive variables to **double-precision**:
  - We identified sensitive variables in previous versions of NEMO. We need to check/update this knowledge with the latest version of NEMO.
  - Un-entangle variables.
- Extensive testing.

What is missing?

- Setting sensitive variables to **double-precision**:
  - We identified sensitive variables in previous versions of NEMO. We need to check/update this knowledge with the latest version of NEMO.
  - Un-entangle variables.
- Extensive testing:
  - Validity of results.
  - Performance.

Entanglement

Setting sensitive variables to double-precision:

```
real(wp) :: x

call my_function(x)
  ...
function my_function(y)
   real(wp) :: y
end function my_function
```

Entanglement

Setting sensitive variables to double-precision:

```
real(wp) :: x

call my_function(x)
  ...
function my_function(y)
   real(wp) :: y
end function my_function
```

X and **Y** must have the same type!

Entanglement

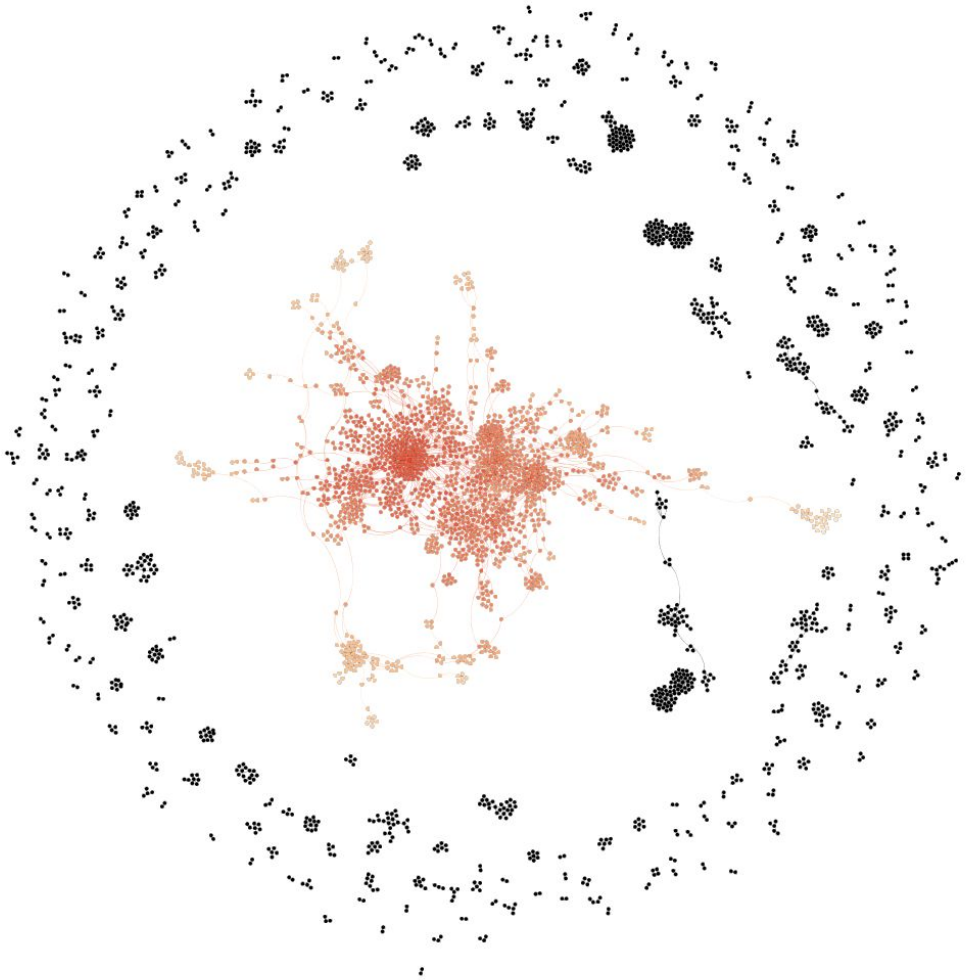
Setting sensitive variables to double-precision:

- Our tool **can automatically detect which variables need to keep the same real type** and modify it accordingly in the code.
- Providing a list of variables to keep in double precision we can obtain a version of the code that compiles and runs.

However, the variables are **too entangled**!

Entanglement

Setting sensitive variables to double-precision:



NEMO real variables and its connections

Each dot represents one real variable and each edge a connection between them.

The red cluster shows a huge group of interconnected variables which makes difficult to change the precision of only a subset of them.

.

Entanglement

Setting sensitive variables to double-precision:

- To **untie the big cluster of variables** it is necessary to have some **type-agnostic** subroutines:
  - Was straightforward for communication routines because it was possible to exploit the template based preprocessor strategy that already existed.
  - It might be necessary to define a clean way to do it with other routines.
    - Duplicated routines?
    - Preprocessor directives like in lbc_lnk?
    - Polymorphic subroutines?
    - …
- ... or/and to cast some input variables on routine calls.

Work plan

- **Update the analysis** with post-merge NEMO version.
- Obtain a **up-to-date mixed-precision version.**
- Iterate to **refine this version** by fixing entanglement issues.

Implications for developers

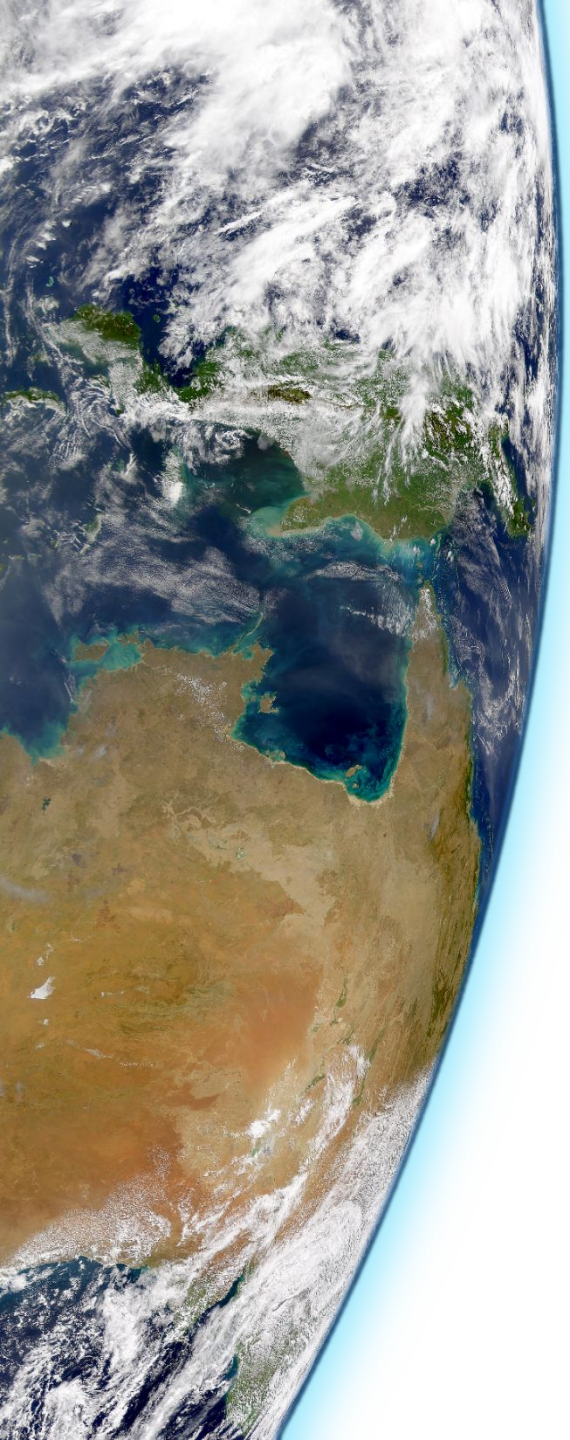
- New developments should keep using **wp** as a default.
- If using variables that are defined in other parts of the code, it could be a good practice to check which precision these variables are.
- **dp** can be used when there's a certainty that an algorithm requires double precision.
- If there's the suspicion, **it can be checked** by comparing two simulations using the two precision settings.
- It could be good to add some test to assess that developments compile in both precision settings.
- Ideally after the next merge-party, instead of using **key_single** we can use **key_double**, letting the optimized-precision version to be the standard.

Bonus

- The experience running with single-precision highlighted that some bugs that remain hidden in double-precision can be more easily spotted in single. Helping to make the code more consistent.
- In a similar way, when we see code regions that do not show any performance improvement when optimizing the numerical precision, it might lead us to performance issues that we were not aware of.

Take home messages

- Development is **ongoing** and will be finished this year.
- Some implementation details need to be defined. Having feedback and suggestions **will be very appreciated**.
- We want to minimize the inconveniences to developers and feedback about how to do it **will be very appreciated too.**



Thank you!

oriol.tinto@bsc.es