IS-ENES3 Deliverable D8.5

Update of the NEMO code

Authors: Italo Epicoco (CMCC), Silvia Mocavero (CMCC), Francesca Mele (CMCC), Stella Paronuzzi (BSC), Mario Acosta (BSC), Miguel Castrillo (BSC)

Reviewer: Daley Clavert (MetOffice)

Release date: 15/11/2022

ABSTRACT

The document reports the updates of the NEMO code as designed in the Milestone M8.4 "Definition of the NEMO optimization strategy". This document describes the main changes to the NEMO code included in the last revision NEMO v4.2 available in the NEMO git repository since March 2022.
The most important developments regard the performance optimization through a redesign of the communication strategy for the halo update and the use of mixed precision in the NEMO model.

Dissemination Level		
PU	Public	X
CO	Confidential, only for the partners of the IS-ENES3 project	

Revision table			
Version	**Date**	**Name**	**Comments**
1.0	28/10/2022	Italo Epicoco	First version sent to internal reviewer
2.0	09/11/2022	Calvert Daley	Comments and improvements
3.0	11/11/2022	Italo Epicoco	Final version

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 824084

Table of contents

1 Executive Summary

The developments carried out within the first three years of the IS-ENES3 project to improve NEMO performance have been fully integrated in the NEMO code and released with the official NEMO release 4.2.0.
The NEMO site reports the changes of this release compared with the previous one (https://sites.nemo-ocean.io/user-guide/changes.html). With regard to the HPC developments performed within WP8 of the IS-ENES3 project and fully described in M8.4, the new release includes:

1. A code cleanup to reduce the number of MPI communications
2. Extension of the halo to reduce the frequency of MPI communications
3. The integration of MPI3 collective neighboring communication for the halo exchange, a preliminary step to reduce the number of exchange where 5-points stencil is enough
4. The mixed precision preparatory phase to speed up the model run by using the least number of significant bits possible.

2 Objectives

The main objective of this document is to describe the development of the actions defined in Milestone 8.4 needed to carry out the performance optimization of the NEMO code proposed in the IS-ENES3 DoW for WP8-Task1.

The proposed changes were discussed with the NEMO System Team and the HPC Working Group prior to their implementation in NEMO. The role of this group was to verify that the proposed optimization strategy matched the NEMO development strategy document (https://zenodo.org/record/1472458#.Y0rTFy0Ro_U). The activities were then included as actions in the NEMO development work plan (https://forge.ipsl.jussieu.fr/nemo/wiki/2020WP). Development branches were created for the actions (https://forge.ipsl.jussieu.fr/nemo/browser#NEMO/branches), tested using the NEMO reference configurations and test cases, internally reviewed, and finally merged in the official 4.2.0 release of NEMO.

Section 2 reports the details of these developments and their code changes, while the last section introduces some actions planned for the future.

3 Code changes in NEMO v4.2

3.1 MPI Communication cleanup (ticket #2607)

The whole code has been revised in order to remove unnecessary communications. Some communications to update the halo of neighboring parallel processes are no longer needed, due to code evolution. For example, some communications performed before writing the output files have been deleted since the halo region is no longer included in these files.
In other cases, this work was a prerequisite for the application of other optimizations. Some communications have been removed or moved earlier in the code by introducing an extended halo in *lbc_lnk*, the interface implementing MPI exchanges with neighboring processors in the NEMO code. This allows developers to reduce the frequency of communications by moving the communication outside of DO loops (where possible) and to implement other optimizations such as tiling or loop fusion. Moving the communications outside of DO loops requires a deep analysis of the algorithm implemented in each routine and, to date, has been performed on the Tracer Advection module (Changeset 14609), the Vertical Ocean Physics module (Changeset 14601) and most of the Ocean Dynamics module routines (Changeset 14682).
Following is an example of the analysis and implementation process that has been carried out on the code. Taking into consideration the original *dyn_keg* routine in Figure 1 (Hollingsworth scheme case, 1-point halo width), the horizontal kinetic energy *zhke* is computed on the inner domain (*DO_3D(0, 0, 0, 0, ...)*) and its halo is then updated via an *lbc_lnk*. Finally, the whole array is used to update the velocities *puu* and *pvv*.

```
SUBROUTINE dyn_keg( kt, kscheme, Kmm, puu, pvv, Krhs )
 ...
   REAL(wp), DIMENSION(jpi,jpj,jpk)        ::   zhke
   zhke(:,:,jpk) = 0._wp
     ...
   SELECT CASE ( kscheme )             !== Horizontal kinetic energy at T-point  ==!
   !
   CASE ( nkeg_C2 )                          !--  Standard scheme  --!
     ...
   CASE ( nkeg_HW )                          !--  Hollingsworth scheme  --!
      DO_3D( 0, 0, 0, 0, 1, jpkm1 )
         zu = 8._wp * ( puu(ji-1,jj  ,jk,Kmm) * puu(ji-1,jj  ,jk,Kmm)    &
            &         + puu(ji  ,jj  ,jk,Kmm) * puu(ji  ,jj  ,jk,Kmm) )  &
            &   +     ( puu(ji-1,jj-1,jk,Kmm) + puu(ji-1,jj+1,jk,Kmm) ) * ( puu(ji-1,jj-1,jk,Kmm) + puu(ji-1,jj+1,jk,Kmm) )
            &   +     ( puu(ji  ,jj-1,jk,Kmm) + puu(ji  ,jj+1,jk,Kmm) ) * ( puu(ji  ,jj-1,jk,Kmm) + puu(ji  ,jj+1,jk,Kmm) )
            !
         zv = 8._wp * ( pvv(ji  ,jj-1,jk,Kmm) * pvv(ji  ,jj-1,jk,Kmm)    &
            &         + pvv(ji  ,jj  ,jk,Kmm) * pvv(ji  ,jj  ,jk,Kmm) )  &
            &   +     ( pvv(ji-1,jj-1,jk,Kmm) + pvv(ji+1,jj-1,jk,Kmm) ) * ( pvv(ji-1,jj-1,jk,Kmm) + pvv(ji+1,jj-1,jk,Kmm) )
            &   +     ( pvv(ji-1,jj  ,jk,Kmm) + pvv(ji+1,jj  ,jk,Kmm) ) * ( pvv(ji-1,jj  ,jk,Kmm) + pvv(ji+1,jj  ,jk,Kmm) )
         zhke(ji,jj,jk) = r1_48 * ( zv + zu )
      END_3D
      CALL lbc_lnk( 'dynkeg', zhke, 'T', 1.0_wp )
      !
   END SELECT
   !
   DO_3D( 0, 0, 0, 0, 1, jpkm1 )
      puu(ji,jj,jk,Krhs) = puu(ji,jj,jk,Krhs) - ( zhke(ji+1,jj  ,jk) - zhke(ji,jj,jk) ) / e1u(ji,jj)
      pvv(ji,jj,jk,Krhs) = pvv(ji,jj,jk,Krhs) - ( zhke(ji  ,jj+1,jk) - zhke(ji,jj,jk) ) / e2v(ji,jj)
   END_3D
     ...
END SUBROUTINE dyn_keg
```

Figure 1 - Original version of dyn_keg routine

If a 2-point halo width is used, the *lbc_lnk* communication can instead be performed on the velocities *puu* and *pvv* before the first *DO_3D*. Their updated halo values can then be used for the computation of horizontal kinetic energy *zhke* in the range [2:jpi-1,2:jpj-1,1:jpk-1] (*DO_3D(1, 1, 1, 1, 1, jpkm1)*), which can then be used to update the velocities *puu* and *pvv*.

```
SUBROUTINE dyn_keg( kt, kscheme, Kmm, puu, pvv, Krhs )
 ...
   REAL(wp), DIMENSION(jpi,jpj,jpk)        ::   zhke
   zhke(:,:,jpk) = 0._wp
     ...
   IF( nn_hls.EQ.2 ) CALL lbc_lnk_multi( 'dynkeg', puu(:,:,:,Kmm), 'U', -1.0_wp, pvv(:,:,:,Kmm), 'V', -1.0_wp)
   SELECT CASE ( kscheme )             !== Horizontal kinetic energy at T-point  ==!
   !
   CASE ( nkeg_C2 )                          !--  Standard scheme  --!
     ...
   CASE ( nkeg_HW )                          !--  Hollingsworth scheme  --!
      DO_3D( 1, 1, 1, 1, 1, jpkm1 )
         zu = 8._wp * ( puu(ji-1,jj  ,jk,Kmm) * puu(ji-1,jj  ,jk,Kmm)    &
            &         + puu(ji  ,jj  ,jk,Kmm) * puu(ji  ,jj  ,jk,Kmm) )  &
            &   +     ( puu(ji-1,jj-1,jk,Kmm) + puu(ji-1,jj+1,jk,Kmm) ) * ( puu(ji-1,jj-1,jk,Kmm) + puu(ji-1,jj+1,jk,Kmm) )
            &   +     ( puu(ji  ,jj-1,jk,Kmm) + puu(ji  ,jj+1,jk,Kmm) ) * ( puu(ji  ,jj-1,jk,Kmm) + puu(ji  ,jj+1,jk,Kmm) )
            !
         zv = 8._wp * ( pvv(ji  ,jj-1,jk,Kmm) * pvv(ji  ,jj-1,jk,Kmm)    &
            &         + pvv(ji  ,jj  ,jk,Kmm) * pvv(ji  ,jj  ,jk,Kmm) )  &
            &   +     ( pvv(ji-1,jj-1,jk,Kmm) + pvv(ji+1,jj-1,jk,Kmm) ) * ( pvv(ji-1,jj-1,jk,Kmm) + pvv(ji+1,jj-1,jk,Kmm) )
            &   +     ( pvv(ji-1,jj  ,jk,Kmm) + pvv(ji+1,jj  ,jk,Kmm) ) * ( pvv(ji-1,jj  ,jk,Kmm) + pvv(ji+1,jj  ,jk,Kmm) )
         zhke(ji,jj,jk) = r1_48 * ( zv + zu )
      END_3D
      IF( nn_hls.EQ.1 ) CALL lbc_lnk( 'dynkeg', zhke, 'T', 1.0_wp )
      !
   END SELECT
   !
   DO_3D( 0, 0, 0, 0, 1, jpkm1 )
      puu(ji,jj,jk,Krhs) = puu(ji,jj,jk,Krhs) - ( zhke(ji+1,jj  ,jk) - zhke(ji,jj,jk) ) / e1u(ji,jj)
      pvv(ji,jj,jk,Krhs) = pvv(ji,jj,jk,Krhs) - ( zhke(ji  ,jj+1,jk) - zhke(ji,jj,jk) ) / e2v(ji,jj)
   END_3D
     ...
END SUBROUTINE dyn_keg
```

Figure 2 - Modified version of dyn_keg routine

These changes, shown in Figure 2, do not alter results in the GYRE PISCES configuration, but produce different results in the ORCA2 ICE_PISCES reference configuration from time step 23 onwards (REPRO84 test) compared with tests using a 1-point halo width, although restartability and reproducibility are preserved.
This implies that the change in results is related to the NORTH FOLD treatment.
It is worth recalling that the differences between the 1- and 2-point haloes are limited to the computation of values held in the first halo line. Indeed, in the 1-point halo case these values are received from the neighbor (or from the corresponding rank in north fold), while in the 2-point halo case these values are computed locally.
Considering an 8x4 domain decomposition with jpi=25 and jpj=39, in the original 1-point halo case the *hdiv* value for the halo point at (21i, 39j) on processor 29 is sent by processor 26 after computing its inner point at (3i, 37j).
When a 2-point halo is instead used, the corresponding halo point at (22i ,40j) is directly computed by processor 29.
So, it is expected that

$$hdiv_{[22,40]} on\ proc\ 29\ (halo\ 2\ case)\ =\ hdiv_{[3,37]} on\ proc\ 26\ (halo\ 1\ case)$$

Looking at the values of the u- and v-grid variables contributing to the calculation of the horizontal divergence *hdiv* at these points, it can be noted that they are arranged in a symmetric/mirrored way in the 2-point halo case compared to the 1-point halo case. Table 1 reports the corresponding values along with the halo region.

Table 1 - Values contributing to the horizontal divergence hdiv in the 1- and 2-point halo cases

$hdiv_{[3,39]}$- P26 (halo 1 case)	$hdiv_{[22,40]}$- P29 (halo 2 case)
$e1v_{[i,j]}$	$-e1v_{[i,j-1]}$
$e1v_{[i,j-1]}$	$-e1v_{[i,j]}$
$e2u_{[i,j]}$	$-e2u_{[i-1,j]}$
$e2u_{[i-1,j]}$	$-e2u_{[i,j]}$
$e3u_{[i,j,k,Kmm]}$	$-e3u_{[i-1,j,k,Kmm]}$
$e3u_{[i-1,j,k,Kmm]}$	$-e3u_{[i,j,k,Kmm]}$
$e3v_{[i,j,k,Kmm]}$	$-e3v_{[i,j-1,k,Kmm]}$
$e3v_{[i,j-1,k,Kmm]}$	$-e3v_{[i,j,k,Kmm]}$
$e3u_{[i,j,k,Kmm]}$	$-e3u_{[i-1,j,k,Kmm]}$
$e3u_{[i-1,j,k,Kmm]}$	$-e3u_{[i,j,k,Kmm]}$
$vv_{[i,j,k,Kmm]}$	$-vv_{[i,j-1,k,Kmm]}$
$vv_{[i,j-1,k,Kmm]}$	$-vv_{[i,j,k,Kmm]}$
$r1\_e1e2t_{[i,j]}$	$r1\_e1e2t_{[i,j]}$
$e3t_{[i,j,k,Kmm]}$	$e3t_{[i,j,k,Kmm]}$

More generally, it can be asserted that when we compute the values on the first line of the halo (which happens only in the 2-point halo case) the following changes happen when an expression is evaluated:

- For the U-grid fields:
  - the sign changes
  - index i turns to i-1
  - index i-1 turns to i
  - index j+1 turns to j-1
  - index j-1 turns to j+1
- For the V-grid fields:
  - the sign changes
  - index j turns to j-1
  - index j-1 turns to j
  - index i+1 turns to i-1
  - index i-1 turns to i+1

After applying these changes to the *hdiv* expression, the code changes as shown in Figure 3 and Figure 4.

```
DO_3D( nn_hls-1, nn_hls, nn_hls-1, nn_hls, 1, jpkm1 )     !== Horizontal divergence ==!
   hdiv(ji,jj,jk) = (  e2u(ji  ,jj  ) * e3u(ji  ,jj  ,jk,Kmm) * uu(ji  ,jj  ,jk,Kmm) &
      &              - e2u(ji-1,jj  ) * e3u(ji-1,jj  ,jk,Kmm) * uu(ji-1,jj  ,jk,Kmm) &
      &              + e1v(ji  ,jj  ) * e3v(ji  ,jj  ,jk,Kmm) * vv(ji  ,jj  ,jk,Kmm) &
      &              - e1v(ji  ,jj-1) * e3v(ji  ,jj-1,jk,Kmm) * vv(ji  ,jj-1,jk,Kmm))&
      &            * r1_e1e2t(ji,jj) / e3t(ji,jj,jk,Kmm)
END_3D
```

Figure 3 - Original expression for hdiv

```
DO_3D( nn_hls-1, nn_hls, nn_hls-1, nn_hls, 1, jpkm1 )     !== Horizontal divergence ==!
   hdiv(ji,jj,jk) = (- e2u(ji-1,jj  ) * e3u(ji-1,jj  ,jk,Kmm) * uu(ji-1,jj  ,jk,Kmm) &
      &              + e2u(ji  ,jj  ) * e3u(ji  ,jj  ,jk,Kmm) * uu(ji  ,jj  ,jk,Kmm) &
      &              - e1v(ji  ,jj-1) * e3v(ji  ,jj-1,jk,Kmm) * vv(ji  ,jj-1,jk,Kmm) &
      &              + e1v(ji  ,jj  ) * e3v(ji  ,jj  ,jk,Kmm) * vv(ji  ,jj  ,jk,Kmm))&
      &             * r1_e1e2t(ji,jj) / e3t(ji,jj,jk,Kmm)
END_3D
```

Figure 4 - Equivalent expression for hdiv on the first halo row after altering indices to reflect the north folding

For the case of *hdiv* the difference is only due to the order of floating-point operations, hence the loss of bit comparison between 1- and 2-point halo cases is acceptable. Such a situation should occur for all of the expressions in the code, hence the north folding algorithm should not be changed when a 2-point halo is used. Moreover, bit comparison is still guaranteed between 1- and 2-point haloes by appropriately using round brackets in the expression to force a consistent order of floating-point operations.
It has been verified that by forcing the *hdiv* formula as shown in Figure 5 in both the original 1-point halo code and modified 2-point halo code, the .stat files do not differ.

```
DO_3D( nn_hls-1, nn_hls, nn_hls-1, nn_hls, 1, jpkm1 )     !== Horizontal divergence ==!
hdiv(ji,jj,jk) = ( (  e2u(ji  ,jj) * e3u(ji  ,jj,jk,Kmm) * uu(ji  ,jj,jk,Kmm)   &
   &               - e2u(ji-1,jj) * e3u(ji-1,jj,jk,Kmm) * uu(ji-1,jj,jk,Kmm) )  &
   &               + ( e1v(ji,jj  ) * e3v(ji,jj  ,jk,Kmm) * vv(ji,jj  ,jk,Kmm)  &
   &               - e1v(ji,jj-1) * e3v(ji,jj-1,jk,Kmm) * vv(ji,jj-1,jk,Kmm) ) )&
   &            * r1_e1e2t(ji,jj) / e3t(ji,jj,jk,Kmm)
END_3D
```

Figure 5 - Modified hdiv expression that preserves bit comparison between 1- and 2-point haloes

Since differences in the results due to a different order of the floating point operations can be considered acceptable, but at the same time a code modifications check was needed to ensure that bugs were not introduced, a new branch named ticket2607_r14608_halo1_halo2_compatibility has been created starting from the current trunk in order to:

1. properly insert round brackets for those expressions that will be affected by the movement of the *lbc_lnk* communications. (The use of round brackets will produce an acceptable bit difference in the results without any other modification in the code)
2. proceed with the *lbc_lnk* clean-up and movement, ensuring that the 1- and 2-point halo cases now produce the same results.

3.2 Extra Halo extension (ticket #2366)

The whole code has been changed in order to support an extended halo. It allows the developers to reduce the frequency of communications by moving the communication outside of DO loops (where possible) and to develop other optimizations such as tiling or loop fusion. As stated before, moving the communications outside of DO loops requires a deep analysis of the algorithm implemented in each routine and, to date, has been performed on the Tracer Advection module, the Vertical Ocean Physics module and most of the Ocean Dynamics module routines. The implementation of other optimizations, which are now feasible thanks to the extended halo, requires other developments (now in progress) before producing valuable benefits (i.e. tiling, loop fusion, etc..).

3.3 Improvements using MPI3 (ticket #2496)

A new halo update strategy, implemented through the MPI3 neighborhood collective communications, has been integrated in the LBC (Lateral Boundary Condition) NEMO library, so that both point-to-point and collective communications are supported for the halo exchange. Only the following few code files have been modified/added so that the implementation is not too invasive and does not require a code refactoring:

- *lbc_lnk_call_generic.h90*, where the selection of the communication strategy is implemented through the *nn_comm* namelist parameter, as shown in Figure 6.

```
IF( nn_comm == 1 ) THEN
   CALL lbc_lnk_pt2pt(   cdname, ptab_ptr, cdna_ptr, psgn_ptr, kfld, kfillmode, pfillval, khls, lsend, lrecv, ld4only )
ELSE
   CALL lbc_lnk_neicoll( cdname, ptab_ptr, cdna_ptr, psgn_ptr, kfld, kfillmode, pfillval, khls, lsend, lrecv, ld4only )
ENDIF
```

Figure 6 - Selection of point-to-point or collective communication strategy

The default communication strategy in the NEMO code is now based on the neighborhood collectives approach. However, the user can activate point-to-point exchange if the MPI version on the target system does not support neighborhood collectives, simply by setting the namelist *nn_comm* equal to 1.

- *lbclnk.F90*, where the interfaces for the new communications are introduced

- *lbc_lnk_neicoll_generic.h90*, where the collective neighborhood exchanges are implemented. The graph topology is preferred over cartesian topology (both described in the Milestone M8.4) in order to maintain the same strategy for both 5-point and 9-point stencils. Moreover, land-point domains exclusion is handled due to the flexibility of graph topology. Two different communicators (Figure 7) are created in the *mpp_ini_nc* routine (*lib_mpp.F90*) to support exchanges with and without the 4 corners.

```
CALL MPI_Dist_graph_create_adjacent( mpi_comm_oce, iScnt4, iSnei4, MPI_UNWEIGHTED, iRcnt4, iRnei4, MPI_UNWEIGHTED,   &
   &                                 MPI_INFO_NULL, ireord, mpi_nc_com4(khls), ierr )
CALL MPI_Dist_graph_create_adjacent( mpi_comm_oce, iScnt8, iSnei8, MPI_UNWEIGHTED, iRcnt8, iRnei8, MPI_UNWEIGHTED,   &
   &                                 MPI_INFO_NULL, ireord, mpi_nc_com8(khls), ierr)
```

Figure 7 - MPI communicators used to use 5-point and 9-point stencils

A unique collective (Figure 8) is used instead of 4 point-to-point communications to exchange halo points with neighbors. Send and receive buffers are filled taking into account which processes are exchanging data.

```
! ------------------------------------------------ !
!     3. Do all MPI exchanges in 1 unique call     !
! ------------------------------------------------ !
!
IF( ln_timing ) CALL tic_tac(.TRUE.)
CALL mpi_neighbor_alltoallv (BUFFSND, iScnt, iSdpl, MPI_TYPE, BUFFRCV, iRcnt, iRdpl, MPI_TYPE, impi_nc, ierr)
IF( ln_timing ) CALL tic_tac(.FALSE.)
!
```

Figure 8 - MPI3 collective communication used to exchange halo data

A specific treatment is required when the parallel processes layout has just two rows/columns. In those cases, an MPI bug causes the order of exchanged data to be

reversed, so that the order of destination processes also has to be reversed, as shown in Figure 9.

```
! MPI3 bug fix when domain decomposition has 2 columns/rows
IF (jpni .eq. 2) THEN
   IF (jpnj .eq. 2) THEN
      jnf(1:8) = (/ 2, 1, 4, 3, 8, 7, 6, 5 /)
   ELSE
      jnf(1:8) = (/ 2, 1, 3, 4, 6, 5, 8, 7 /)
   ENDIF
ELSE
   IF (jpnj .eq. 2) THEN
      jnf(1:8) = (/ 1, 2, 4, 3, 7, 8, 5, 6 /)
   ELSE
      jnf(1:8) = (/ 1, 2, 3, 4, 5, 6, 7, 8 /)
   ENDIF
ENDIF
```

Figure 9 - Reordering of the destination processes to fix the MPI3 bug

The choice between 5-point and 9-point stencils requires a deep analysis of the code in order to understand whether the update of each domain point depends on the corner points. Where data dependencies allow developers to use a 5-point stencil, the communication call includes as its last parameter *ld4only* which means only east-west-north-south exchanges are needed. In this case, *mpi_nc_com4* is used to perform the halo update, as shown in Figure 10.

```
ll4only = .FALSE.     ! default definition
IF( PRESENT(ld4only) )   ll4only = ld4only
!
impi_nc = mpi_nc_com8(ihls)    ! default
IF( ll4only )   impi_nc = mpi_nc_com4(ihls)
```

Figure 10 - Choice of the communicator to exchange data with the right neighbors

The Tracer Advection module has been fully analyzed in order to identify which halo updates are satisfied by a 5-point stencil. These communications have been changed by adding the *ld4only* parameter, as shown in Figure 11.

```
END_3D
IF (nn_hls==1) CALL lbc_lnk( 'traadv_qck', zfc(:,:,:), 'T', 1.0_wp , zfd(:,:,:), 'T', 1.0_wp, ld4only= .TRUE. )   ! Lateral boundary conditions
```

Figure 11 - Call to lbc_lnk performing the halo exchange when the 5-point stencil is enough

Accuracy tests included in the NEMO SETTE package have been successfully executed on different HPC systems to verify the output reproducibility.

3.4 Mixed precision preparatory phase

3.4.1 AutoRPE refactorization

There has been a complete refactorization of the AutoRPE code aimed at improving its portability: actual runs on platforms other than MareNostrum4, the Barcelona Supercomputing Center HPC, are planned in the next months. With this goal in mind, a great effort has been made to improve the code readability and workflow. The run time of the analysis phase has been reduced thanks to a study of the interdependencies among variables.

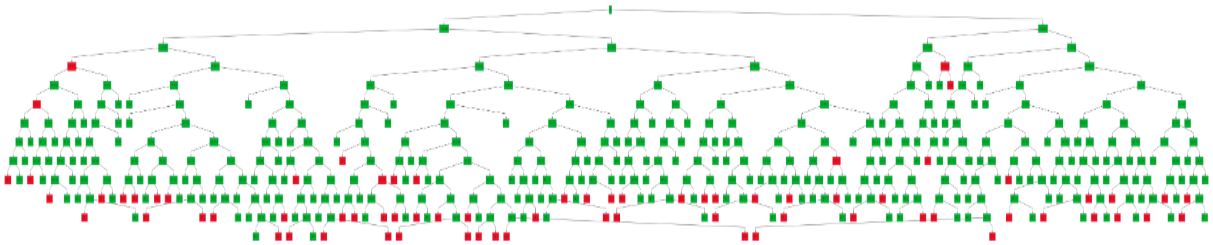

Figure 12 - The binary search tree produced when running a precision analysis on ~1900 variables. Each square represents a run: in green are depicted those runs that produced an output differing from the double precision results by less than a set threshold, in red the ones that didn't.

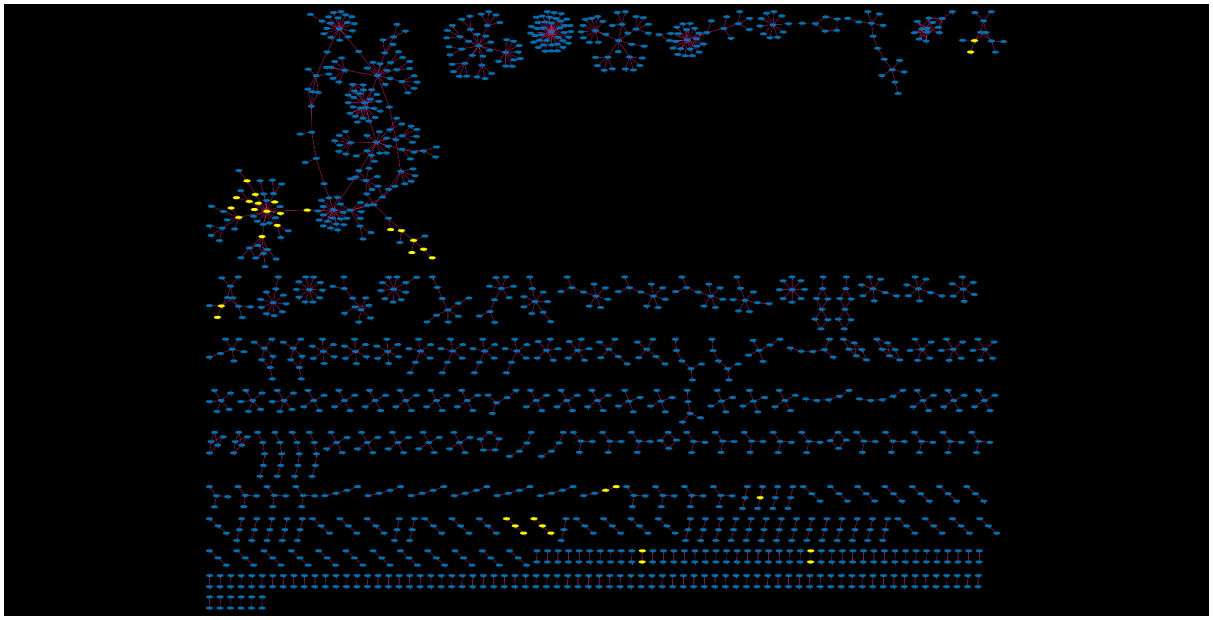

Figure 13 - Graph of all the variables taken into account in one of the analyses performed in the last year at BSC. The yellow variables are those that must retain double precision.

Initially, all the variables considered in the analysis were independently studied. This means that if, for example, a thousand variables were considered, the tree of the binary search algorithm could hypothetically have up to a thousand leaves. As seen in Figure 12, this can require a huge number of simulations to be run. In order to reduce this number, we started analyzing the relation between actual arguments and dummy arguments.

Consider the following:

```
real(dp) :: actual_arg1
real(dp) :: actual_arg2
call my_sbr(actual_arg1, actual_arg2)

subroutine my_sbr(dummy_arg_1, dummy_arg_2)
    real(dp) , intent(in) :: dummy_arg_1
    real(dp) , intent(out) :: dummy_arg_2
    [...]
end subroutine
```

In this case, we can say that the type of *actual_arg2* is bound to the type of *dummy_var_2*. This definition is somewhat too strict, as it does not take into account the existence of interfaces. When a subroutine or a function has an interface for double/single precision reals, the actual argument is bound to two different dummy arguments. The dummy argument that will be used depends on the value of the working precision. Using this definition of "bound variables" in Figure 13, we analyzed all the variables that were taken into account when producing a mixed precision version of NEMO v4.2, and showed in a graph their dependencies (Figure 13). We can see several clusters of different dimensions with the yellow dots representing the results of the analysis, i.e. those variables that must retain double precision (dp). Since once a variable in a cluster is dp then all other variables in the cluster must also be dp, there is no need to study all of the variables independently.

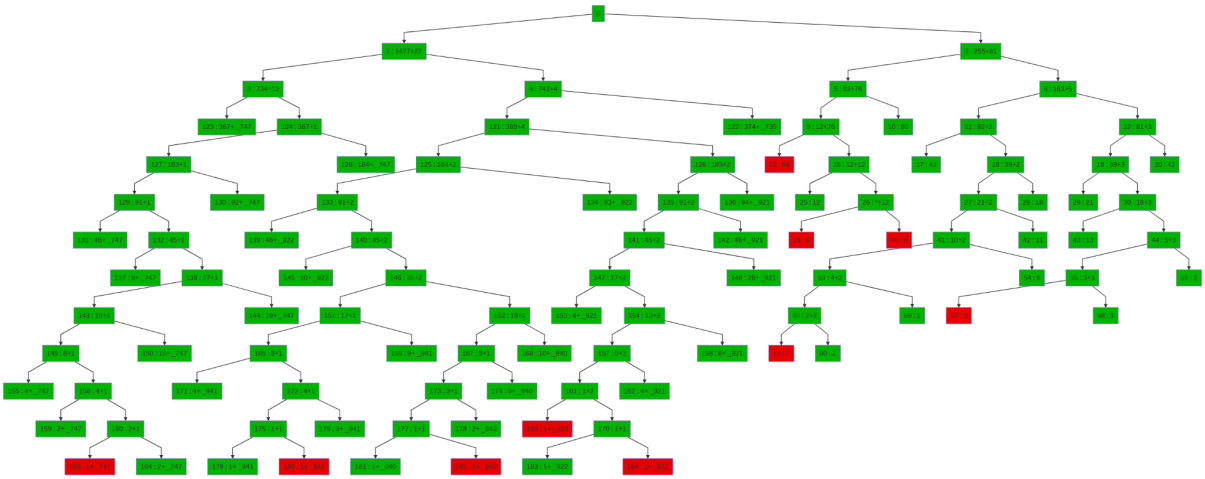

Figure 14 - Once the concept of clustering is considered, the number of runs is visibly reduced. Clustering implies that certain groups are not further subdivided when the run fails, but also suggests a different way of dividing variables in the analysis.

We thus introduced the concept of clustering in the analysis, meaning that certain runs will contain only those variables belonging to a cluster. Once a particular run fails, it will no longer be subdivided as with the other runs (the squares in Figure 14). In this way we are able to reduce the number of total runs in the binary search by about 10% (Figure 14). We have been able to further reduce the time of each run by speeding up the test to decide if a certain configuration is good or not, just by optimizing the size of the file we use for the comparison. Before, we were reusing results from a 30-year spinup run, while now we are producing custom small files that are very fast to load into memory.

3.4.2 Support for the new git repository

The workflow has also been adapted to run with the official NEMO Git repository, which replaced the SVN repository used up to December 2021. We created a mixed precision branch, called "dev_mixed_precision" where we proceeded to fix all lines of code that deviated from the official NEMO coding guidelines. This resulted in minor changes to around 30 files, such as adding intent to dummy arguments or adding the name of the subroutine after the statement END SUBROUTINE. We also added the definition of a new precision, the quadruple precision, since it is now needed in debugging control routines. In line with these changes, we also produced a document called "NEMO developer's guidelines" where we explain these and other

rules in detail. Adhering to these rules would not only ease the pressure on the parsing phase of the AutoRPE tool, but also improve the readability and maintainability of the NEMO code itself.

3.4.3 Testing

Since we worked hard on the usability of the tool, we have now agreed with CMCC and IPSL to hold a training session on the AutoRPE tool. At the end of the training, we expect to have the tool tested on platforms available at CMCC. This will provide feedback on AutoRPE needed to release it publicly, providing the users with a reasonably free-of-bugs and portable version of the tool. We also expect to learn from this experience how to make the AutoRPE tool easier to deploy for inexperienced users.

4 Perspectives and conclusions

Changes implemented in the NEMO code due to IS-ENES3 activities need further improvements during the next period.

With regard to the extended halo management, the increase in halo size reduces the exchange frequency and is a prerequisite for other code changes, but has the drawback of increasing the computational domain size. Valuable benefits in terms of performance can be achieved when other developments, such as tiling and loop fusion (already in progress), are completed.

As preliminary tests reported in M8.4 show, the integration of MPI3 neighboring collective communications improves computational performance when a 5-point stencil is sufficient for halo exchanges. A full analysis of the code is therefore needed to identify whether further exchanges with corners can be avoided. To date, this analysis has been performed on the Tracer advection module, which is the most expensive one.

Finally, future tests of the AutoRPE tool on the CMCC platform will provide additional information on its portability, so that it can be officially released as a NEMO utility.

In conclusion, we can say that the work funded by the IS-ENES3 project has resulted in important changes to the NEMO code, but has also laid the foundations for future developments that would benefit from the work done.