North Fold Algorithm and Halo 2

The use of halo > 1 should allow to remove the communications between DO loops eventually shifting the exchange at the beginning/end of a routine.
For example, in the original *div_hor* routine, the horizontal divergence *hdiv* is computed on the inner domain. Then, its halo is updated through an *lbc_lnk* using a ‘T’ type for the north folding.

```
      !
      DO_3D( 0, 0, 0, 0, 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
      !
      IF( ln_rnf )   CALL sbc_rnf_div( hdiv, Kmm )                     !==  runoffs    ==!   (update hdiv field)
      !
#if defined key_asminc
      IF( ln_sshinc .AND. ln_asmiau )   CALL ssh_asm_div( kt, Kbb, Kmm, hdiv )   !==  SSH assimilation  ==!   (update hdiv field)
      !
#endif
      !
      IF( ln_isf )                      CALL isf_hdiv( kt, Kmm, hdiv )           !==  ice shelf         ==!   (update hdiv field)
      !
      CALL lbc_lnk( 'divhor', hdiv, 'T', 1.0_wp )   !   (no sign change)
      !
```

If halo = 2, *uu, vv* (velocities) and *r3u, r3v, r3t* (ratio *ssh/h_0*) can be exchanged through a lbc_lnk communication, before the first DO_3D, so that updated halo values can be used for the computation of horizontal divergence *hdiv* in the range [2:jpi,2:jpj,1:jpk-1].

```
      !
      IF (nn_hls.eq.2) CALL lbc_lnk( 'divhor', uu(:,:,:,Kmm), 'U', -1.0_wp, vv(:,:,:,Kmm), 'V', -1.0_wp )
      IF (nn_hls.eq.2) CALL lbc_lnk( 'divhor', r3u(:,:,Kmm), 'U', 1.0_wp, r3v(:,:,Kmm), 'V', 1.0_wp, r3t(:,:,Kmm), 'T', 1.0_wp )

      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
      !
      IF( ln_rnf )   CALL sbc_rnf_div( hdiv, Kmm )                     !==  runoffs    ==!   (update hdiv field)
      !
#if defined key_asminc
      IF( ln_sshinc .AND. ln_asmiau )   CALL ssh_asm_div( kt, Kbb, Kmm, hdiv )   !==  SSH assimilation  ==!   (update hdiv field)
      !
#endif
      !
      IF( ln_isf )                      CALL isf_hdiv( kt, Kmm, hdiv )           !==  ice shelf         ==!   (update hdiv field)
      !
      IF (nn_hls.eq.1) CALL lbc_lnk( 'divhor', hdiv, 'T', 1.0_wp )   !   (no sign change)
      !
```

These changes do not alter run.stat and tracer.stat files in GYRE PISCES configuration, but ORCA2_ICE_PISCES reference configuration shows different results since time step 23 (REPRO84 test) comparing with halo 1 case although restartability and reproducibility are preserved.

This leads to think it is something related to NORTH FOLD treatment. Indeed (we’ve got discussion some months ago with Seb and Gurvan about this) if we give a look at the contribution in horizontal divergence *hdiv* computation it can be noted that u-grid and v-grid related values are placed in a symmetric/mirrored way compared to the original case.

It is worth recalling that the differences between halo 1 and halo 2 are only for the computation of the values held in the first halo line, indeed in the halo 1 case those values are received from the neighbor (or from the corresponding rank in north fold); in halo 2 case, those values are computed locally.

Considering an 8x4 domain decomposition with jpi=25 and jpj=39, in the original halo 1 case the *hdiv* value for the halo point (21,39) on proc 29 is sent by proc 26 after computing its (3,37) inner point.

On the other side, when halo 2 is used, the corresponding halo point (22,40) is directly computed by proc 29.
So, what we expect is that
$hdiv_{[22,40]}\text{on proc 29 (halo 2 case)} = hdiv_{[3,37]}\text{on proc 26 (halo 1 case)}$
Unfortunately, in the current implementation, these values are different.

Looking at the different values contributing to the horizontal divergence *hdiv* computation, we can note that they are placed in a symmetric/mirrored way compared to the original case. Table 1 reports the corresponding values along with the halo region:

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

More in general, we can hence assert that when we compute the values in the first line of halo (this happens only in the halo 2 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

when we apply these changes to the hdiv expression, we have that:

```
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
```

(this is the original expression which will be actually evaluated for the inner points)

```
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
```

(this is how the original expression is actually turned for the values in the first line of halo due to the north fold)

For the case of hdiv we found that the difference is only due to the order of floating-point operations hence the loss of bit comparison between halo 1 and halo 2 is acceptable. We are confident that such a situation should happen for all of the expressions in the code, hence the north folding algorithm should not be changed when halo is 2. Moreover, we can still guarantee the bit comparison between halo 1 and halo 2 by forcing the same order of floating-point operations by properly using round brackets in the expression.

We verified that forcing the hdiv formula in the following way in both original halo 1 code and modified halo 2 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
```

Since differences in the results due to a different order of the floating-point operations can be considered acceptable, but at the same time we need to check our code modifications to be sure they don't introduce bugs in the code, what we propose is:

1. to properly insert in the current trunk the round brackets for those expressions that will be touched by the movement of the lbc_lnk. (The use of round brackets will produce an acceptable bit difference in the results without any other modification in the code)
2. to proceed with the lbc_lnk clean-up and movement being sure that halo 1 and halo 2 must now produce the same results.