Subroutine Arguments

(for tra_xxx and dyn_xxx, i.e RHS of momentum and tracer Eqs.)

(0) Preliminary remarks

• I'm deeply against changing the DOCTOR rule with respect to naming convention of subroutine arguments. These rules can evolve to better discriminate an unthought type of variable. This was the case for namlist parameters (starting with nn_, rn_, or ln_), but fondamental properties such as routine argument can't be overcome. Furthermore, if we change this logic, we will have to do it every where in all components of NEMO not only in TRA/DYN subroutines….

• I don't like the idea of having the state variables encapsulated in a structure and I prefer to have a fourth (fifth) dimension (associated with the time) added time-varying variables.

(1) Having a long list of argument is counterproductive and source of errors (especially with arguments that have the same characteristics (typically 3D array of same size). Therefore I don't think it is a good idea to suppress the use of "USE" instruction to have access to variable used in the computation of TRA and DYN subroutines.

(2) The real need for readability is to pass in argument: *(i)* the variable that are computed or updated in the subroutine, i.e. that have an INTENT out or inout : typically the RHS of TRA and DYN Eqs. ; *(ii)* for input variables, the required knowledge is the time used (e.g. before now after for leapfrog), so just these index can be put in argument and (iii) potentially we need information associated to the chosen tiling or the Open-MP strategy. The latest point is discussed in the next slide.

The 2 Tiling/Open-MP strategies (1/2)

(1) micro-tiling/Open-MP

It consists in distributing the tailing at low level, i.e. inside each TRA and DYN subroutines. This is possible for tiling as it is sequential: we can have, inside a subroutine, a succession of computation on horizontal and vertical tiles.

Pros:
- …. not that much !

Cons:
- each changes in tile means more memory loading
- inefficient with Open-MP (synchronization required when switching from H/V tiles).
- choosing the number of H-tiles and V-tiles is highly problematic…

(2) macro-tiling/Open-MP

It consists in using the same pencil domain decomposition as for MPI domain. In practice, it means that the loop over tiles/Open-MP sub-domain is done at step level.

Pros:
- same implementation for tiling, Open-MP, and MPI (relatively easy to do)
- minimize the memory loading (tiles/Open-MP) (faster code)
- minimize the synchronization in Open-MP
- 2 degrees of liberty to optimize the size of tiles / Open-MP domain (like MPI)
- solve the issues of monotonic schemes (i.e. need of the knowledge of 3D fluxes in the 3 directions)

Cons:
- same increase in computation as in an equivalent pure MPI decomposition but with less communications.
- Open-MP: call to lbclnk and iom_put inside TRA/DYN subroutines

NB: that's the chosen approach in ROMS, CROCO, MITgcm…

The 2 Tiling/Open-MP strategies (2/2)

The two issues with Open-MP and pencil domain decomposition:

(1) output inside TRA/DYN subroutines

iom_put calls are probably already solved in xIOS with mixture of MPI / OpenMP.

either in DYNAMICO or in CROCO <<<=== To be verified

If not, since there is no conceptual issue for each pencil domain to cumulate and send to I/O its own domain, it can be solved.

(2) lbclnk inside TRA/DYN subroutines

This is an issue only in Open-MPtwo solutions :

(i) use a halo-size of 2 (and thus remove lbc_lnk calls form these subroutines)

(ii) add as first instruction in lbc_lnk an Open-MP synchronization

Proposed practical implementation (1/3)

Use of CPP in order to defined some subroutine arguments and all 2D/3D loops put into a included in all TRA/DYN routines (and also in ICE, TOP, etc…)

(1) size of pencils in argument

In *domain_substitute.h90* file :

```
#define DOM2D_ARG   kls,kle,kJs,kJe
#define WRK_2D      kls:kle,kJs:kJe
```

```
#include "domain_substitute.h90"
CONTAINS

 SUBROUTINE tra_ldf_iso( kt, DOM2D_ARG, ….)
    !!----------------------------------------------------------------------
    INTEGER, INTENT(in   ) ::   kt                ! ocean time-step index
    INTEGER, INTENT(in   ) ::   DOM2D_ARG   ! first time step index
    …
    REAL(wp), DIMENSION(WRK_2D)           ::    z2d    ! 2D workspace
    REAL(wp), DIMENSION(WRK_2D,1:jpkm1) ::    z3d    ! 3D workspace
    REAL(wp), DIMENSION(WRK_2D,jpk)       ::    z3de  ! 3D workspace
```

Proposed practical implementation (2/3)

(2) 2D/3D loops

In *domain_substitute.h90* file (here halos size = 1 easily adjust for a size of 2)

```
#define DO_2D_00_00   DO jj = kJs  , kJe      ;  DO ji = kIs  , kIe
#define DO_2D_10_10   DO jj = kJs-1, kJe      ;  DO ji = kIs-1, kIe
#define DO_2D_01_01   DO jj = kJs  , kJe+1    ;  DO ji = kIs  , kIe+1
#define DO_2D_11_11   DO jj = kJs-1, kJe+1    ;  DO ji = kIs-1, kIe+1
!
#define DO_3D_00_00(ks,ke)   DO jk = ks, ke   ;  DO_2D_00_00
#define DO_3D_10_10(ks,ke)   DO jk = ks, ke   ;  DO_2D_10_10
#define DO_3D_01_01(ks,ke)   DO jk = ks, ke   ;  DO_2D_01_01
#define DO_3D_11_11(ks,ke)   DO jk = ks, ke   ;  DO_2D_11_11
!
#define END_2D          END DO   ;  END DO
#define END_3D          END DO   ;  END DO   ;  END DO
```

NB: strongly reduce the need of indentation for 3D loop….

Proposed practical implementation (2/3)

```
DO jk = 1, jpkm1      ! Horizontal slab
  !
  DO_2D_11_11
    zdk1t(ji,jj) = ( ptb(ji,jj,jk,jn) - ptb(ji,jj,jk+1,jn) ) * wmask(ji,jj,jk+1)
  END_2D
  DO_2D_10_10
     zftu(ji,jj,jk) = (  zabe1 * zdit(ji,jj,jk)   …
     zftv(ji,jj,jk) = (  zabe2 * zdjt(ji,jj,jk)   …
  END_2D
  !
END DO              ! End of slab

! Vertical fluxes
! ---------------
!                    ! Surface and bottom vertical fluxes set to zero
zftw(WRK_2D, 1 ) = 0._wp      ;      zftw(WRK_2D,jpk) = 0._wp

DO_3D_00_00(2,jpkm1)                  ! interior (2=<jk=<jpk-1)
    zftw(ji,jj,jk) = zcoef3 * (   zdit(ji  ,jj  ,jk-1) + zdit(ji-1,jj  ,jk)   ….
END_3D
  !
```