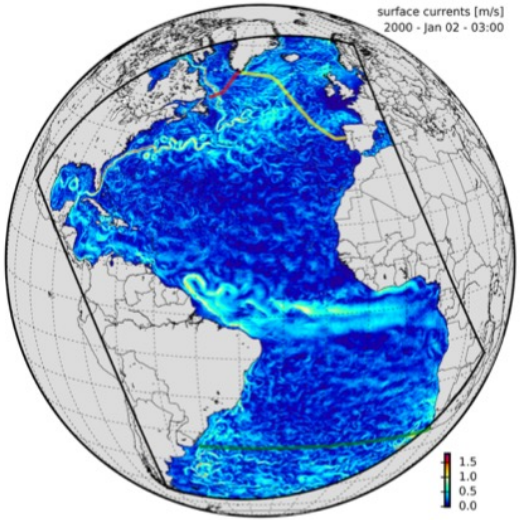
# CROCO – training 2023

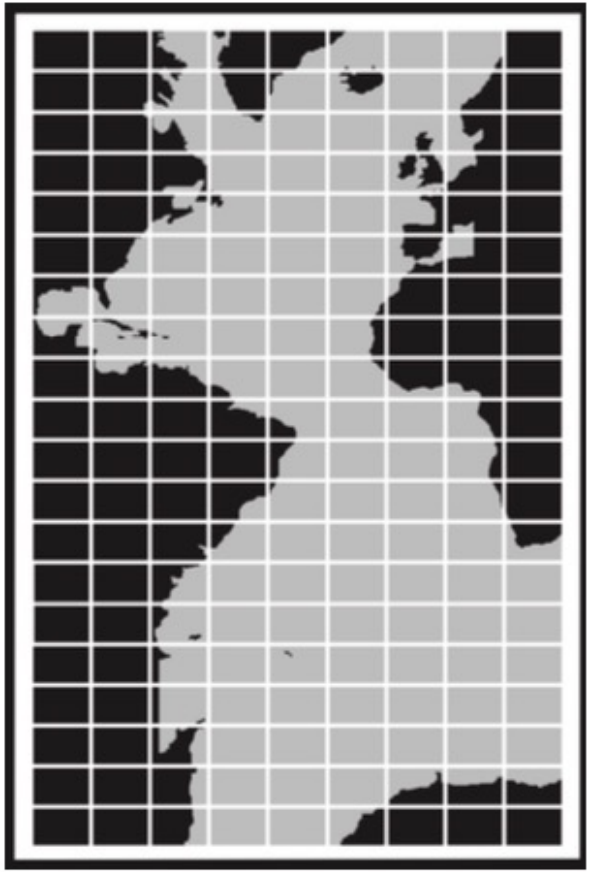
## Parallelization and HPC aspects

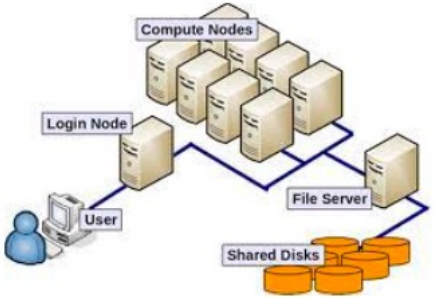
CROCO
Coastal and Regional Ocean COmmunity model



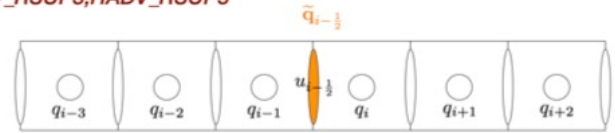

3km resolution

# Domain decomposition

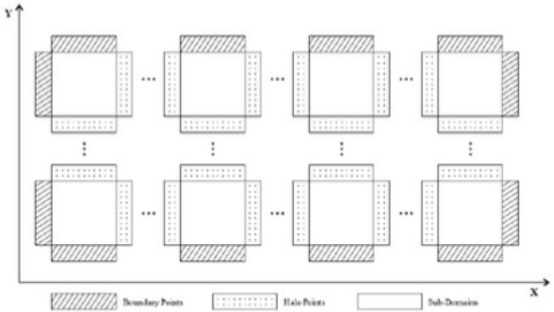




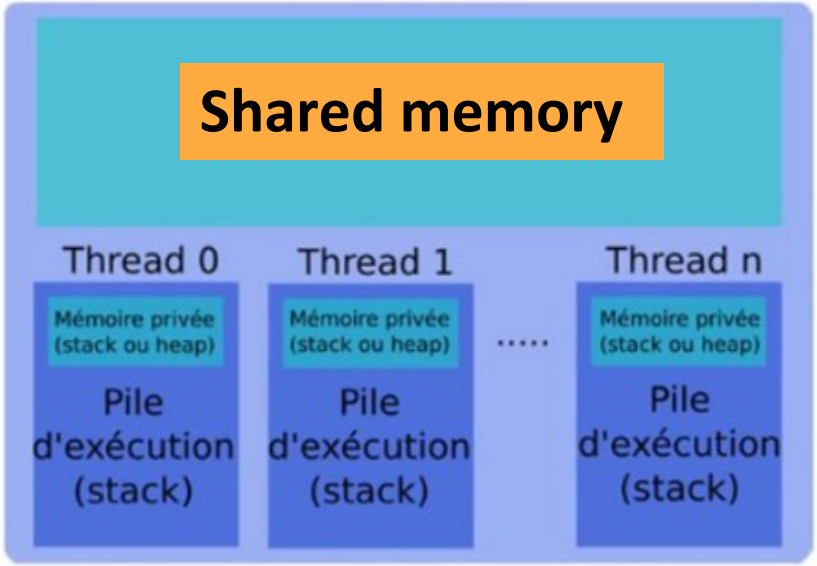

*HADV_RSUP3,HADV_RSUP5*

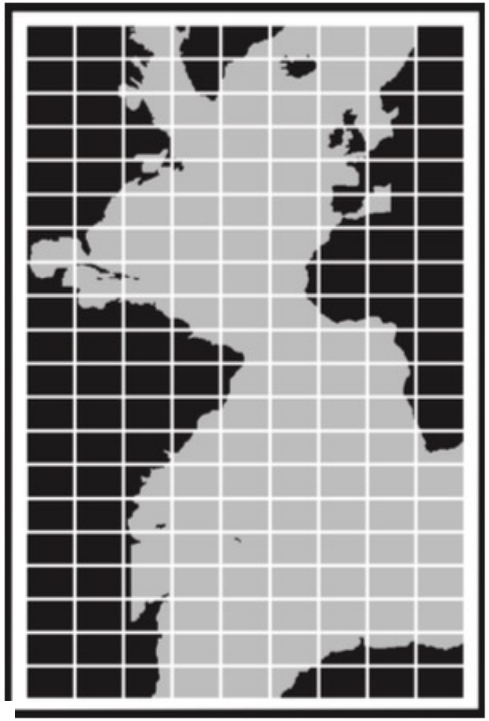



$$\partial_x(uq)|_{x=x_i} = \frac{1}{\Delta x_i}\left\{u_{i+\frac{1}{2}}\widetilde{q}_{i+\frac{1}{2}} - u_{i-\frac{1}{2}}\widetilde{q}_{i-\frac{1}{2}}\right\}$$



# Approach 1 : shared memory

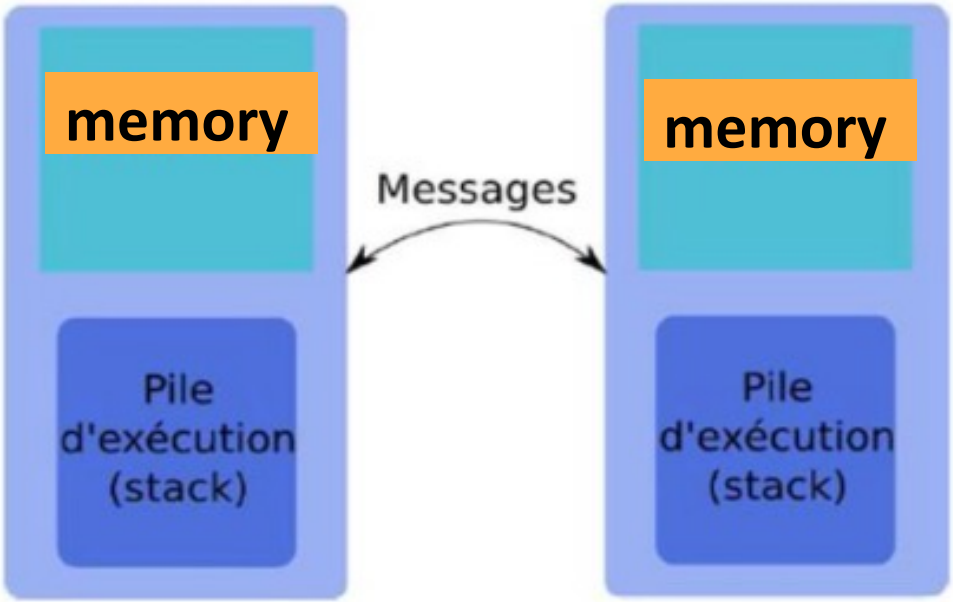


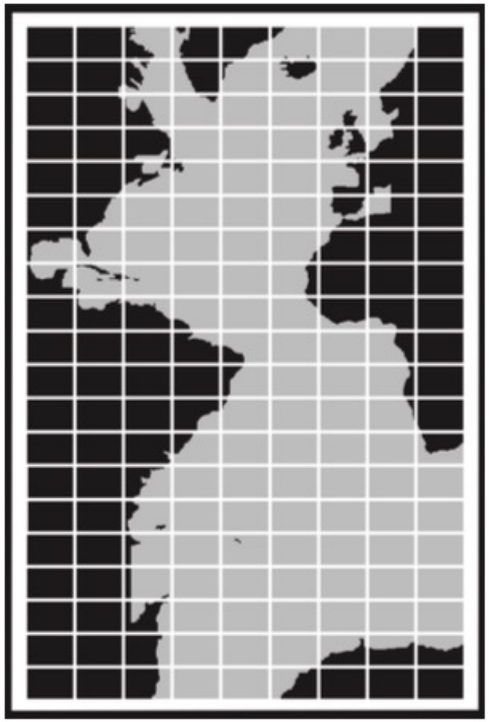



- Computing node have access to a shared memory
- Exchange by memory copy

**Standard OpenMP**
**(Open Multi-Processing)**

# Approach 1 : distributed memory



- Computing nodes DO NOT have access to a shared memory
- Exchange by explicit network message
- Practically, MPI manage also shared memory
- **Preferred approach in CROCO**

**Standard MPI**
**(Message Passing Interface)**

# Implementation in CROCO : OpenMP

2 files to edit in CROCO
param.h : specify the x and y decomposition
cppdefs.h : #define OPENMP
then compilation

Need to specify the number of computing core to the environment
export OMP_NUM_THREADS = 4

```
!
! Domain subdivision parameters
! ====== =========== ==========
!
! NPP            Maximum allowed number of parallel threads;
! NSUB_X,NSUB_E  Number of SHARED memory subdomains in XI- and
!                                                   ETA-directions;
! NNODES        Total number of MPI processes (nodes);
! NP_XI,NP_ETA  Number of MPI subdomains in XI- and ETA-directions;
!
      integer NSUB_X, NSUB_E, NPP
#ifdef MPI
      integer NP_XI, NP_ETA, NNODES
      parameter (NP_XI=1,  NP_ETA=4,  NNODES=NP_XI*NP_ETA)
      parameter (NPP=1)
      parameter (NSUB_X=1, NSUB_E=1)
#elif defined OPENMP
      parameter (NPP=4)
# ifdef AUTOTILING
      common/distrib/NSUB_X, NSUB_E
# else
      parameter (NSUB_X=1, NSUB_E=NPP)
# endif
#else
      parameter (NPP=1)
```

# Implementation in CROCO : OpenMP

CROCO
Coastal and Regional Ocean COmmunity model

- Each core can read/write global variables.
- Has to ensure that different cores will not write the same indices of variables.
- Also has to synchronize between different threads

```
      Do tile=my_first,my_last

          Call compute_1(tile)
          Call compute_2(tile)
      Enddo
C$OMP BARRIER ! synchronisation

      Do tile=my_first,my_last

          Call compute_3(tile)
          Call compute_4(tile)
      Enddo
C$OMP BARRIER ! synchronisation
```

# Implementation in CROCO : MPI

2 files to edit in CROCO

- param.h : specify the x and y decomposition : NP_XI and NP_ETA
- cppdefs.h : #define MPI

then compilation

Execution :

mpirun –np 4 ./croco croco.in (or mpiexec, srun, … depending on the MPI scheduler)

```
!
! Domain subdivision parameters
! ====== =========== ==========
!
! NPP            Maximum allowed number of parallel threads;
! NSUB_X,NSUB_E  Number of SHARED memory subdomains in XI- and
!                                                   ETA-directions;
! NNODES        Total number of MPI processes (nodes);
! NP_XI,NP_ETA  Number of MPI subdomains in XI- and ETA-directions;
!
      integer NSUB_X, NSUB_E, NPP
#ifdef MPI
      integer NP_XI, NP_ETA, NNODES
      parameter (NP_XI=1,  NP_ETA=4,  NNODES=NP_XI*NP_ETA)
      parameter (NPP=1)
      parameter (NSUB_X=1, NSUB_E=1)
#elif defined OPENMP
      parameter (NPP=4)
# ifdef AUTOTILING
      common/distrib/NSUB_X, NSUB_E
# else
      parameter (NSUB_X=1, NSUB_E=NPP)
# endif
#else
      parameter (NPP=1)
```

# Implementation in CROCO : MPI

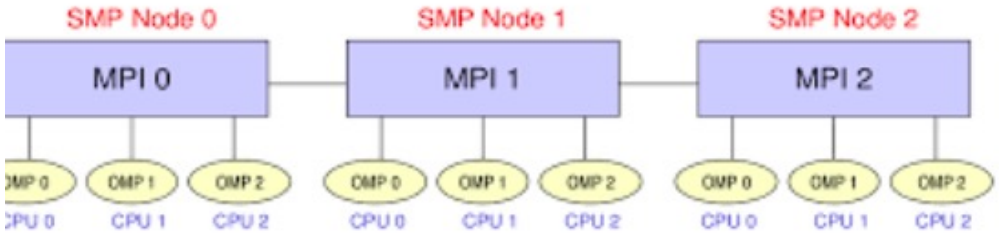
- Each core has access to the variables only over the tile
- Cores have to communicate with each other to exchange information about boundaries

```
# if defined EW_PERIODIC || defined NS_PERIODIC || defined MPI
      call exchange_u3d_tile (Istr,Iend,Jstr,Jend,
     &                                u(START_2D_ARRAY,1,nnew))
      call exchange_v3d_tile (Istr,Iend,Jstr,Jend,
     &                                v(START_2D_ARRAY,1,nnew))

      call exchange_u3d_tile (Istr,Iend,Jstr,Jend,
     &                                  Huon(START_2D_ARRAY,1))
      call exchange_v3d_tile (Istr,Iend,Jstr,Jend,
     &                                  Hvom(START_2D_ARRAY,1))

      call exchange_u2d_tile (Istr,Iend,Jstr,Jend,
     &                                  ubar(START_2D_ARRAY,knew))

      call exchange_v2d_tile (Istr,Iend,Jstr,Jend,
     &                                  vbar(START_2D_ARRAY,knew))
#   if defined TS_MIX_ISO || defined TS_MIX_GEO
      call exchange_u3d_tile (istr,iend,jstr,jend, dRdx  )
      call exchange_v3d_tile (istr,iend,jstr,jend, dRde  )
#   endif
```

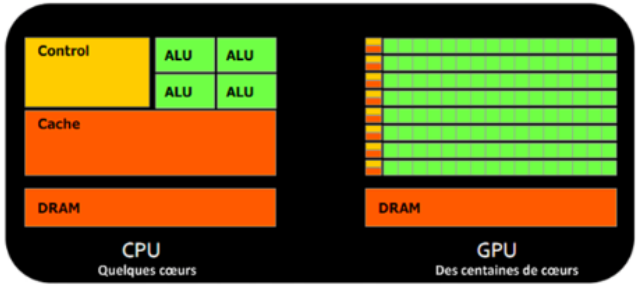
# Take home message

- 2 paradigm available : MPI and OpenMP
- Need to recompile the code
- MPI is prefered because much more used
- Think about Favour the domain decomposition in the ETA (Y) direction for performances

- No hybrid (OpenMP/MPI) version

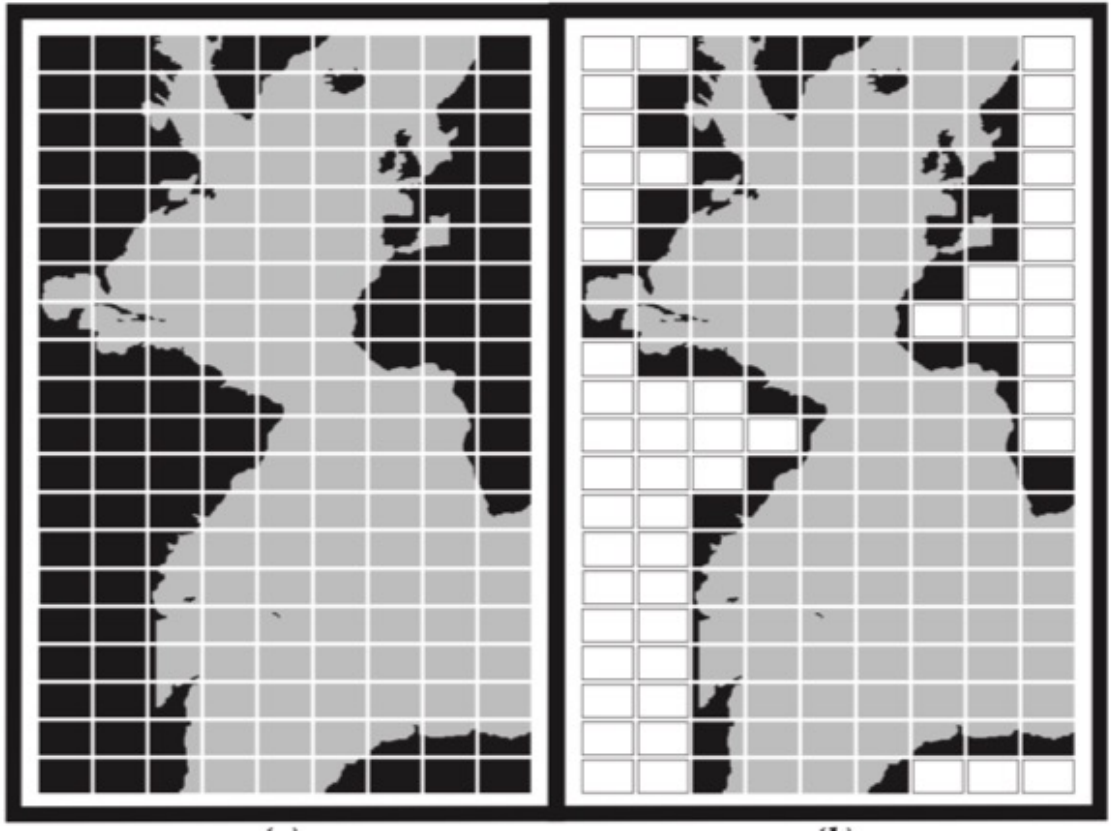



- GPU version in développement



# Land sub-domains suppression

To suppress computing node and speed up the computation



# Land sub-domains suppression

**Need preprocessing + Compilation with MPI_NOLAND**

1. Preprocessing in croco/MPI_NOLAND :

- compile the preprocessing : edit makefile + make
- edit namelist with the name of your grid file and the max CPU you can use
- Run the code : ./mpp_optimize
- Visualize : *./mpp_plot.py croco_grd.nc benguela-008x005_033*

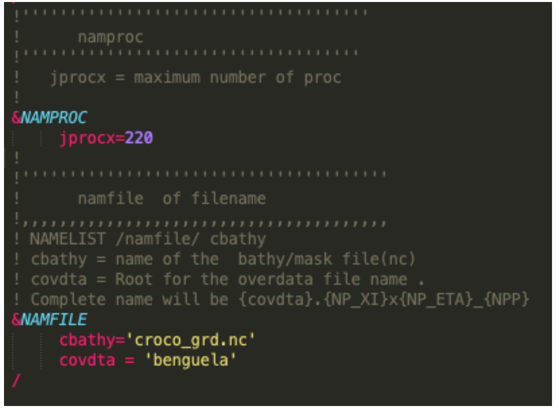
2. Compilation part

- add key MPI_NOLAND
- edit param.h : values for NP_XI, NP_ETA and NNODE from preprocessing

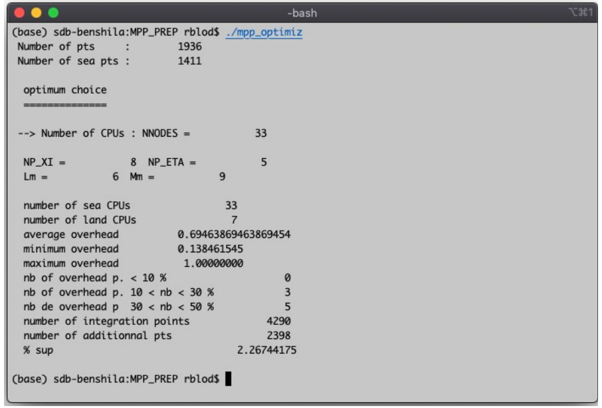
(NNODE <= NP_XI x NP_ETA )

- run as usual

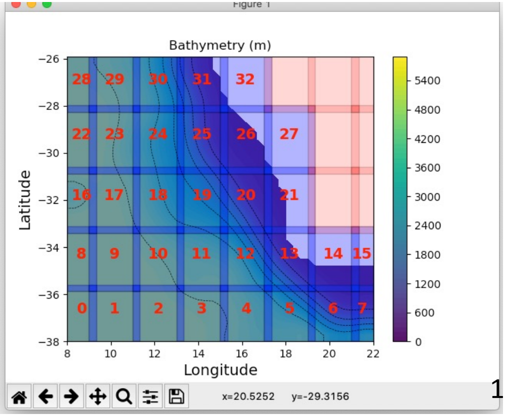
WARNING : grid file as to be called croco_grd.nc (or to be changed in MPI_Setup.F)



```
!''''''''''''''''''''''''''''''''''''''
!       namproc
!''''''''''''''''''''''''''''''''''''''
!   jprocx = maximum number of proc
!
&NAMPROC
        jprocx=220
!
!''''''''''''''''''''''''''''''''''''''
!       namfile  of filename
!,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
! NAMELIST /namfile/ cbathy
! cbathy = name of the  bathy/mask file(nc)
! covdta = Root for the overdata file name .
! Complete name will be {covdta}.{NP_XI}x{NP_ETA}_{NPP}
&NAMFILE
        cbathy='croco_grd.nc'
        covdta = 'benguela'
/
```



```
(base) sdb-benshila:MPP_PREP rblod$ ./mpp_optimiz
 Number of pts      :        1936
 Number of sea pts :        1411

  optimum choice
  ==============

 --> Number of CPUs : NNODES =            33

  NP_XI =           8  NP_ETA =           5
  Lm =           6  Mm =           9

  number of sea CPUs                33
  number of land CPUs                7
  average overhead         0.69463869463869454
  minimum overhead         0.138461545
  maximum overhead          1.00000000
  nb of overhead p. < 10 %                    0
  nb of overhead p. 10 < nb < 30 %            3
  nb de overhead p  30 < nb < 50 %            5
  number of integration points             4290
  number of additionnal pts                2398
  % sup                               2.26744175

(base) sdb-benshila:MPP_PREP rblod$ 
```



# Outputs with MPI (cpp key MPI)

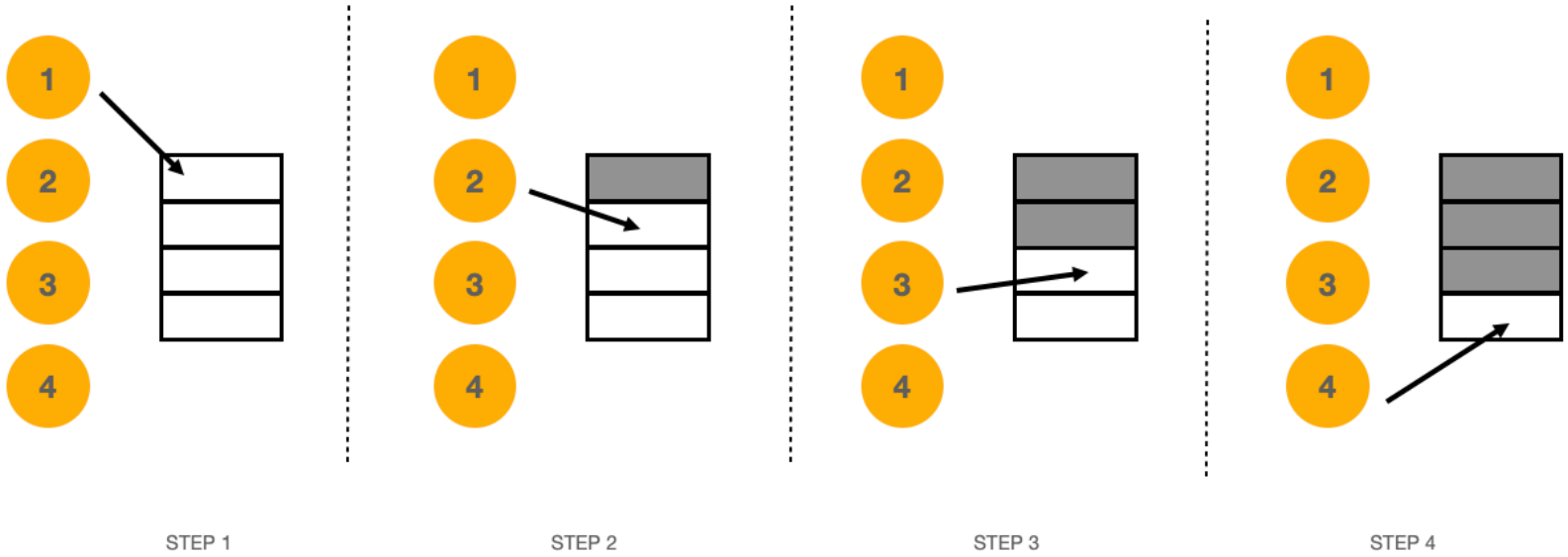
CROCO
Coastal and Regional Ocean COmmunity model

**4 choices :**

- do nothing
- use parallel files option : key PARALLEL_FILES
- use NETCDF4 parallel capabilities : key NC4PAR
- use XIOS (next section)

# Outputs with MPI : nothing specified

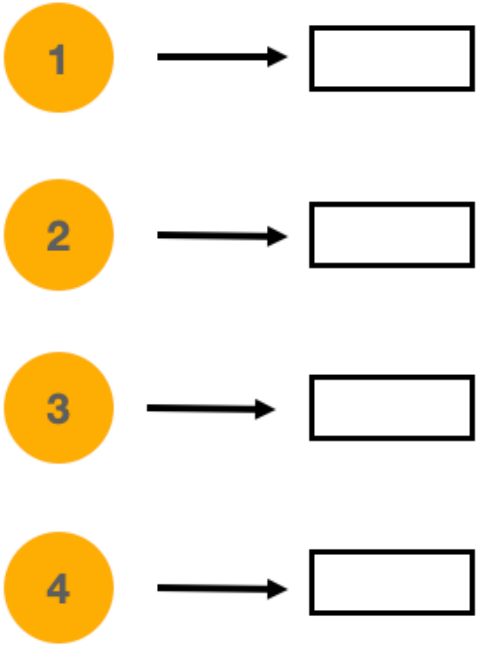
mpirun -np 4 ./croco.  (NP_ETA=4)




Very unefficient !!

# Outputs with MPI : KEY PARALLEL_FILES

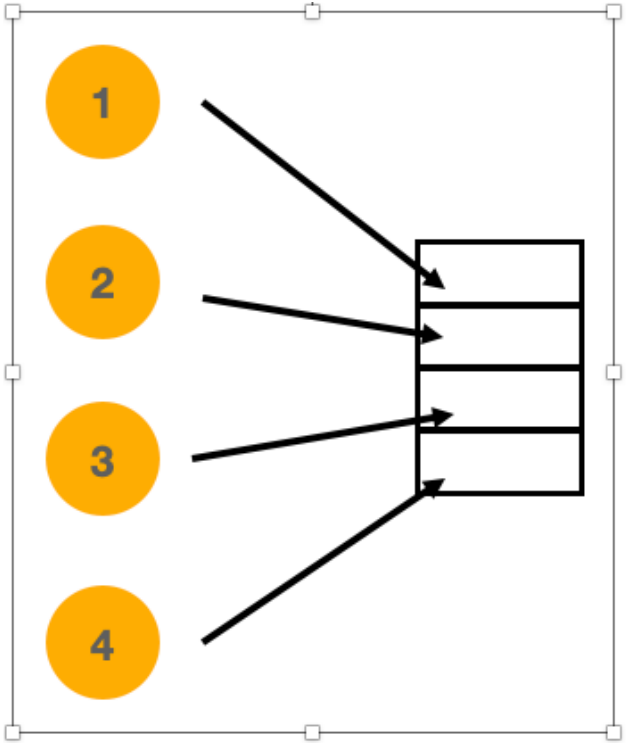
mpirun -np 4 ./croco.  (NP_ETA=4)




Speed-up writing
but end up with split files to recombine (ncjoin)

# Outputs with MPI : key NC4PAR

mpirun -np 4 ./croco.  (NP_ETA=4)



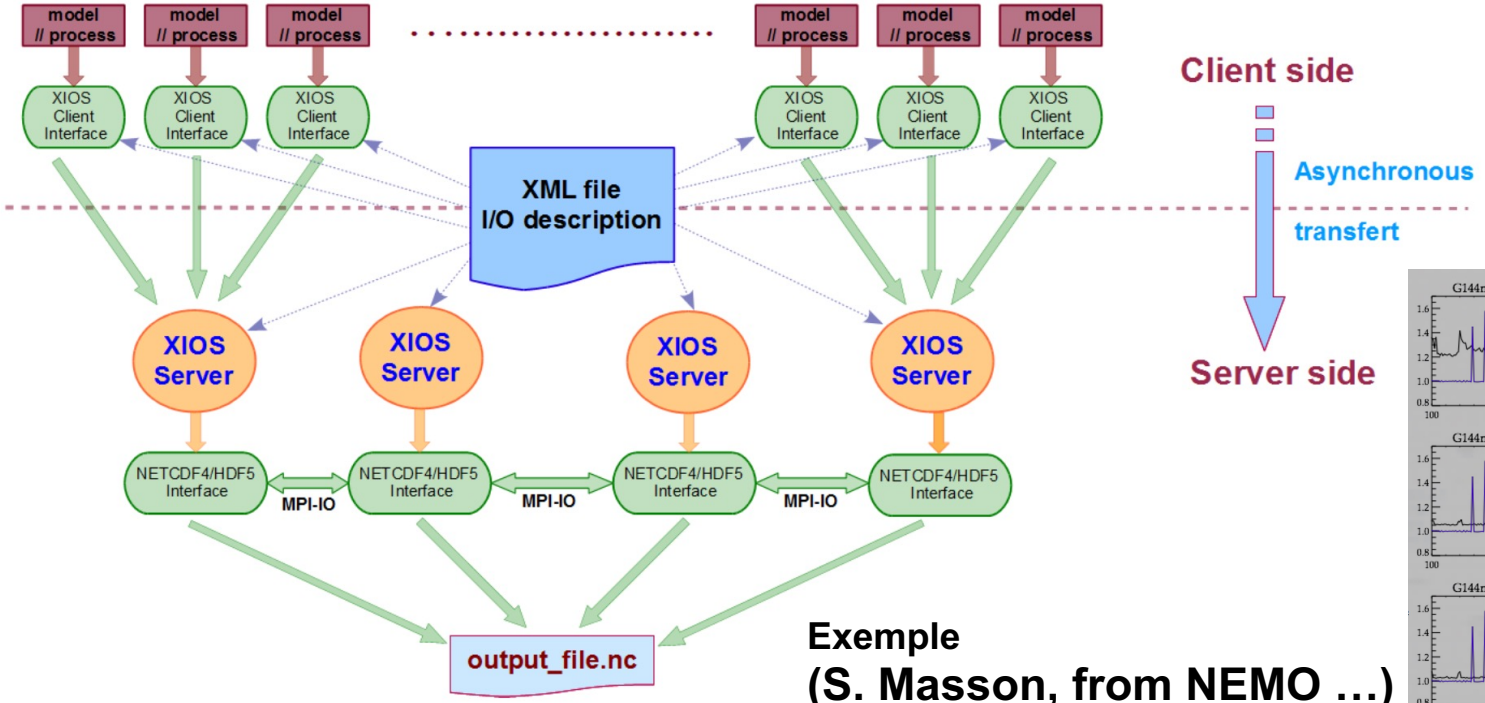

Speed-up with only one file at the end
Need NetCDF4 build with parallel capabilities

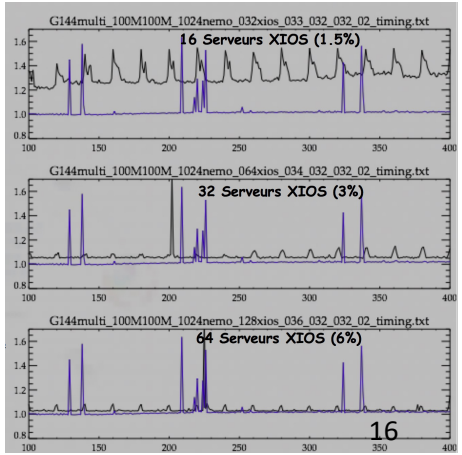
# XIOS

**Strategy for outputs**

**XIOS : external server developed at IPSL**

**http://forge.ipsl.jussieu.fr/ioserver**




**Exemple**

**(S. Masson, from NEMO …)**

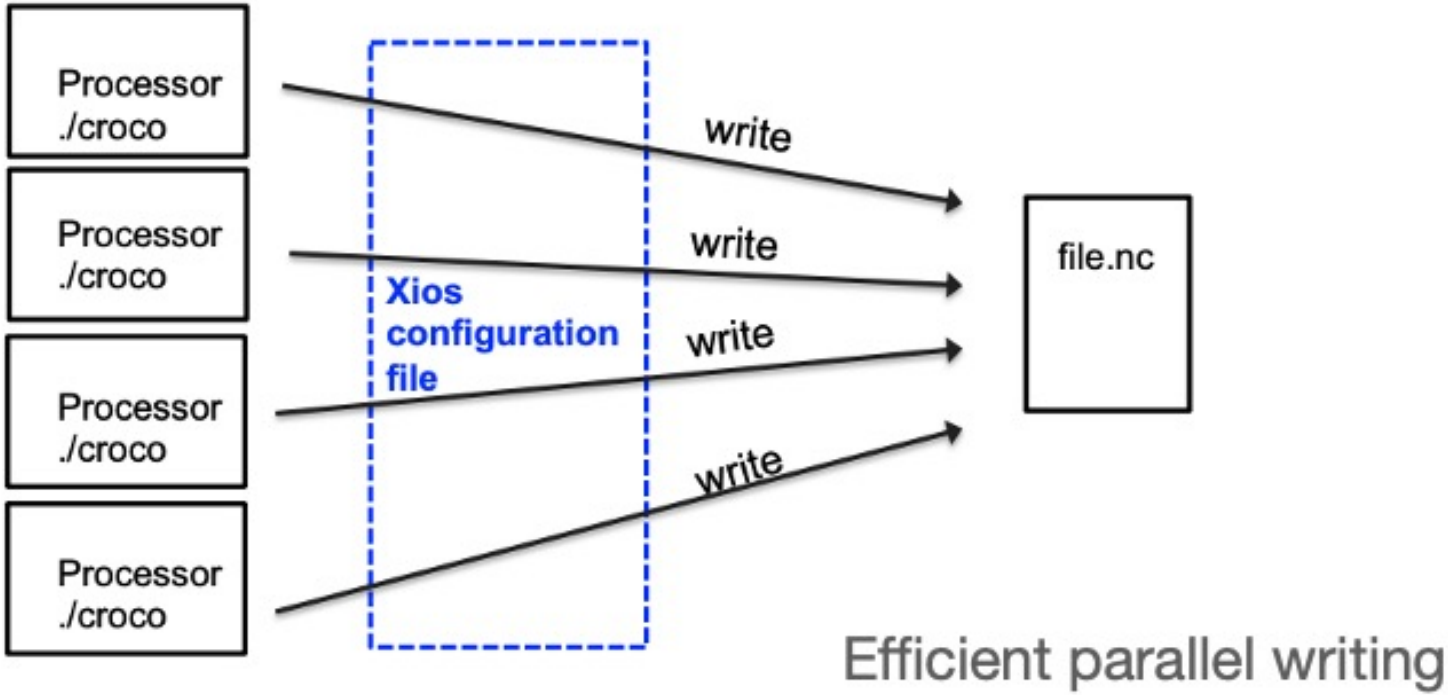
# XIOS

CROCO
Coastal and Regional Ocean COmmunity model

- Originally, a library dedicated to Input/Output management of large climate coupled models (e.g. CMIP simulations for IPCC with NEMO and other code)
- Written and managed at (LSCE-IPSL) by Y. Meurdesoif et al.
- XIOS creates output NetCDF files
- Implemented in other codes (ROMS, MARS3D, CROCO) by non-xios-expert developers despite of a light existing documentation.
- All documentation at http://forge.ipsl.jussieu.fr/ioserver with tutorials, user guide
- Installation of XIOS could be not an easy task to do on a new machine, be sure it is already well installed with the right netcdf4 library !
- In the next croco version, XIOS version >=2

# XIOS: attached mode

Using xios in **attached mode** : each croco executable **compute** and **write** (like a classical library)



# XIOS: Why and when

CROCO
Coastal and Regional Ocean COmmunity model

- I/O becomes a bottleneck in parallel computing with using a large amount of processors

e.g. Atlantic model at **1km** resolution : 10000 x 14000 x 200 grid points ; using up to ~50000 procs
=> Very difficult or impossible to manage such amount of output datas with classical netcdf library.

- Only an external configuration file is needed to configure the outputs (no need to compile each time)
  - create new files
  - create new variables from referenced variables
  - use time filter (instantaneous, average, cumulate, ...)

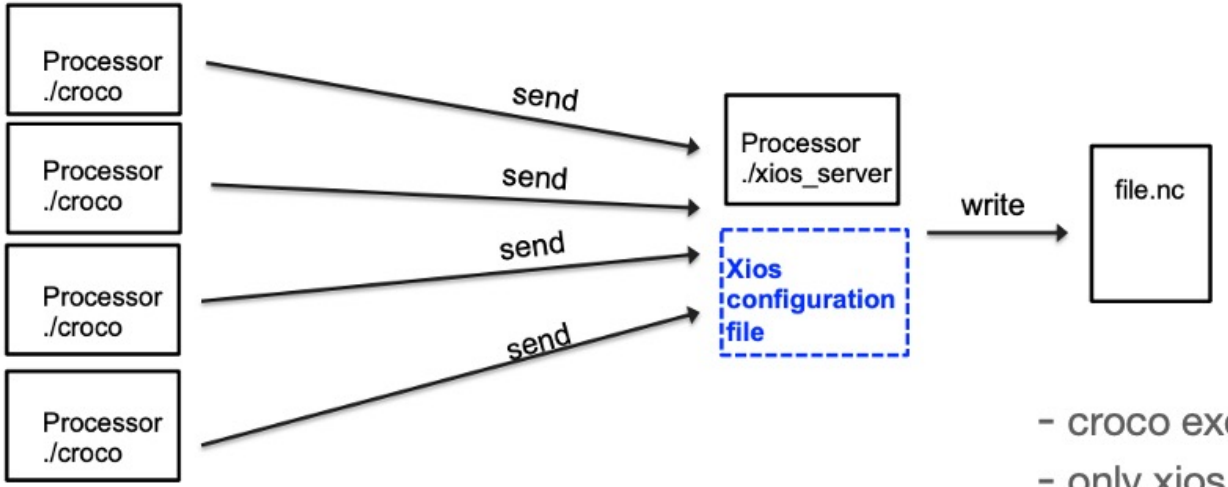
- Efficiency in production of data on supercomputer parallel file system

- Flexibility and “simplicity” in management of I/O and data definition

Remark : It is may be not so “ simple ” for beginners because you need to understand how to modify the configuration file written in xml

# XIOS: detached mode (server mode)

each croco executable compute and send field to the server




- croco executables only compute
- only xios server writes output

# XIOS: in practice

In cppdefs.h add  ccp keys : #define XIOS

- Add the XIOS library path in jobcomp
- Compile once : ./jobcomp

Edit/modify xios configuration file : iodef.xml

To run :

- in attached mode : as usual
- in detached mode : like a coupled model … (mpirun -np 10 ./croco : -np 2 ./xios.exe)