CROCO and Parallelisation : an overview

Rachid Benshila

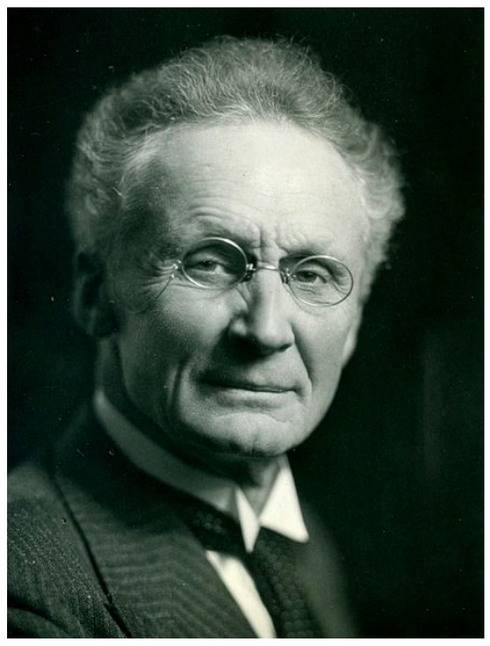
Background (from NWP)

Concept and techniques

How to use

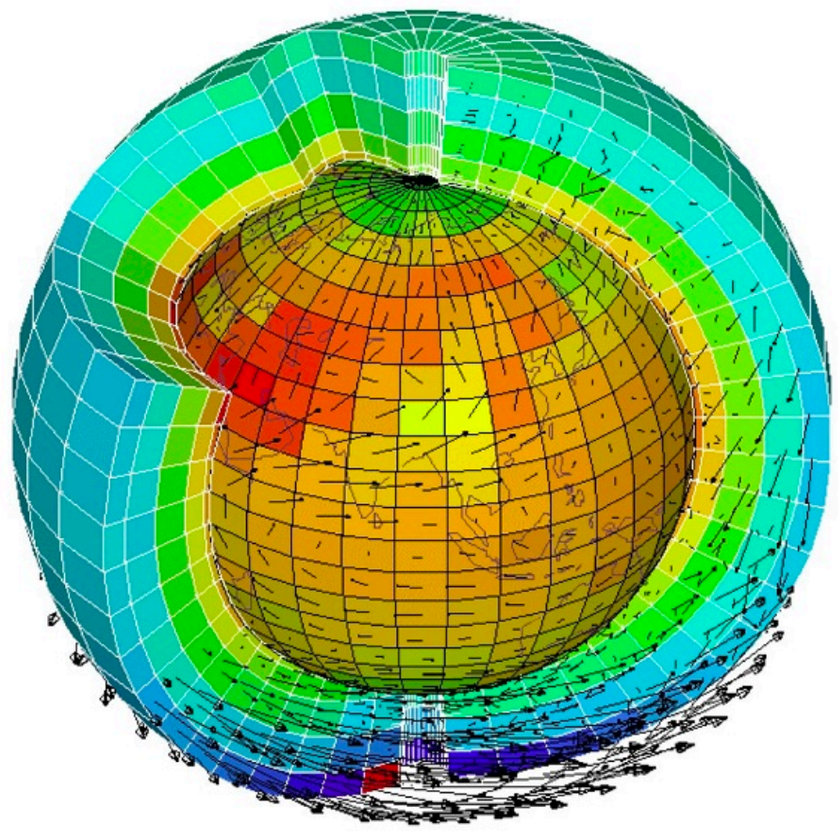
Related aspects (Advanced)

Numerical Weather Forecasting, the early days

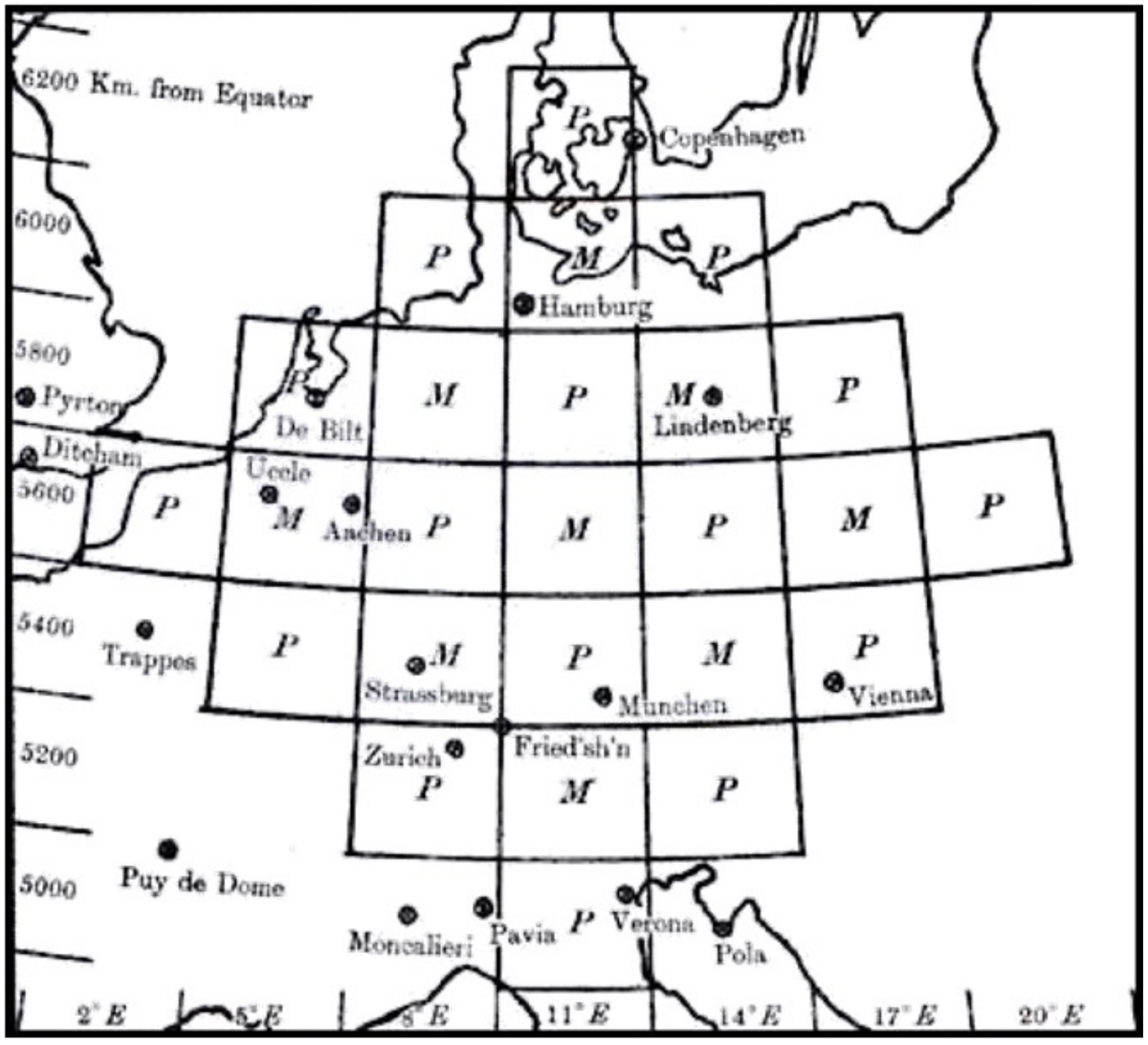


1904 : **Vilhelm Bjerknes** Mathematical Formulation
« Das Problem der Wettervorhersage, betrachtet von Standpunkt der Mechanik und Physik »
=> initial values + time integration of partial differential equation

1920 : **William Frye Richardson**
« Weather Prediction by Numerical Process »
=> discretisation and resolution



Numerical Weather Forecasting, the early days



"64,000 computers would be needed to predict in real time the evolution of weather over the entire globe" (L.F. Richardson, 1922).

First numerical forecast published in 1922 by Lewis Fry Richardson
Took several months, calculating by hand, to produce a 6-hour forecast.

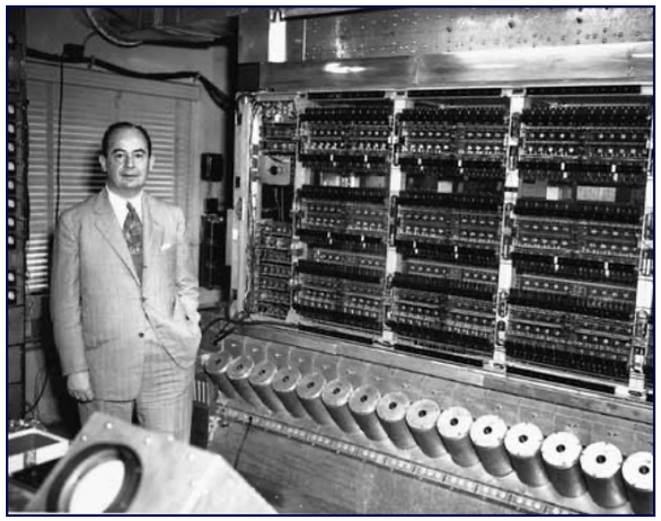
It failed…badly!

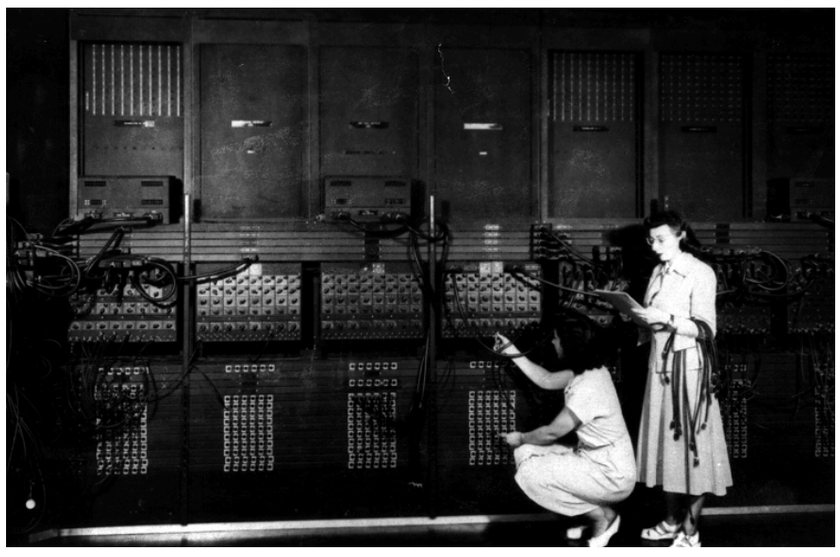
Numerical Weather Forecasting, the early days

Richardson's Forecast Factory

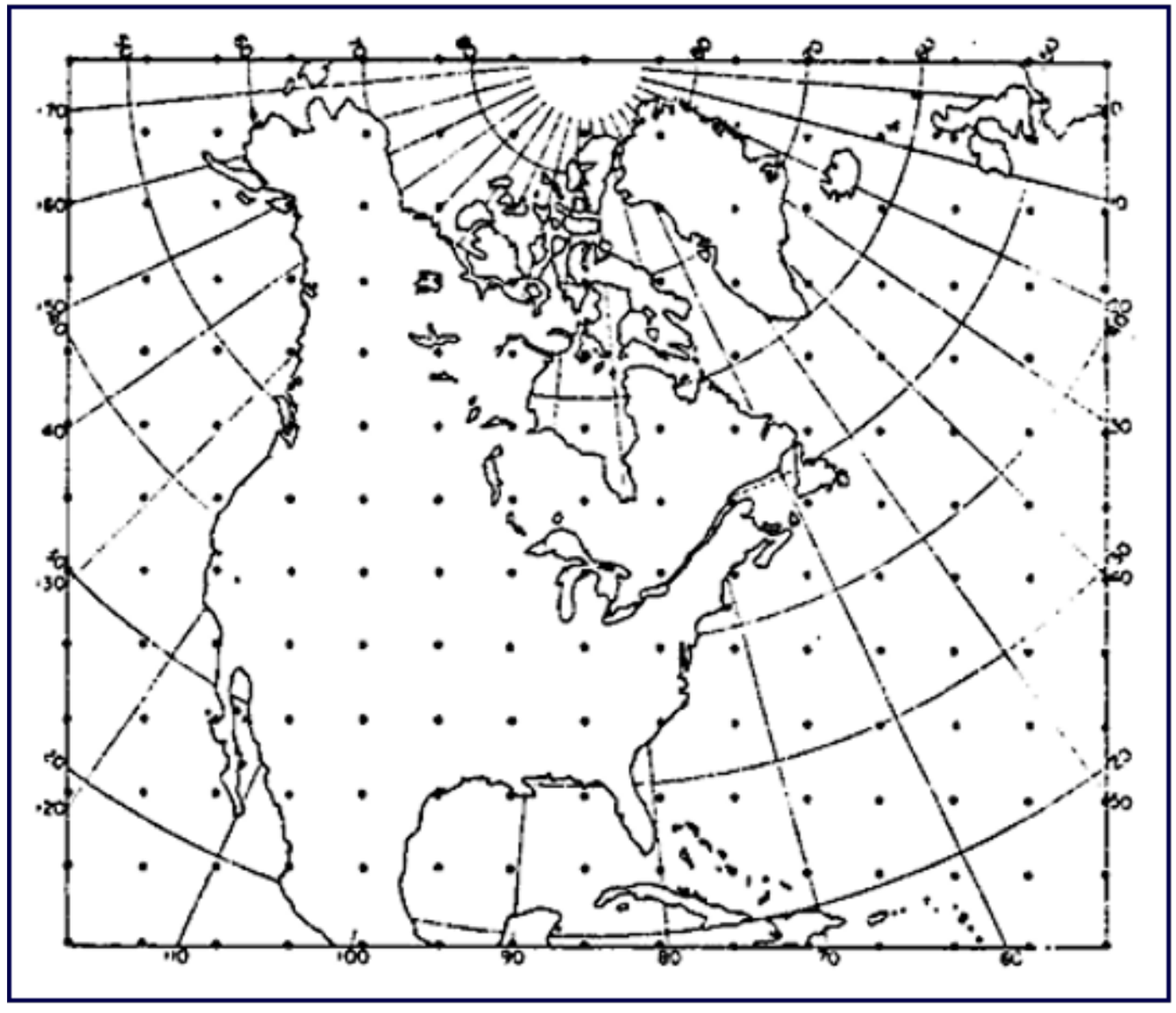
The working of the forecast factory is coordinated by a Director of Operations. Standing on a central dais, he synchronises the computations by signalling with a spotlight to those who are racing ahead or lagging behind …

Numerical Weather Forecasting, the very beginning





ENIAC : 30 tons, 200 digits, 500 Flops



(LENGAU : 1 PFlop)

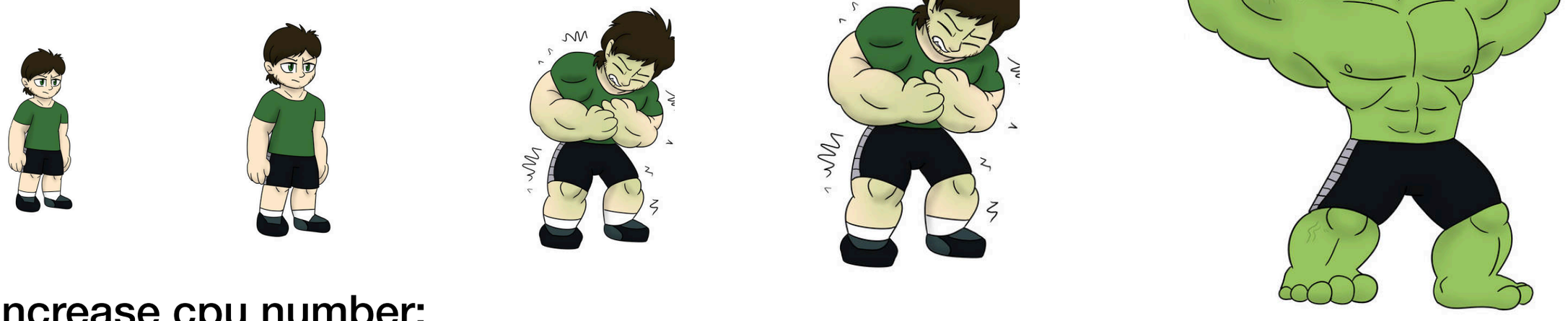
First successful forecast: 1950 by Jule Charney, Fjörtoft, and von Neumann, using ENIAC.

A **24-hour forecast** took **33 days** to produce, working day and night.

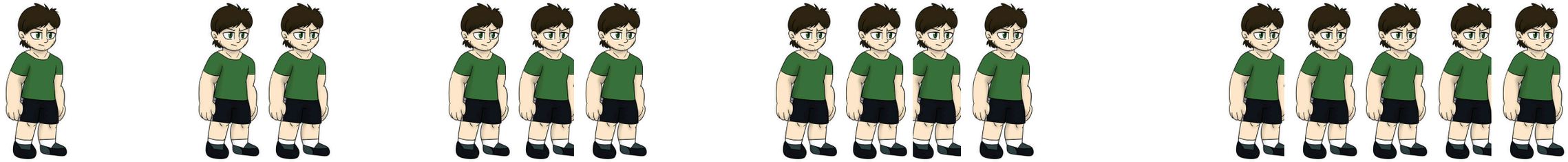
NWP :Getting to the FLOPS : from 100 to 1.e15

2 ways of evolution over time:

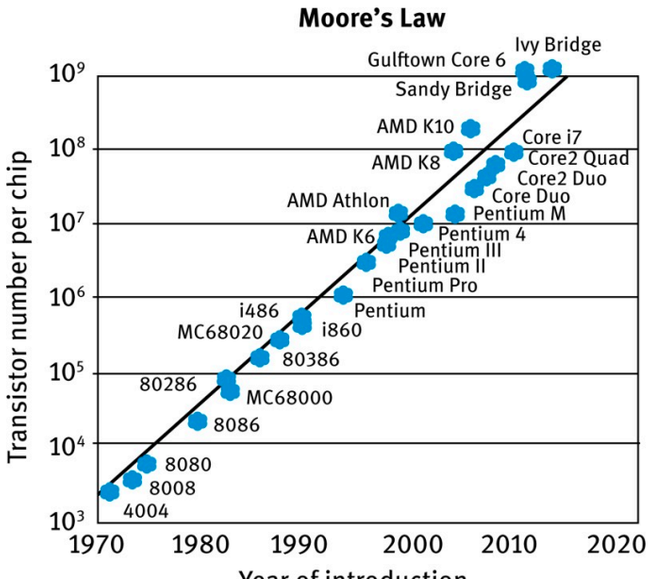
1. Increase cpu power (frequency):



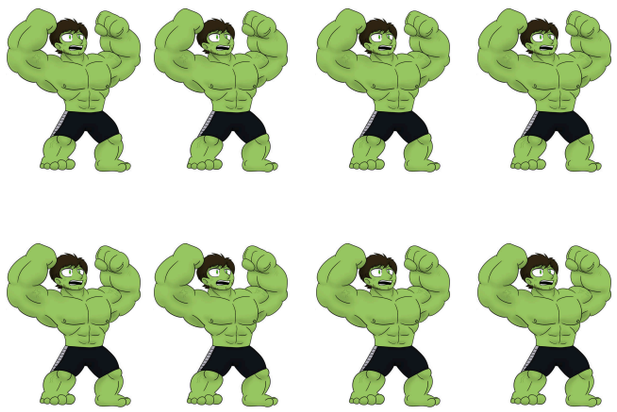
2. Increase cpu number:



In practice, both happened :

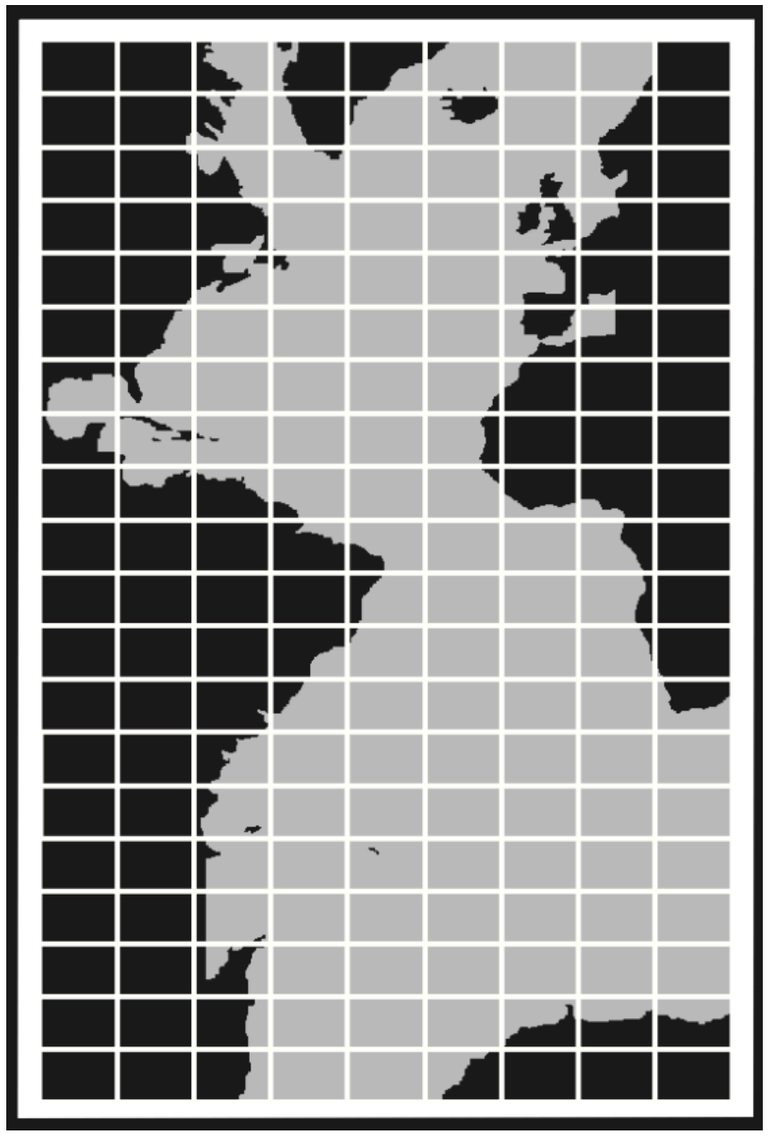


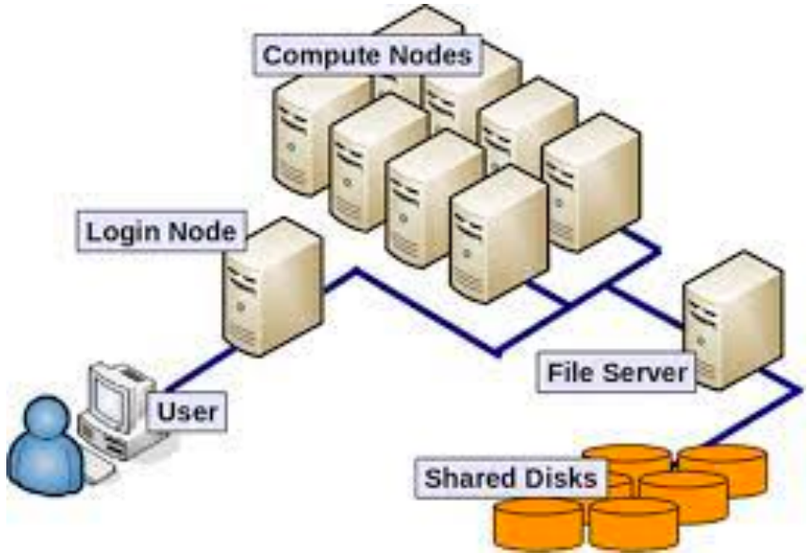
LENGAU :
32832 CPU
2.6 Ghz

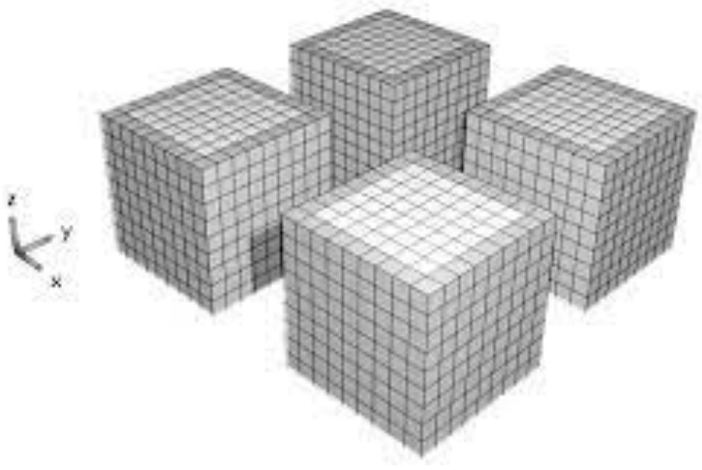


How to take advantage of it ?

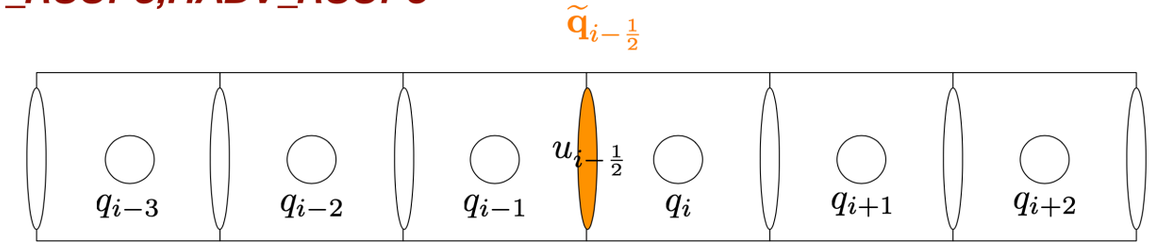
Parallelisation : domaine decomposition



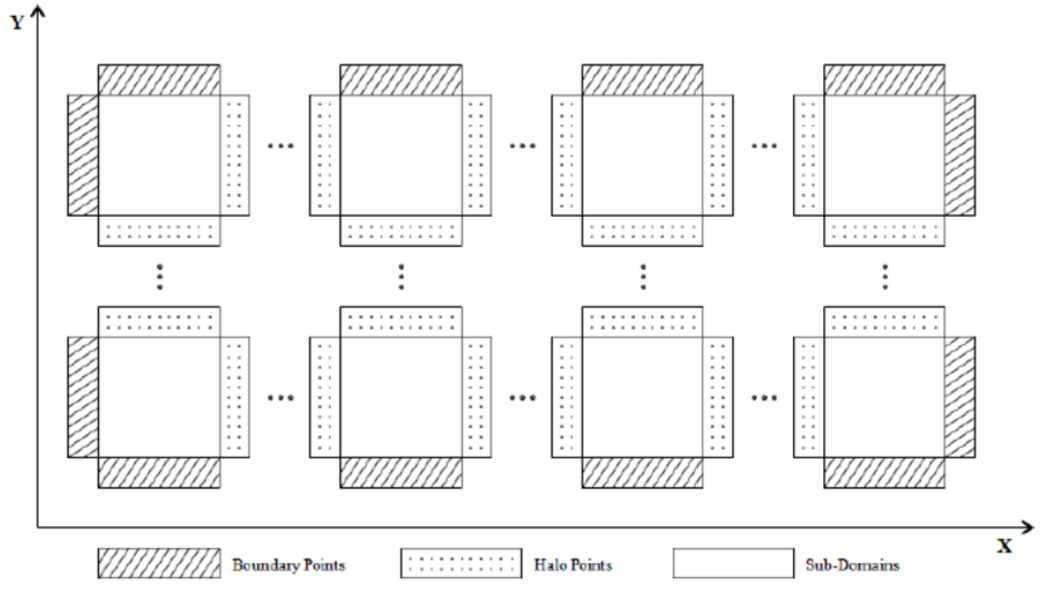




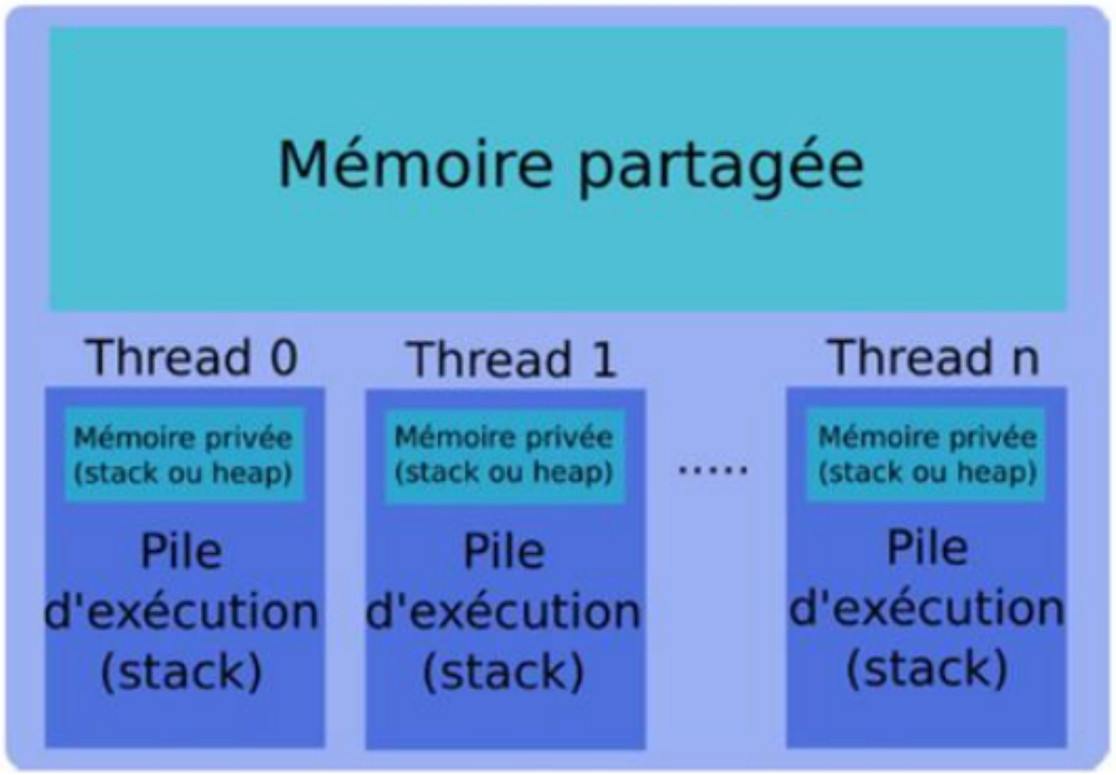
HADV_RSUP3,HADV_RSUP5

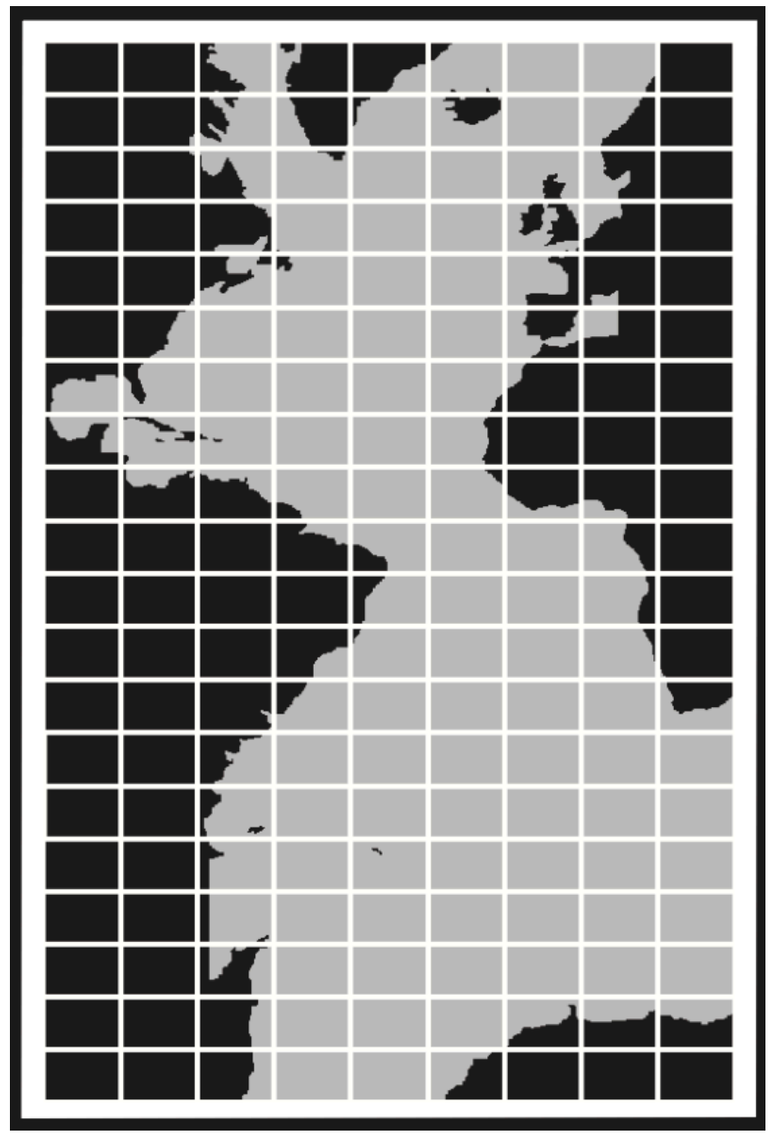


$$\partial_x(uq)|_{x=x_i} = \frac{1}{\Delta x_i}\left\{u_{i+\frac{1}{2}}\widetilde{q}_{i+\frac{1}{2}} - u_{i-\frac{1}{2}}\widetilde{q}_{i-\frac{1}{2}}\right\}$$



Approach 1 : shared memory

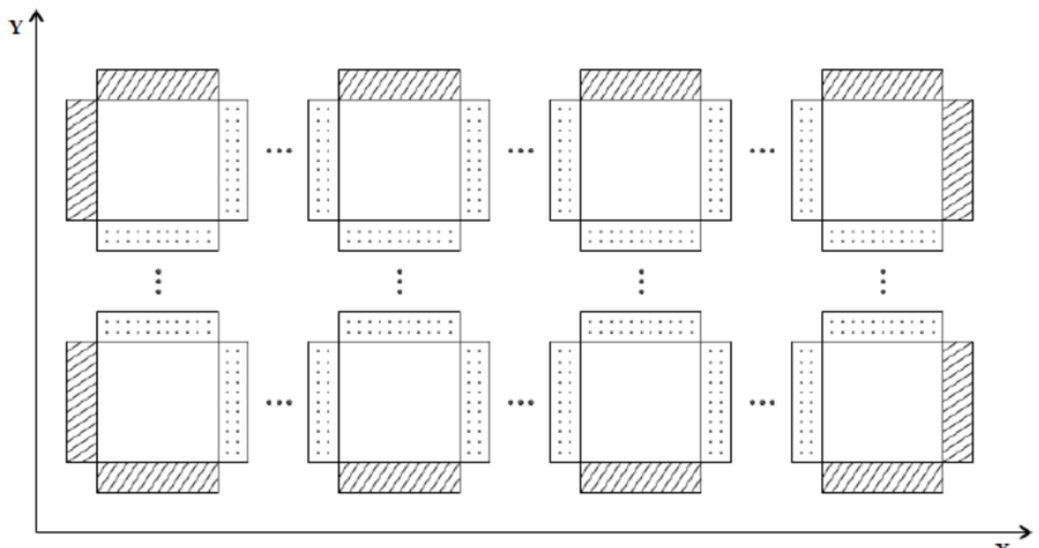




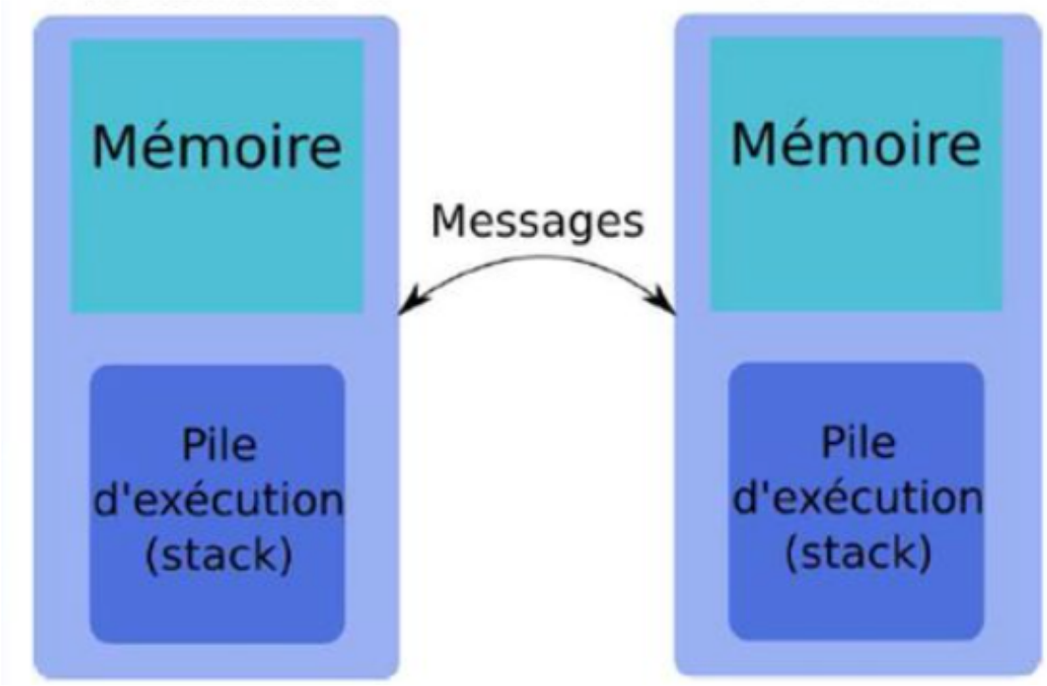
=> cores have access to a common shared memory

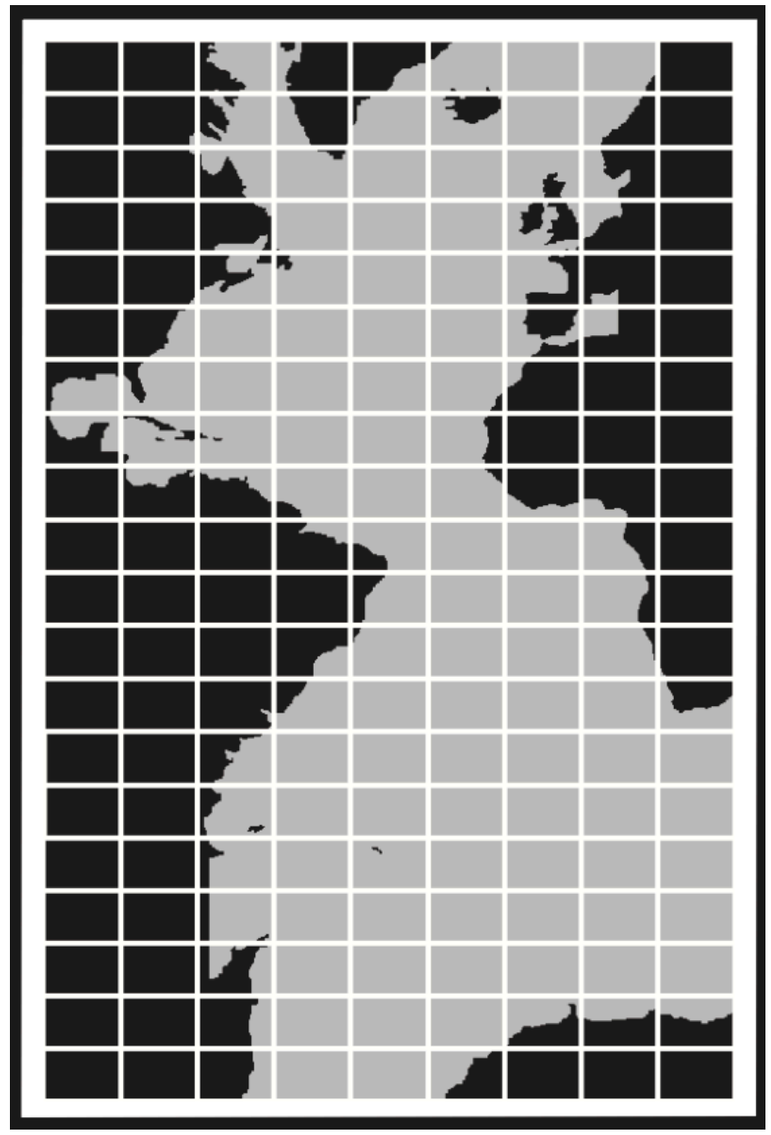
=>exchange of information through memory copy

Standard OpenMP
(Open Multi-Processing)



Approach 2 : distributed memory

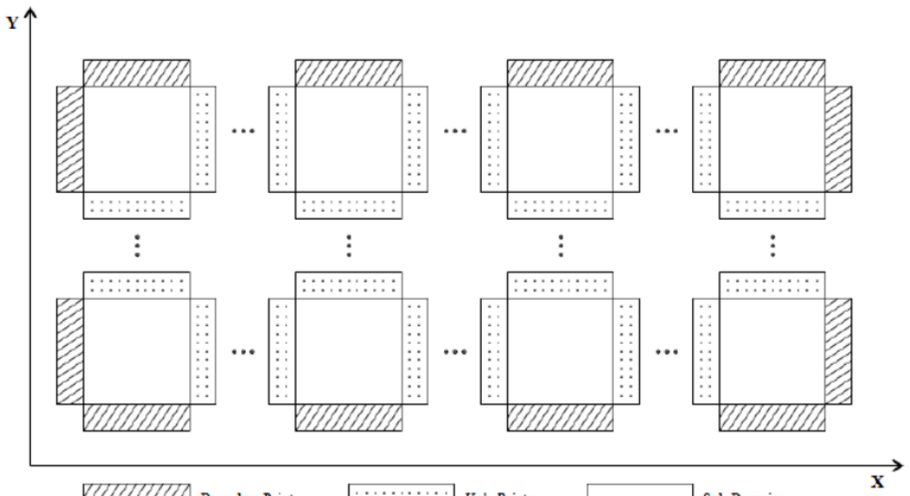




=> cores don’t have access to a common memory

=> exchanges through network and interconnection

=> in practice MPI can handle efficiently shared memory



Standard MPI
(Message Passing Interface)

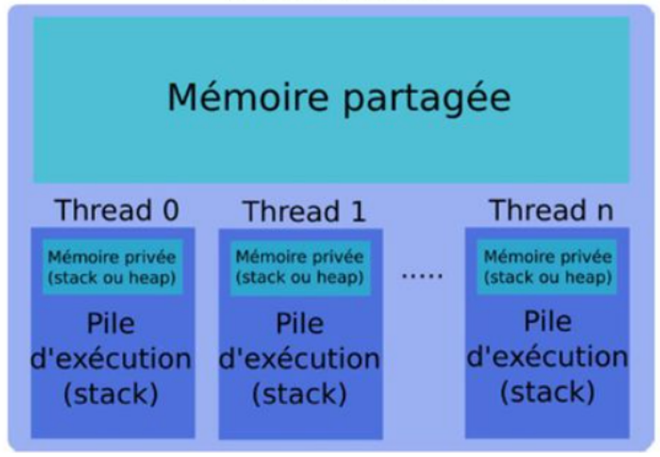
Implementation within CROCO : OPENMP

- step 1 : 2 files to edit
  - param.h
    specify the decomposition
    in x et y directions => NPP=4
  - cppdefs.h :
    activate OpenMP. => *#define OPENMP*

```
!
! 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)
```

- step 2 : compilation
  ./jobcomp

- étape 3 : execution
  - export OMP_NUM_THREADS=4
    specify the number of cores for the environment
  - ./croco



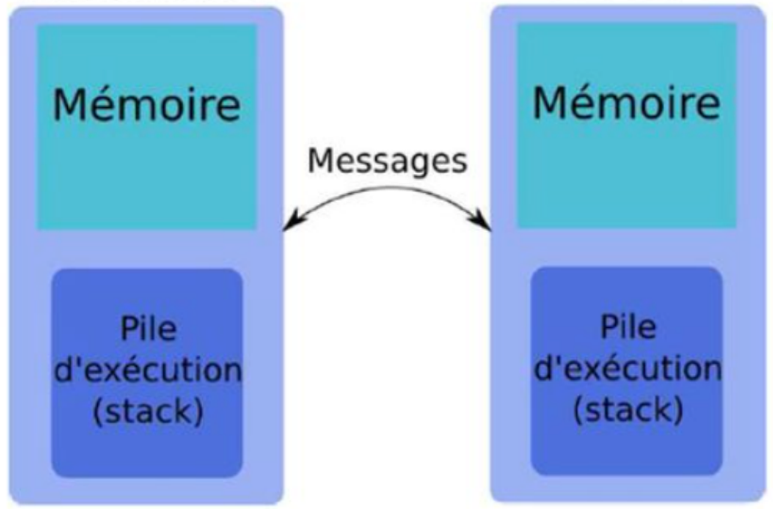
Implementation within CROCO : MPI

- step 1 : 2 files to edit
  - param.h
    specify the decomposition
    in x et y directions => NP_XI, NP_ETA
  - cppdefs.h :
    activate MPI => *#define MPI*

```
!
! 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)
```

- step 2 : compilation
  ./jobcomp

- étape 3 : compilation
  - mpirun -n 4 ./croco
    (or mpiexec or ….)



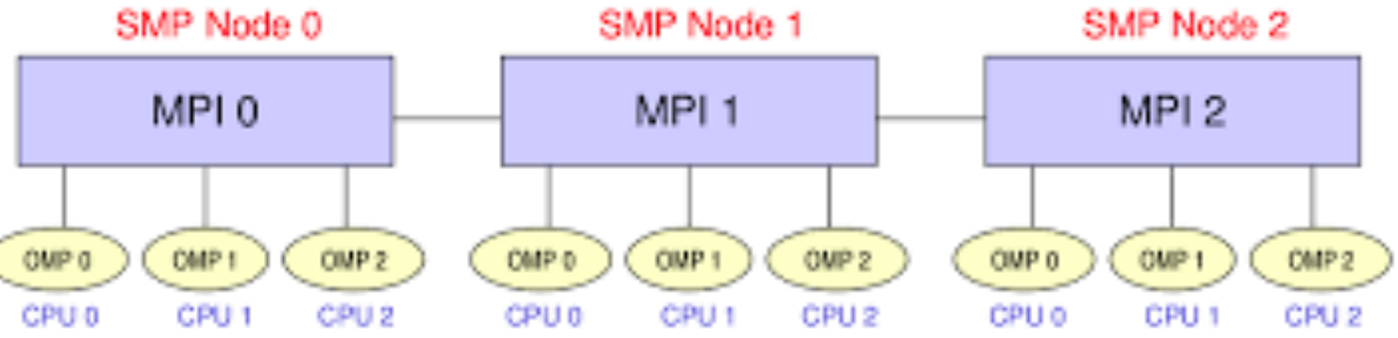
Summary and perspectives

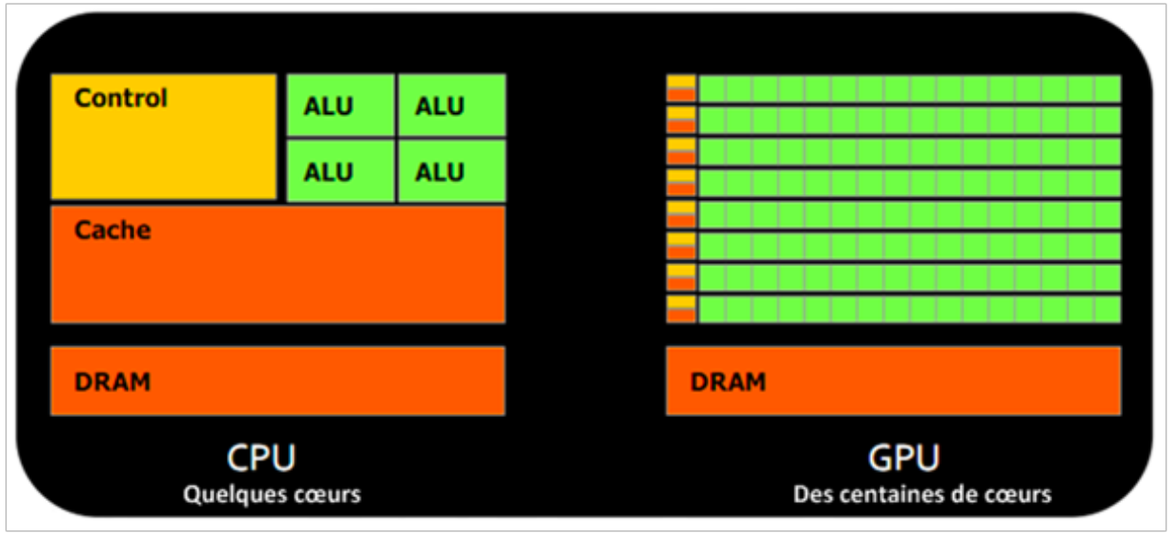
- 2 paradigmes available MPI et OpenMP

- code to re-compile !!

- MPI currently more used for Croco

- ETA direction for decomposition





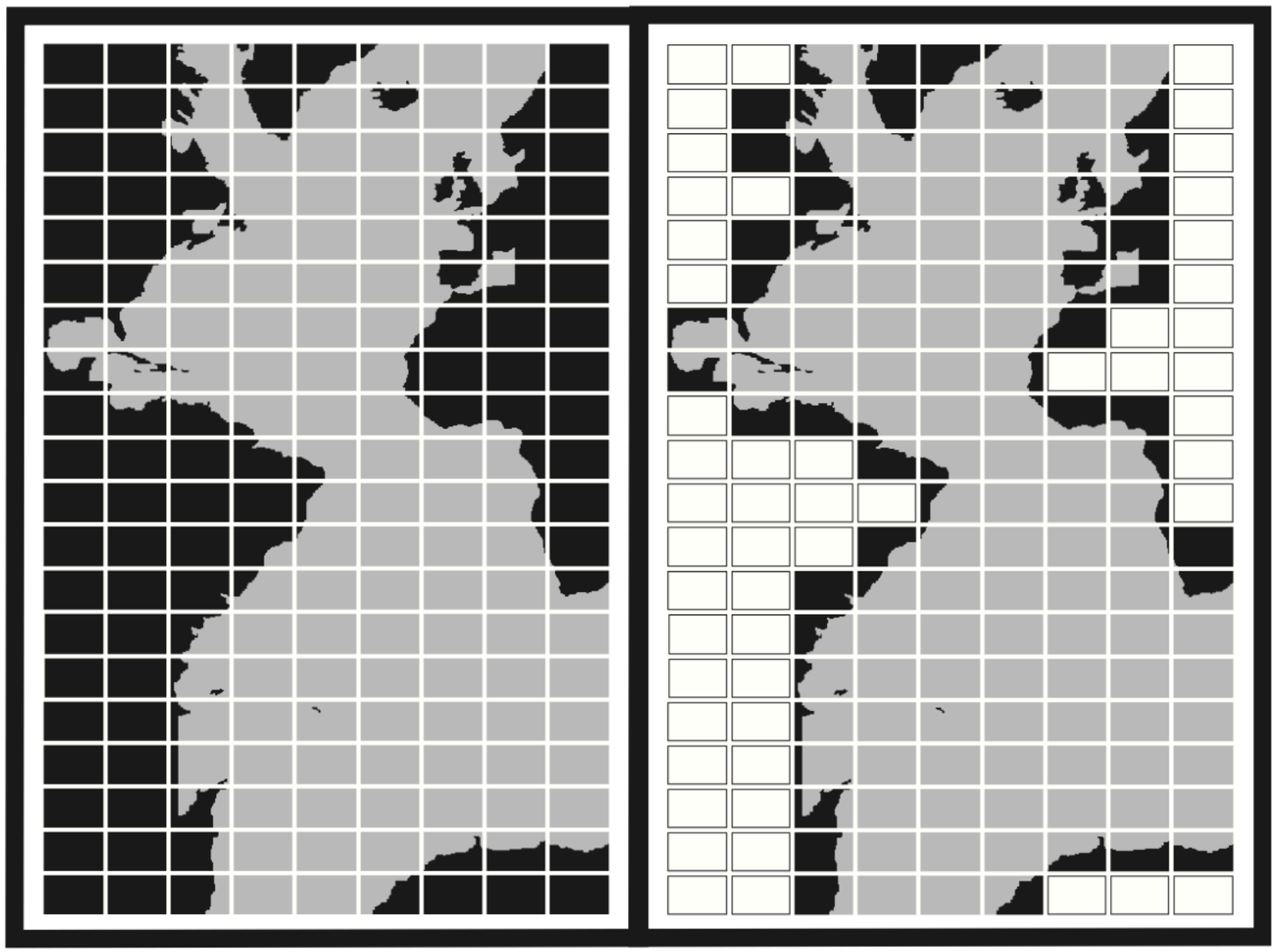
no hybrid MPI/OpenMP

GPU version underway

Parallelisation : but also ….

A few tricks :

- the land only processors case
- the output files case in MPI (super advanced)



The land only processors case

1. Preprocessing

In croco/MPI_NOLAND :
- read the README
- compile: edit makefile + make
- edit the namelist :
  - name of the grid file
  - number max of cores
- execute : ./mpp_optimize
- visualize :

./mpp_plot.py croco_grd.nc benguela-008x005_033

- re-read le README …

2. Before compiling CROCO

- cppdefs.h : #define MPI_NOLAND
- param.h : insert values for NP_XI, NP_ETA and NPP given by the preprocessing (NPP <= NP_XI x NP_ETA)
- execute as usual (mpirun -np *etc*)

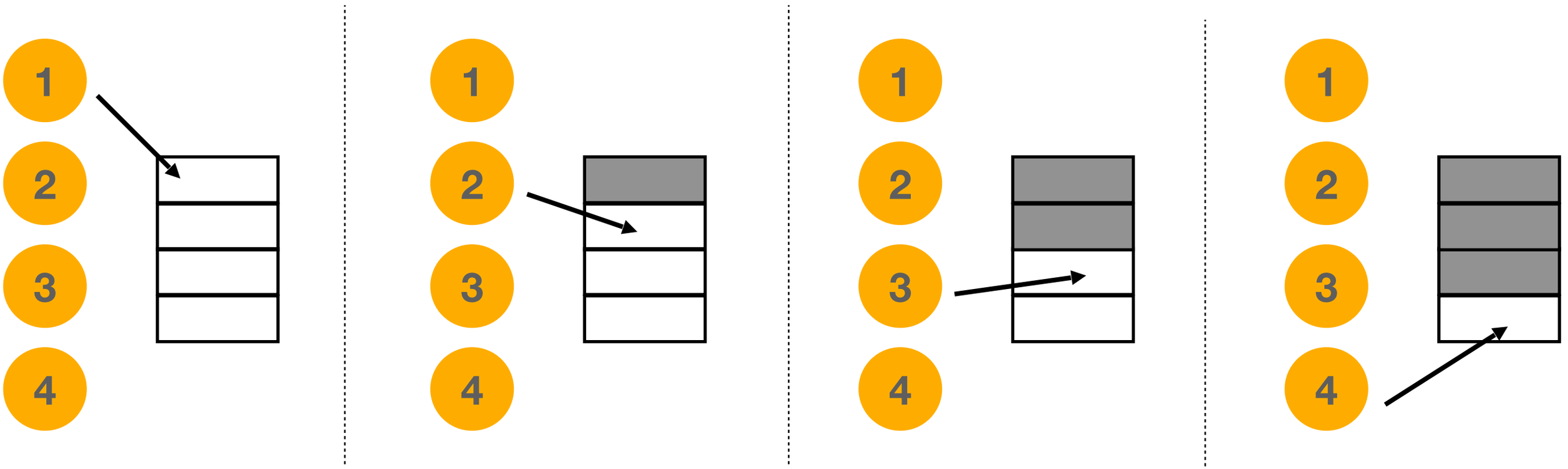
WARNING : grid file as to be called croco_grd.nc (or to be changed in MPI_Setup.F)

```
!'''''''''''''''''''''''''''''''''''''''
!      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'
/
```

```
-bash
(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$
```

MPI => Writing files 1/4 : default

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

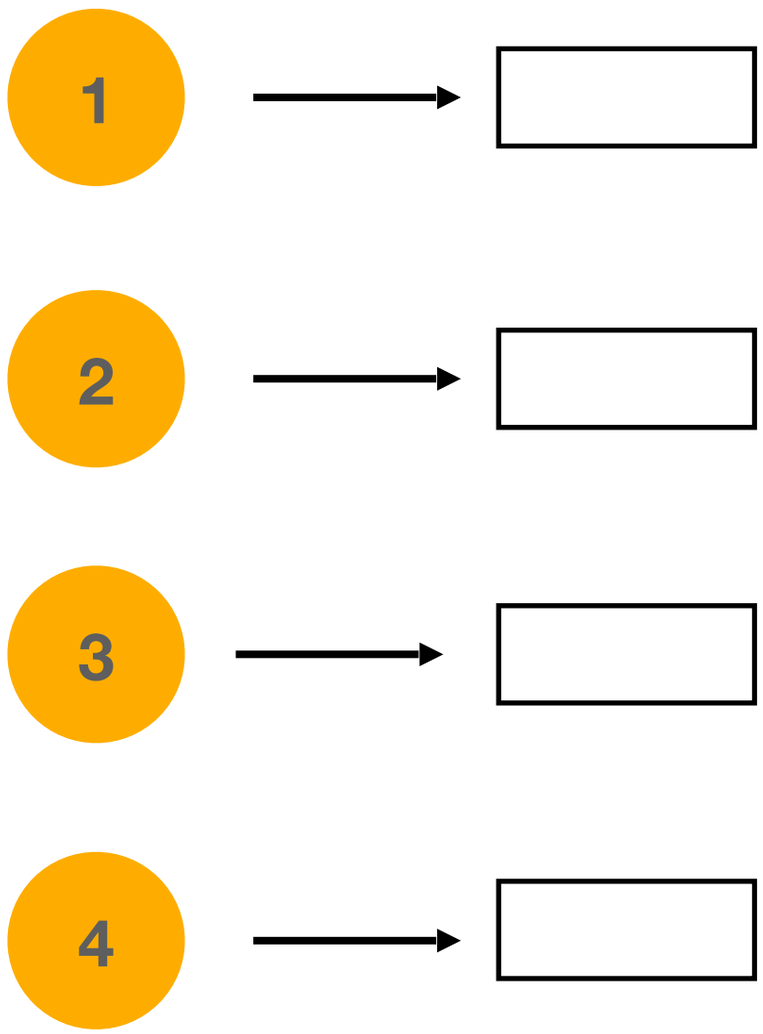


Unefficient !!!!!!!

MPI => Writing files 2/4 : parallel files

#define PARALLEL_FILES

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



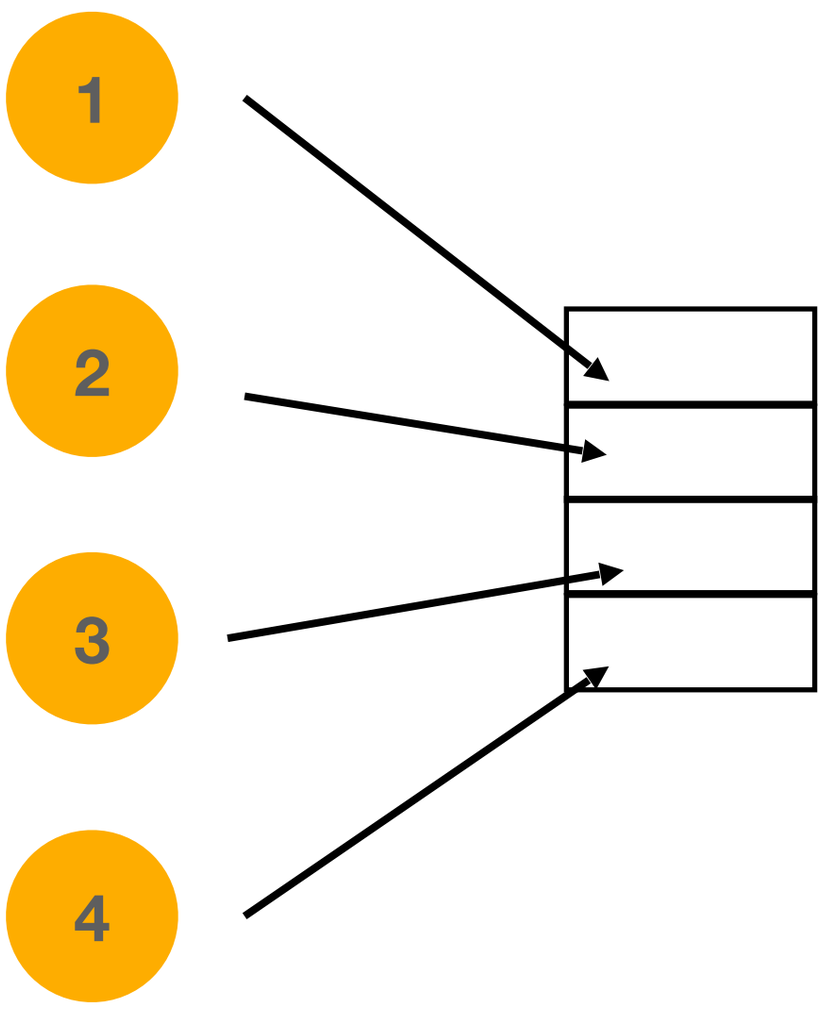
Fast but a lot of small files at the end

Need to reconstruct a global file (cf utility ncjoin)

MPI => Writing files 3/4 : parallel writing

#define KEY NC4PAR

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

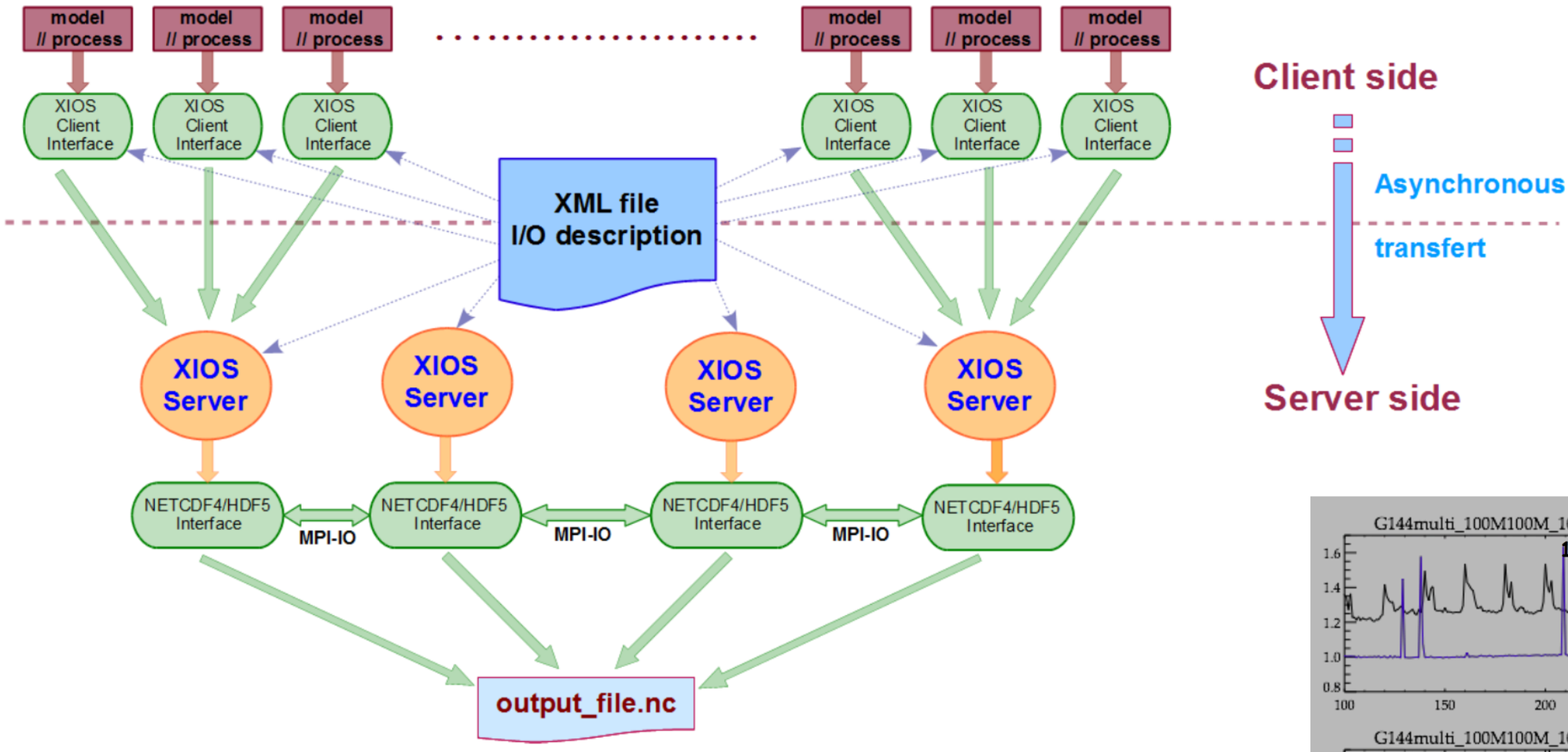


Fast with a global file at the end !!!

Need NetCDF4 library
built with parallel capabilities

MPI => Writing files 4/4 : XIOS

XIOS



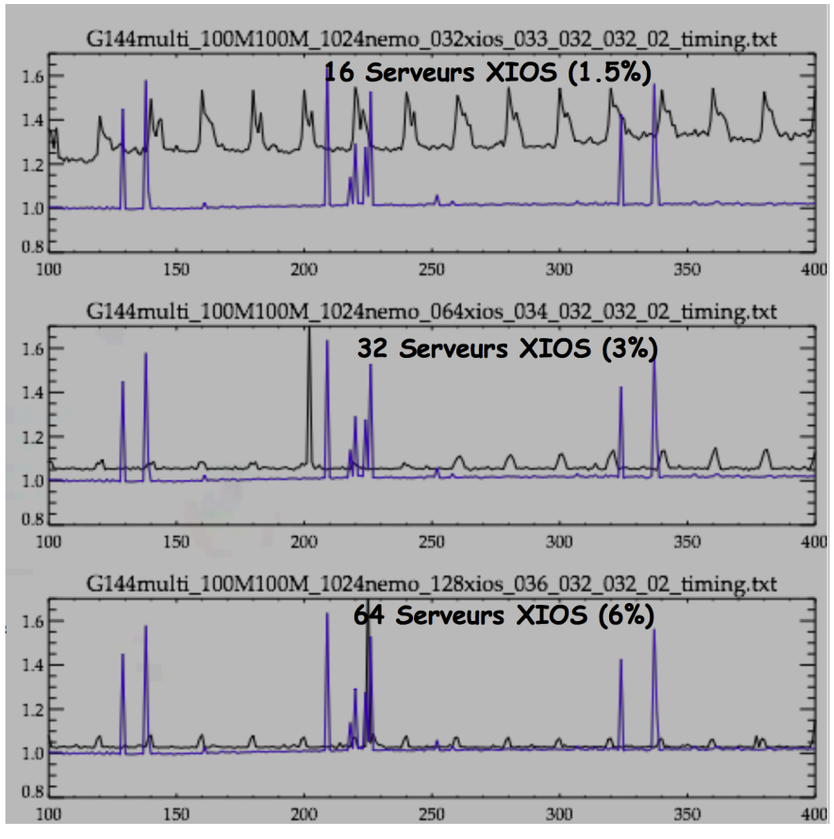
Strategy for outputs

XIOS : external server developed at IPSL

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

Exemple

(S. Masson, from NEMO …)



MPI => Writing files 4/4 : XIOS

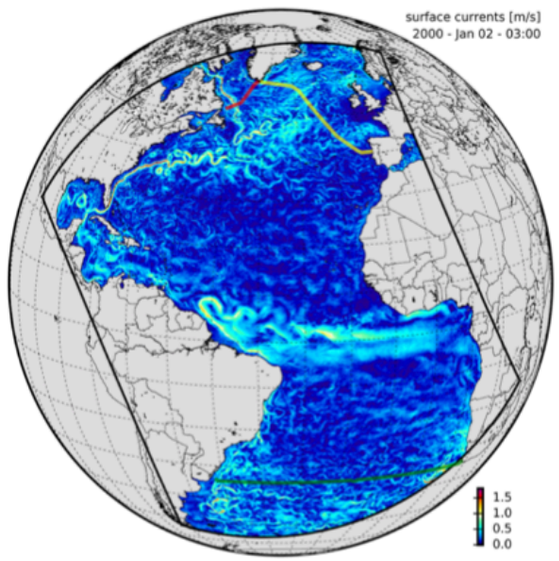
XIOS general

- 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

MPI => Writing files 4/4 : XIOS

XIOS why and when ?

- 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



3km resolution

=> 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, ...)

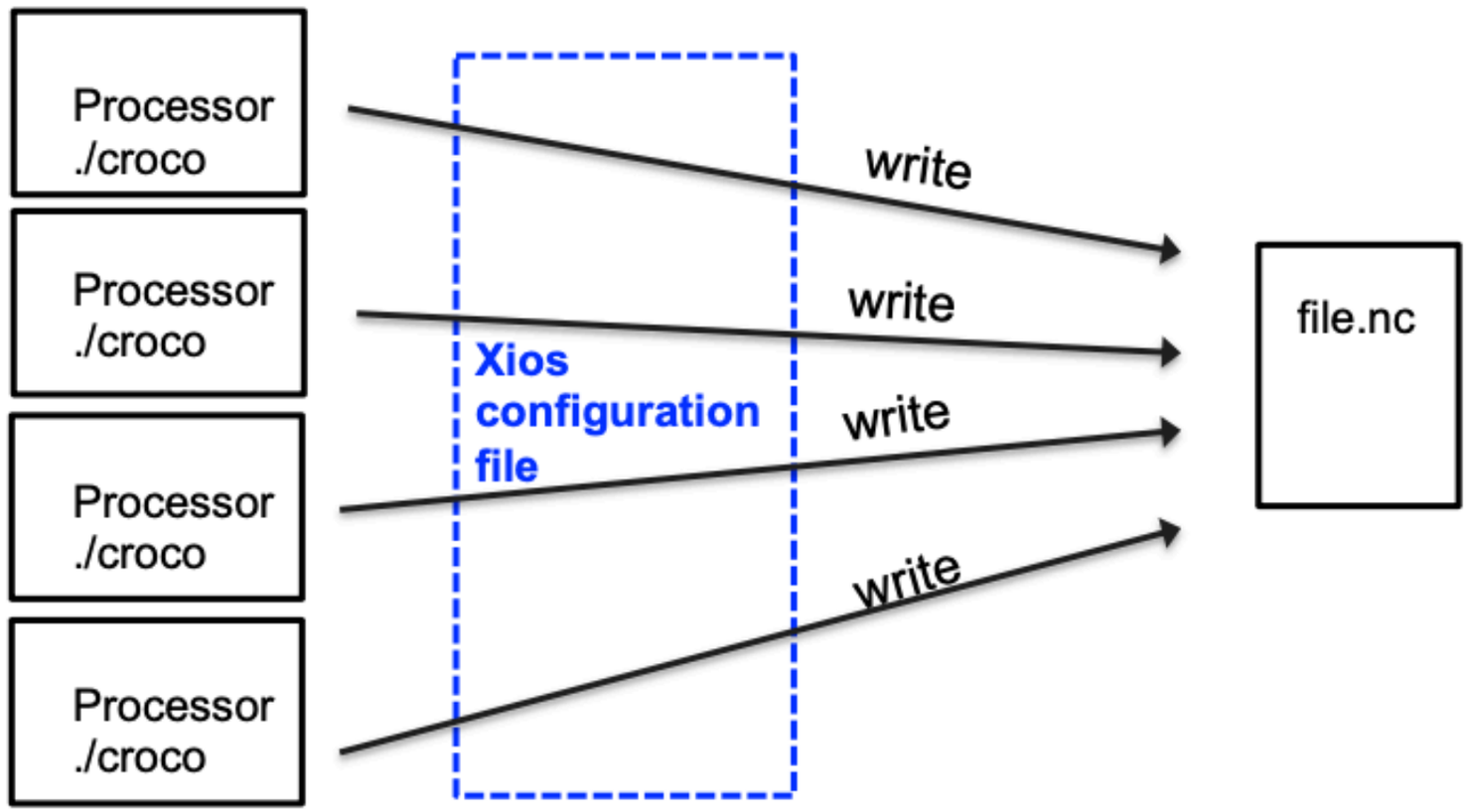
1. Efficiency in production of data on supercomputer parallel file system
2. Flexibility and “simplicity” in management of I/O and data definition

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

MPI => Writing files 4/4 : XIOS

XIOS : attached mode

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

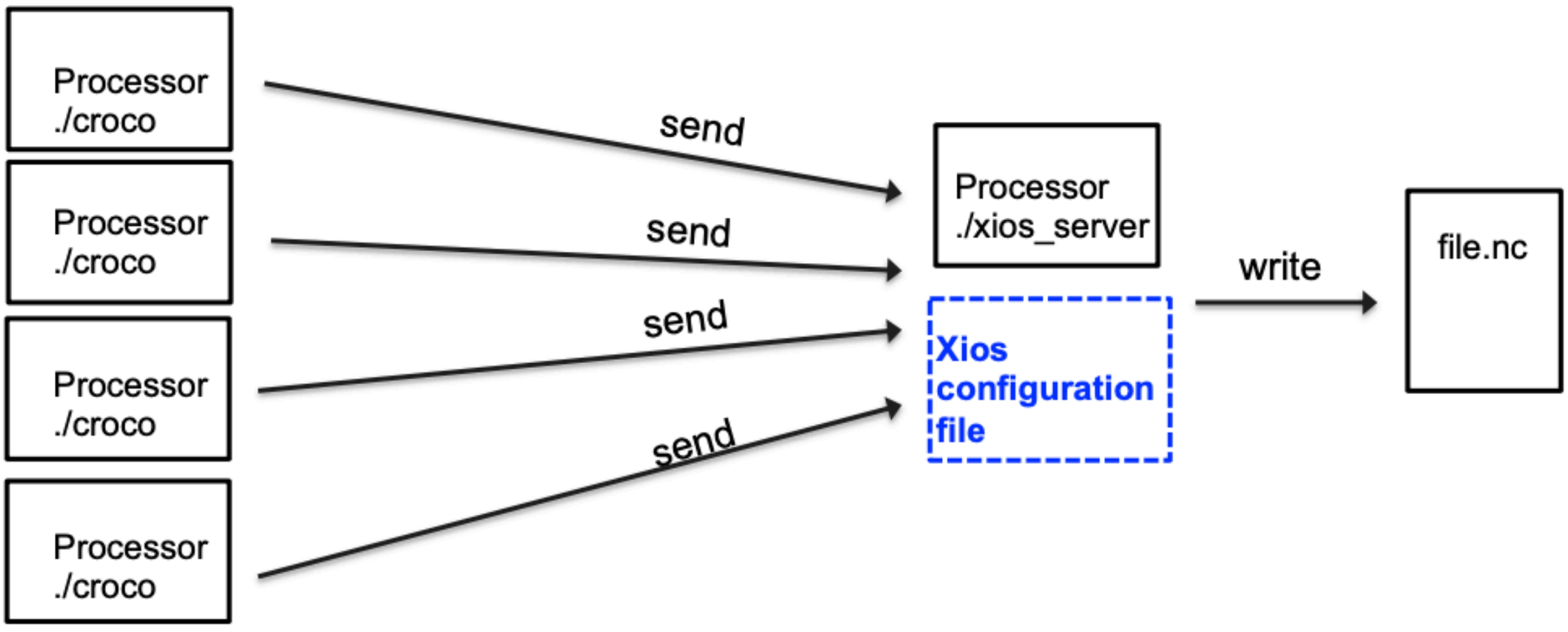


Ergonomy AND efficient parallel writing BUT writing overhead

MPI => Writing files 4/4 : XIOS

XIOS : detached mode (server mode)

each croco executable compute and send field to the server



- croco executables for computing only
- only xios server writes output
- Flexibility AND efficient parallel writing AND (almost) no overhead

MPI => Writing files 4/4 : XIOS

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