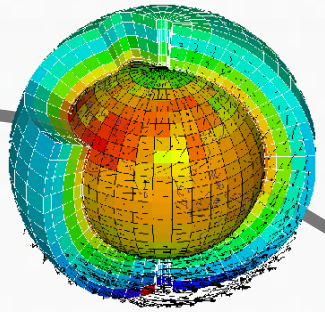
$$\frac{\partial \mathbf{U}}{\partial t} + 2\Omega \times \mathbf{U} = .$$

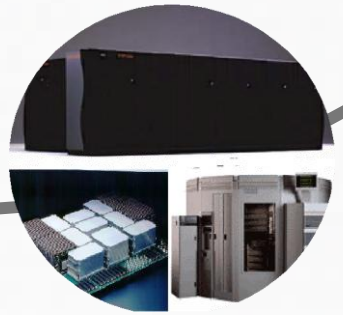


XIOS

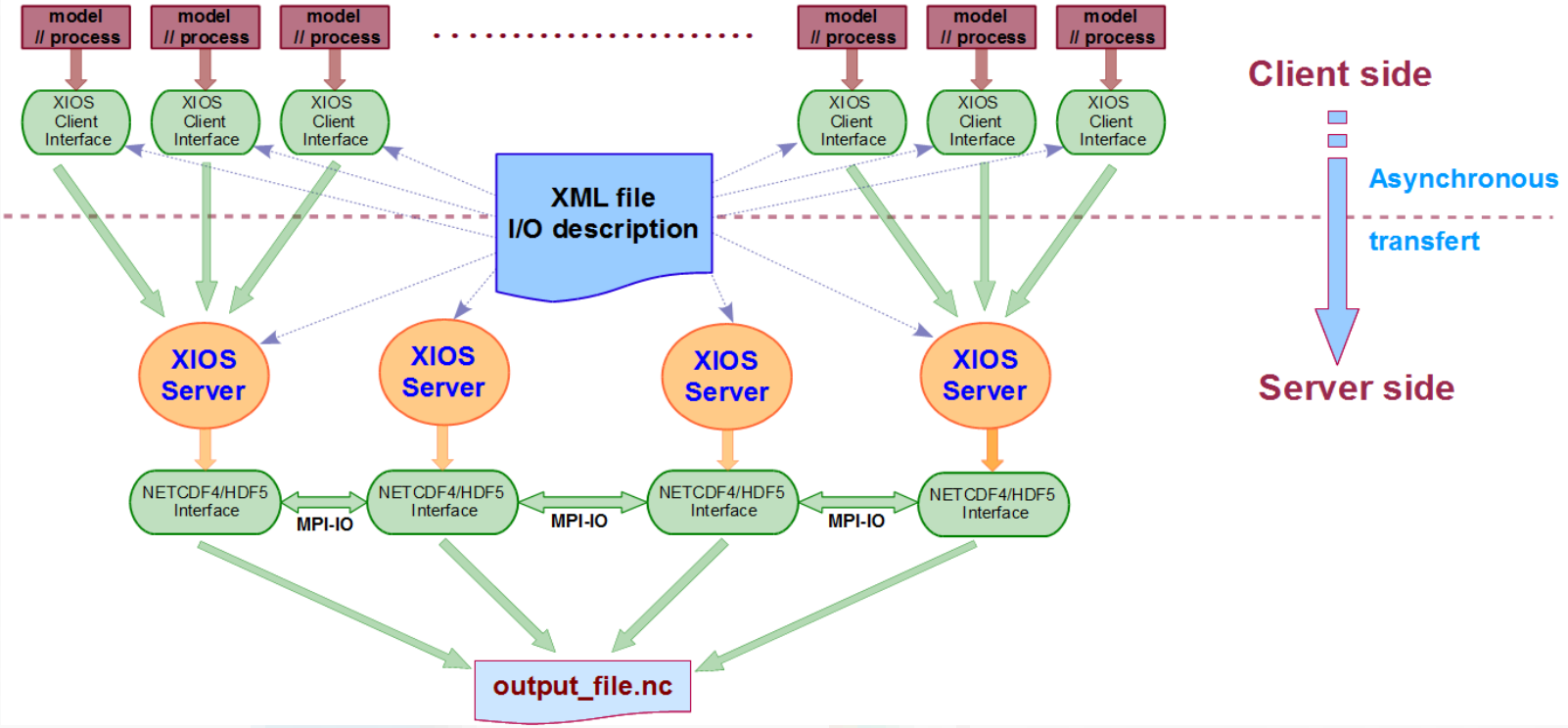
```
CTER*(*) pfilename
EGER,INTENT(IN) :: pim,pjm
EAL,DIMENSION(pim,pjm),INTENT(IN) :.
INTEGER,INTENT(IN):: par_orix, par_sz
INTEGER,INTENT(IN) :: pitau0
REAL,INTENT(IN) :: pdate0, pdeltat
INTEGER,INTENT(OUT) :: pfileid, phoriz

INCLUDE "netcdf.inc"

NTEGER :: ncid, iret
TEGER :: leng, lengf, lenga
CTER(LEN=120) :: file, tfil
R*30 :: timenow
```



XIOS - Motivation



- XIOS stands for XML - IO - SERVER
  - Library developed at IPSL and dedicated to IO management of climate model.
    - Management of output diagnostic (history file).
    - Online temporal post-processing operation (averaging, max/min, instant, etc…)
    - Online spatial post-processing operation.
  - Rewriten in C++ and improved as a part of the European IS-ENES project.
    - ~ 35 000 code lines under SVN : http://forge.ipsl.jussieu.fr/ioserver/browser/XIOS/trunk

XIOS main goals

- Flexibility
  - Simplification of the IO management into the model code
    - Minimize calling subroutine related to IO and metadata definition (file creation, axis and dimensions management, adding and output field…)
    - Minimize the number of argument of an IO call.
  - Ideally : output a field require only a identifier and the data.
    - `CALL send_field("field_id", field)`
  - Outsourcing the output definition in an XML file
    - Hierarchical management of definition with inheritance concept
    - Simple and more compact definition, avoiding unnecessary repetition of metadata parameters
  - Changing IO definitions without recompiling
    - Everything is dynamic, XML file is parsed at runtime.
- Performance
  - Targeted for large core simulation (> ~10 000) on climate coupled model.
  - Writing data must not slow down the computation.
    - Simultaneous writing and computing using asynchronicity.
  - Using one or more "server" processes dedicated exclusively to the IO management.
    - Asynchronous transfer of data from clients to servers.
    - Asynchronous data writing by each server.
  - Use of parallel file system ability via Netcdf4-HDF5 format.
    - Simultaneous writing in a same single file by all servers
    - No more post-processing rebuilding of the files

Simple "Hello World" XML file

- Output averaging field: field_A in the one day frequency file : hello_world.nc

```
<simulation>
  <context id="hello_word" calendar_type="Gregorian" start_date="2012-02-27 15:00:00">

    <axis_definition>
      <axis id="axis_A" value="1.0" size="1" />
    </axis_definition>

    <domain_definition>
      <domain id="domain_A" />
    </domain_definition>

    <grid_definition>
      <grid id="grid_A" domain_ref="domain_A"  axis_ref="axis_A" />
    </grid_definition>

    <field_definition >
      <field id="field_A"  operation="average" freq_op="1h" grid_ref="grid_A" />
    </field_definition>

    <file_definition type="one_file" output_freq="1d" enabled=".TRUE.">
      <file id="output" name="hello_world">
        <field field_ref="field_A" />
      </file>
    </file_definition>

  </context>
</simulation>
```

Fortran Interface

- Interoperability C/C++/Fortran through Fortran 2003 normalization feature.
- Every element in XML tree file can be created or fill in from code model through the Fortran interface

  - Create or adding an element in the XML tree

```
CALL xios_get_handle("field_definition", field_group_handle)
CALL xios_add_child(field_group_handle,field_handle,id="toce")
```

  - Complete or define attributes of an element
    - Using handle or id

```
CALL xios_set_field_attribut(field_handle,long_name="Temperature", unit="degC")
CALL xios_set_field_attribut("toce", enabled=.TRUE.)
```

"Hello World" : model side

```
SUBROUTINE client(rank,size)
  USE xios
  IMPLICIT NONE
  INTEGER :: rank, size
  TYPE(xios_time)       :: dtime
  DOUBLE PRECISION,ALLOCATABLE :: lon(:,:),lat(:,:),field_A(:,:)
  ! other variable declaration and initialisation .....

  CALL xios_initialize("client", return_comm=comm)
  CALL xios_context_initialize("hello_word",comm)
  CALL xios_set_current_context("hello_word")
  ! domain definition
  CALL xios_set_domain_attr("domain_A",ni_glo=ni_glo,nj_glo=nj_glo,ibegin=ibegin,ni=ni,jbegin=jbegin,
   nj=nj)
  CALL xios_set_domain_attr("domain_A",lonvalue=RESHAPE(lon,(/ni*nj/)),latvalue=RESHAPE(lat,(/ni*nj/)))
  dtime%second=3600
  CALL xios_set_timestep(dtime)
  CALL xios_close_context_definition()

  ! time loop
  DO ts=1,96
    CALL xios_update_calendar(ts)
    CALL xios_send_field("field_A",field_A)
  ENDDO

  CALL xios_context_finalize()
  CALL xios_finalize()
END SUBROUTINE client
```

XML structuration

- Inheritance : avoid unnecessary repetition of attributes
  - From parent to children

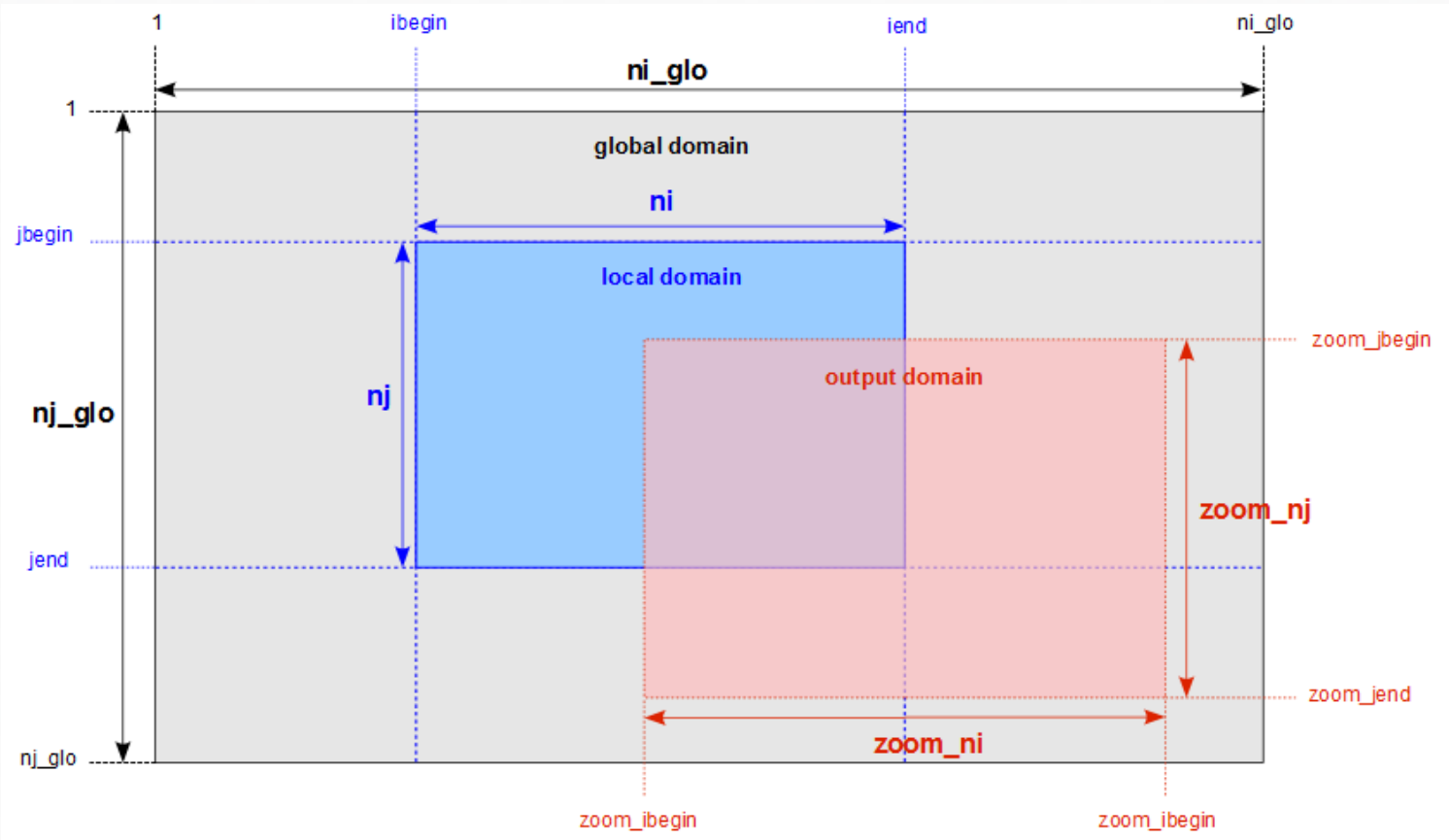
```
<field_definition level="1" prec="4" operation="average" enabled=".TRUE.">
  <field_group id="grid_W" domain_ref="grid_W">
    <field_group axis_ref="depthw">
      <field id="woce"       long_name="vertical velocity"           unit="m/s" />
   </field_group>
  </field_group>
</field_definition>

==>    <field id="woce" long_name="vertical velocity" unit="m/s" axis_ref="depthw"
         domain_ref="grid_W" level="1" prec="4" operation="average" enabled=".TRUE." />
```

- Context
  - isolate definition between different component : i.e. for coupled model
  - no interference between different context

- Calendar management and duration
  - **Gregorian, Julian, D360(360 days), NoLeap (365 days), AllLeap (366 days)**
  - **date format** : "2012-02-27 15:30:00"
  - duration units : year(y), month(mo), day(d), hour(h), minute(mi) and second (s)
  - can be combined : ex : "1d 6h 30mi"

- Grid managment
  - Only cartesian grid (lon-lat) or curvilinear grid are now supported (logically rectangular)
  - Soon unstructured grid (G8-ICOMEX) and gaussian grid
  - Describe global and local domain, and how field data are stored in code memory



Field variable

- Associated to a grid : "grid_ref" attribute
- Can be included in one or more file for output : "field_ref" attribute

```
<file id="1d" name="out_1day" output_freq="1d" enabled=".TRUE.">
   <field field_ref="toce" operation="average" enabled=".FALSE." />
   <field name="max_toce" field_ref="toce" operation="maximum" />
</file>
```

- Field value is sent at each timestep from the model
  - `CALL xios_send_field("field_id",field)`
- Temporal operations can be performed
  - once, instant, average, minimum, maximum
- Fields can be combined together to create new field which can be output

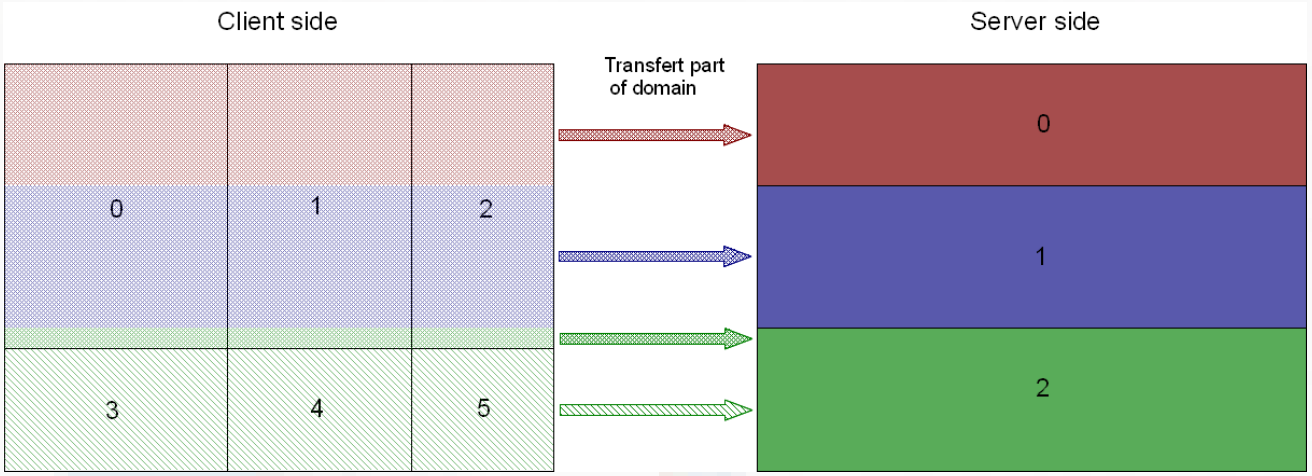
```
<field id="A" />
<field id="B" />
<field id="C" > (A + B) / (A*B)   </field>
<field id="D"> (this + exp(C))/3. </field>
```

- Can be mixed with temporal operation : @ operator
  - ie : output the monthly average of the daily maximum of the temperature

```
<field id="Temp" operation ="maximum" unit="K"/>
< file name ="output" output_freq ="1mo">
  <field name="T" operation ="average" freq_op="1d" > @Temp </field>
</ file >
```

Data distribution between clients and servers

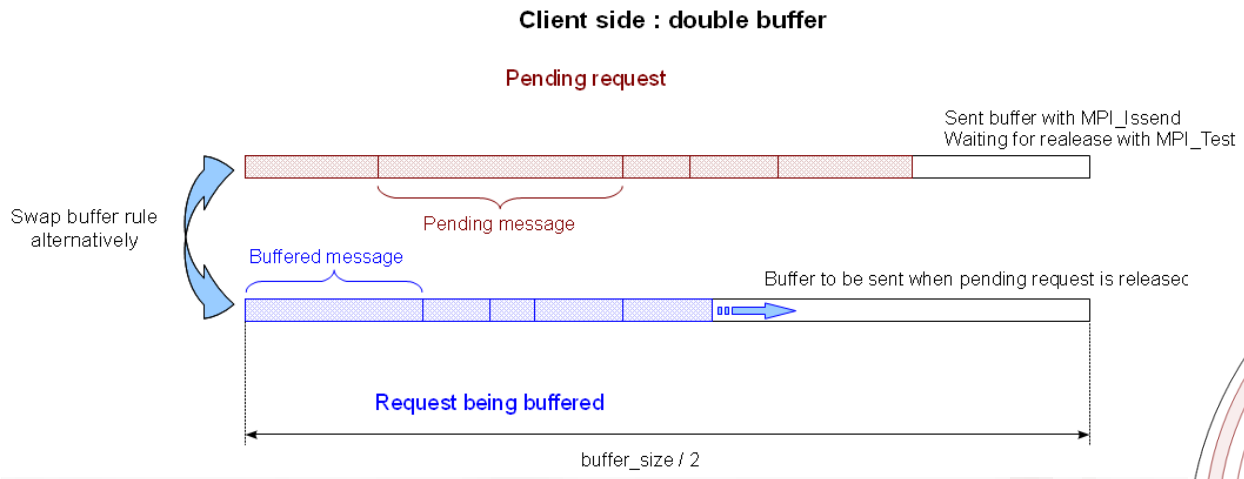
- Data distribution on client is defined by the model
- Data distribution on server is equally distributed over the second dimension
  - Client can communicate with several servers.
  - Server can receive data from several clients.



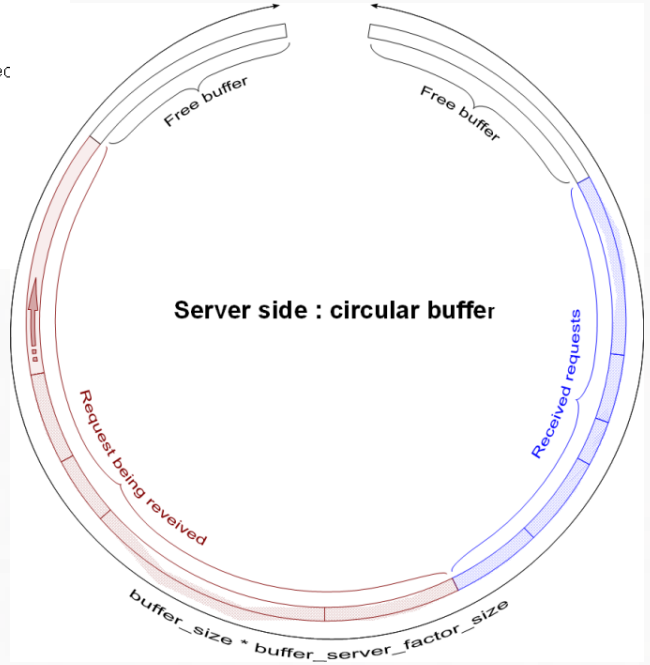
- Field Data are sent to servers only at writing time
- Use only MPI point to point asynchronous communication
  - MPI_Issend, MPI_Irecv, MPI_Test, MPI_Probe...
  - No synchronization point between clients and server, and between servers
  - No latency cost, communications are overlapped by computation
  - Writing is also overlapped by computation

Communication protocol

- Use buffers to smooth large peaks of data flow (monthly or daily output)
  - Larger are the buffers, better IO peak swill be smoothed
- Client Side protocol : use double buffer
  - one for message being transferring, one for buffering



- Server side protocol
  - Using circular buffer
  - Received requests can be processed at the same time as a new request is transferring



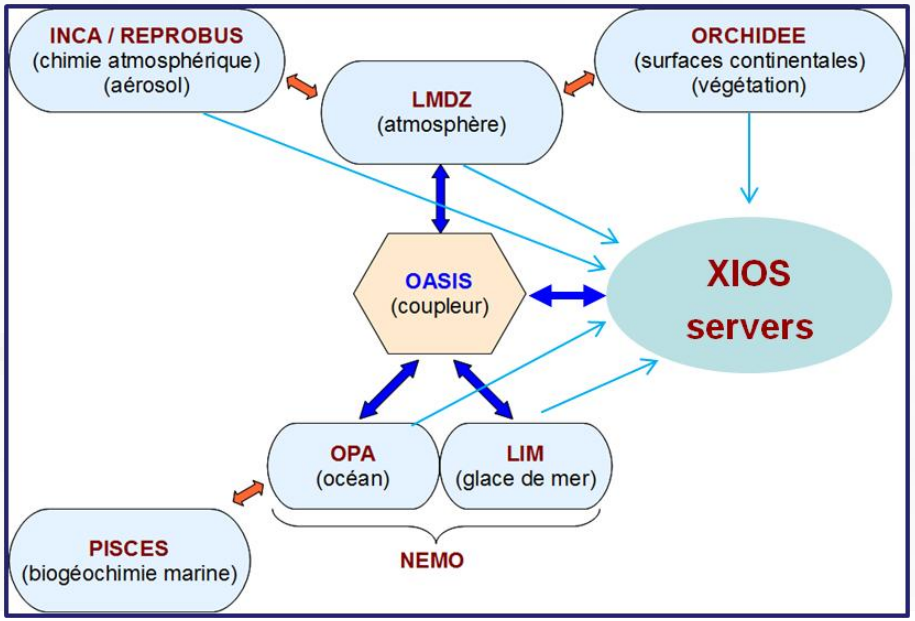
Load balancing – coupled model

Load balancing…

- Data flux from clients >> server writing rate capability
  - When buffers are full, XIOS switch in blocking mode
  - Wait until some buffer parts are released => impact on performance
- To avoid this, add more servers
  - a diagnostic is done at the end of the job.

Interaction with a coupled model

- XIOS has been designed to work within a coupled model
- A same pool of servers can manage several model output
- XIOS is also interfaced with the OASIS coupler



IO Layer

- For now, output layer use only NETCDF4/HDF5 parallel library.
  - optionally netcdf3 can be used, but no parallel support
- 2 modes are possible : "one_file" or "multiple_file"
  - Fixed using file attribute : type="one_file"/multiple_file"
- "Multiple_file" mode
  - One file is output by each XIOS servers
  - rebuilding phase is needed at post-treatment to obtain a global file
- "One_file" mode
  - All XIOS servers write simultaneously in a single file => no rebuilding phase
  - Use MPI/IO layer to aggregate file system bandwidth
- But achieve good performance with netcdf4/hdf5/MPI_IO layer is very challenging
  - strongly file system dependent
  - a lot of recipes to avoid very bad performance
  - a lot of work done for improving performances.

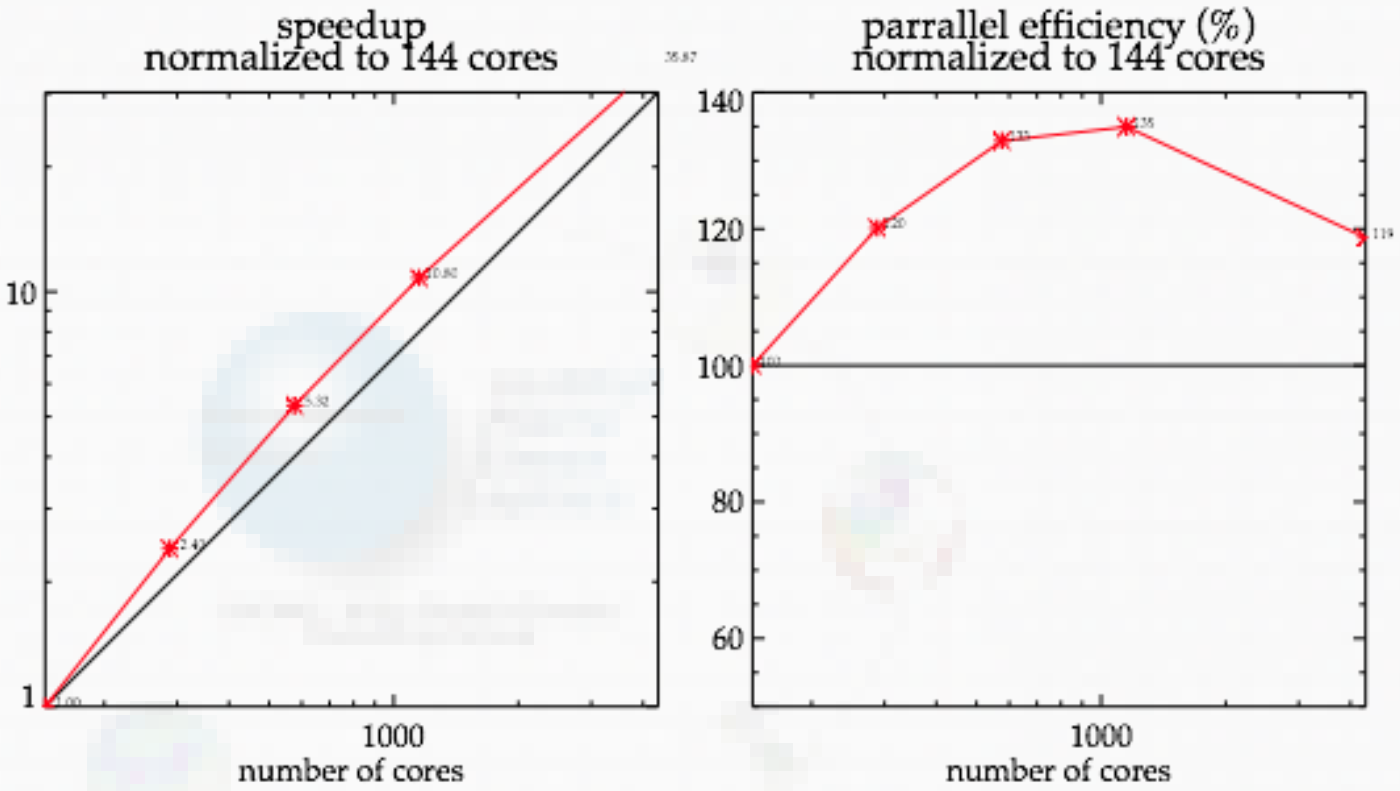
NEMO test case on Curie tier0 computer (S. Masson)

- Very huge NEMO configuration : 1/12th degrees global
  - GYRE 144 : 4322x2882x31 , up to 8160 cores
  - Run on Curie petaflopic Bull computer : 1.6 PFlops, Intel Sandy-Bridge core
  - 6 days simulation (2880 time steps), hourly means output : 300s run, ~1.1 Tb
  - **3.6 Gb/s, 13 Tb/hour, 312 Tb/day, 9.4 Pb/month (real time)**
- File system : Lustre 150 Gb/s (global) theoretically
  - In practice with an optimal MPI/IO simple parallel write test-case in a single file
    - must tune the number of OST used.
    - peak ~ 20 Gb/s, average 10 Gb/s.
  - With NETCDF4/HDF5/MPI_IO layer on an ideal test case
    - only MPI_IO call : ~ 8 Gb/s
    - whole < 5 Gb/s average

Multiple file mode

- Works fine, good scaling up to 8160 NEMO cores with 128 XIOS servers

CURIE Fat Nodes: NEMO 3_4_b GYRE Big IO multi_file, jp_cfg = 144



ncores:	144	288	576	1152	4352
timing:	8000	3330	1505	741	223

One file mode

- More challenging, recent results...
  - Gyre 144, daily mean output
    - 8160 NEMO, 32 XIOS : works perfectly
      - **+1.5% for IO** < computer jittering
  - Gyre 144, 6 hours mean output
    - 8160 NEMO, 32 XIOS
      - **+5% for IO**
  - Gyre 144, hourly mean output
    - 8160 NEMO, 128 XIOS
      - Extreme testcase, close to NEMO strong scalability limit.
      - Close to filesystem capability bandwith => we obtain ~ 3.6 Gb/s
      - **+ 15-20% for IO**
    - Do we need to improved this result ?
    - Are we able to store all this amount of data ?

THANK YOU

QUESTIONS ?