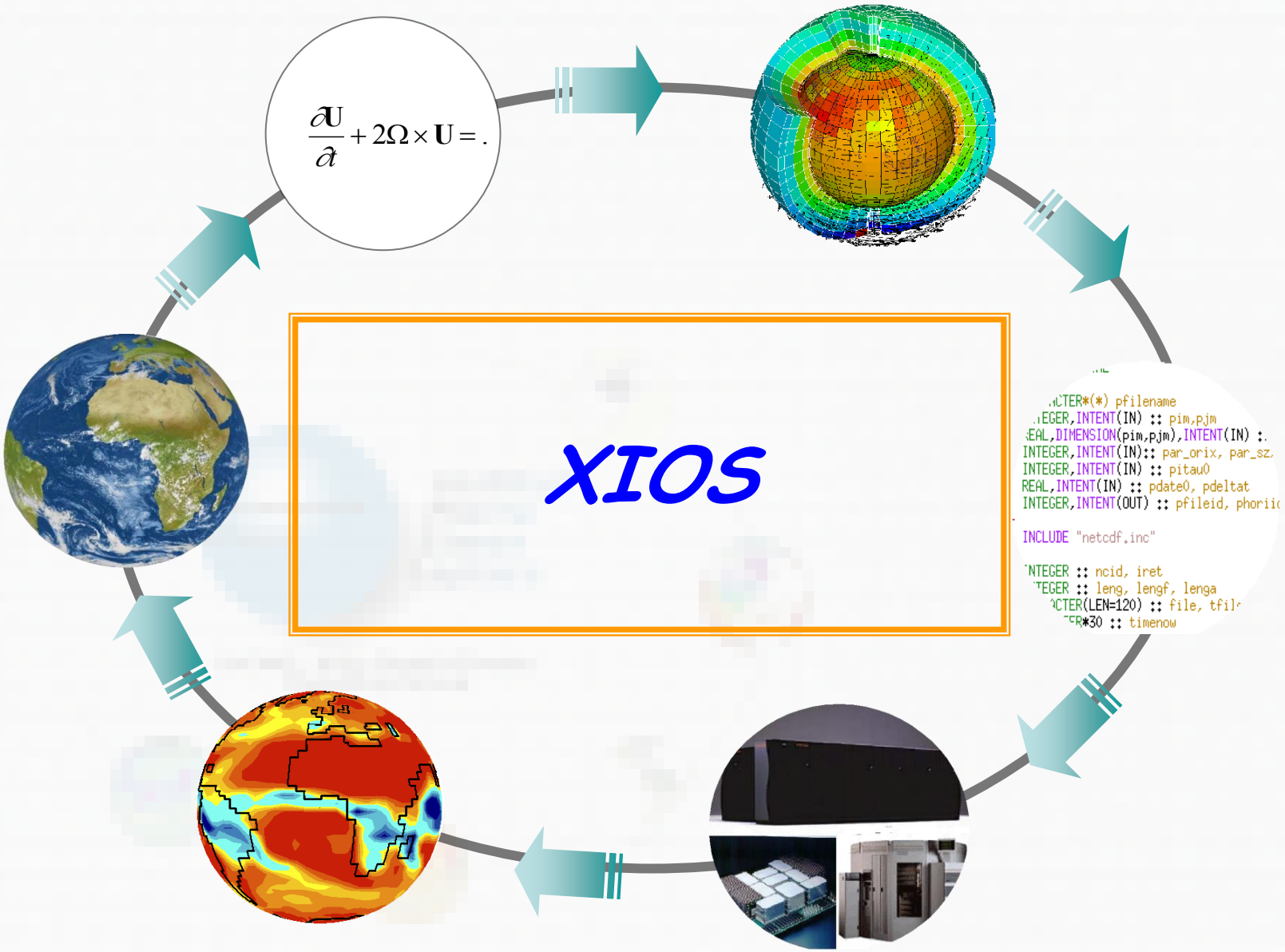
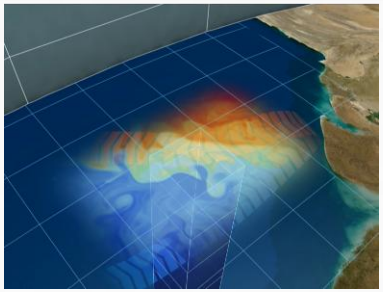
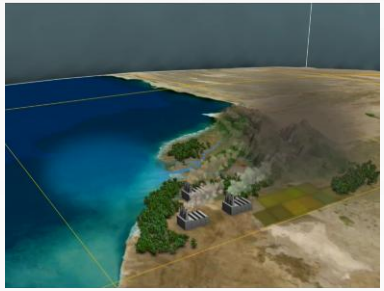
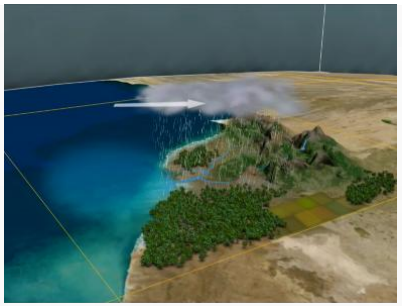

$\frac{\partial \mathbf{U}}{\partial t} + 2\Omega \times \mathbf{U} = .$
XIOS
CTER*(*) pfilename
EGER,INTENT(IN) :: pim,pjm
EAL,DIMENSION(pim,pjm),INTENT(IN) :.
INTEGER,INTENT(IN):: par_orix, par_sz.
INTEGER,INTENT(IN) :: pitau0
REAL,INTENT(IN) :: pdate0, pdeltat
INTEGER,INTENT(OUT) :: pfileid, phoriic
INCLUDE "netcdf.inc"
NTEGER :: ncid, iret
TEGER :: leng, lengf, lenga
CTER(LEN=120) :: file, tfil
R*30 :: timenow

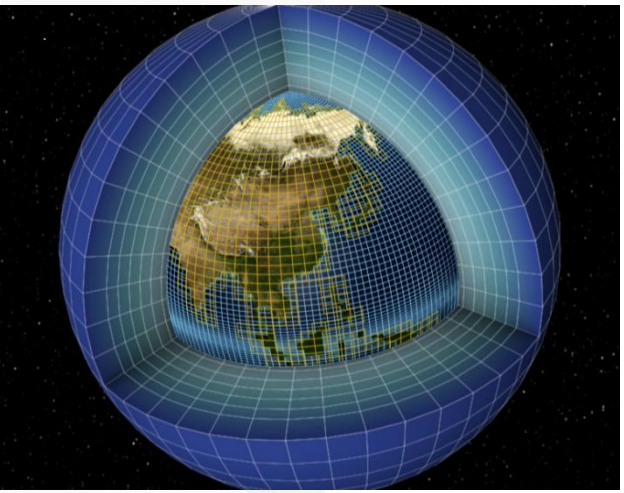
Le modèle couplé "Système Terre" de l'IPSL

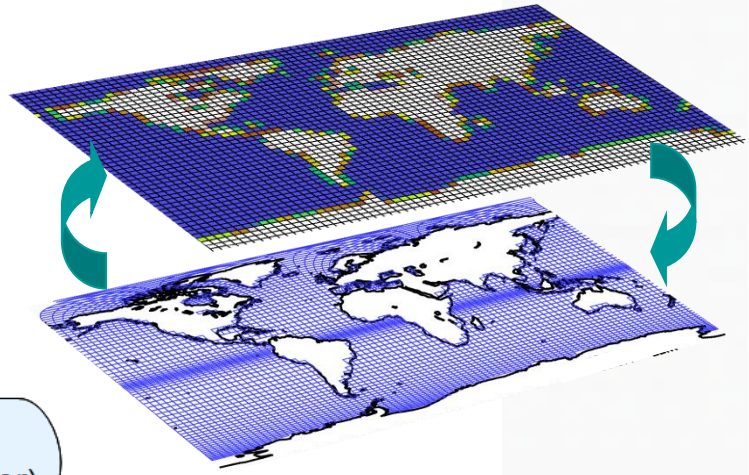
INCA / REPROBUS
(chimie atmosphérique)
(aérosol)

LMDZ
(atmosphère)

ORCHIDEE
(surfaces continentales)
(végétation)



OASIS
(coupleur)



OPA
(océan)

LIM
(glace de mer)

NEMO

PISCES
(biogéochimie marine)

L'exercice CMIP 5 – IPCC AR5

- **CMIP5** (Coupled Model Intercomparison Projet phase 5)
  IPCC AR5 (Intergovernmental Panel Climate Change, Assesment Report 5)
  - Des simulations « centennales » variées:
    - 20 et 21è siècles (historiques + scénarios futurs)
    - paléoclimat, dernier millénaire…
  - Des modèles de complexités différentes:
    - Modèle climatique "physique" (AOGCM)
    - Modèles avec cycle biogéochimique (modèle système Terre)
    - Configurations idéalisées (aqua-planète, …)
  - Des simulations décennales à hautes résolutions…
  - Fichiers journaliers et mensuels
    - plus de 800 variables différentes
- Produit un énorme flots de données
  => 2 Po de données produites pour l'IPSL dont 0.5 Po distribuées
  (le climat représente 80% du stockage de donnée au CCRT)
  - Production efficace des données
  - Post-traitement
  - Stockage
  - Distribution

Caractéristiques des sorties "histoires" des modèles de l'IPSL

Fichiers

- Chaque modèle produit ces propres fichiers « histoires » au format netcdf
- Les fichiers sont composés de plusieurs dizaines de variables sur la grille du modèle, intégrées sur plusieurs centaines d'années.
- Les fréquences de sortie des fichiers peuvent être horaires (3h, 6h), journalières et/ou mensuelles.

Variables

- Les champs peuvent être 2D (champ de surface) ou 3D (grille globale).
- Les champs sont intégrés temporellement suivant la fréquence de sortie des fichiers : valeurs instantanées, moyenne temporelle sur la période, valeurs minimum ou maximum...
- A chaque champ sont associées de nombreuses meta-données permettant sa description (titre, description, unité, axes associés,etc.…).
- Un même champ peut apparaître dans plusieurs fichiers (horaire, journalier, mensuel)

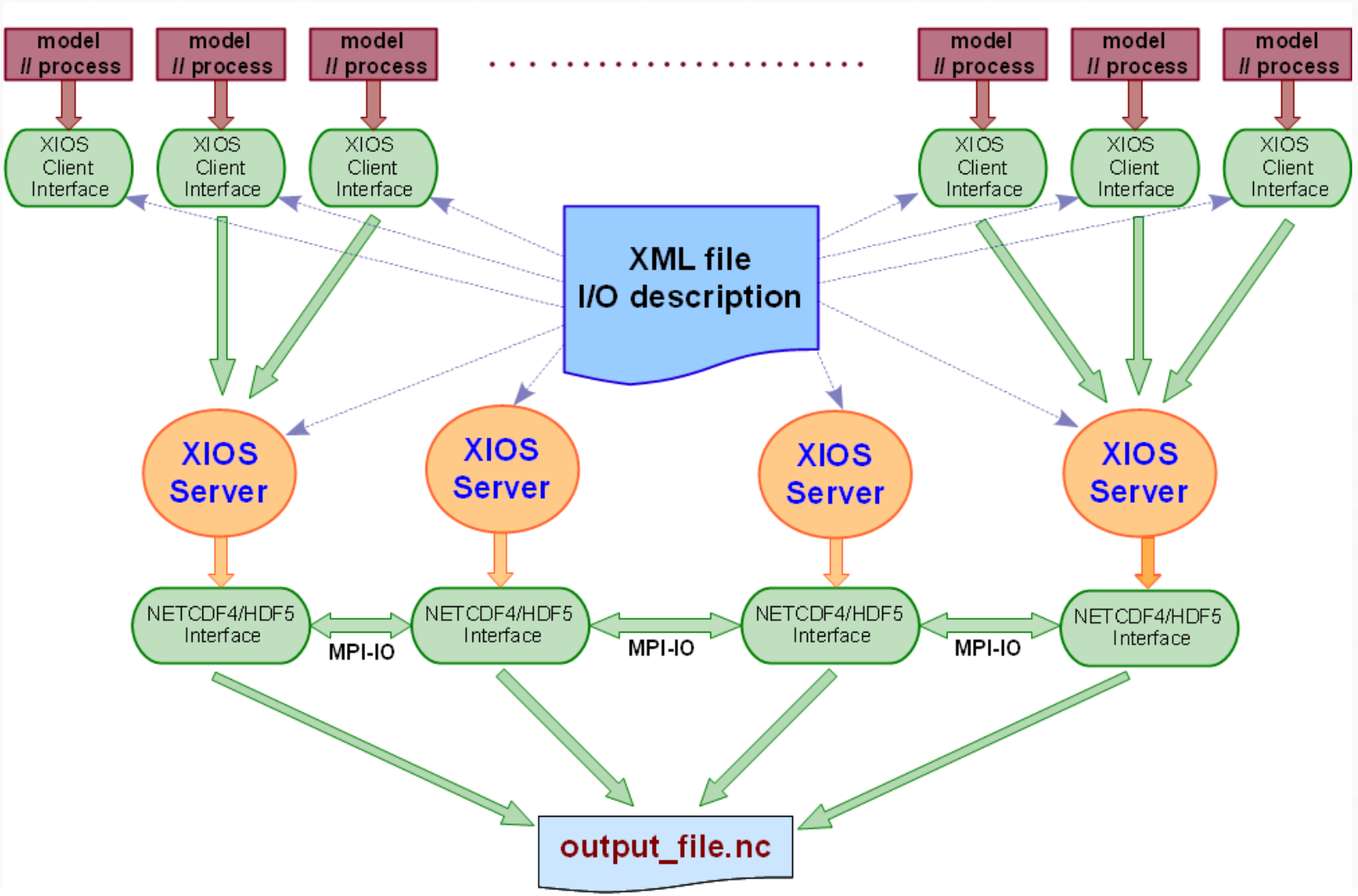
Approche actuelle : la bibliothèque IOIPSL

- Sortie des fichiers au format netcdf, écrite en fortan.
- Gestion des calendriers, des fichiers de redémarage et des sorties histoire.
- Gestion des opérations de moyennage temporel, minimum/maximum.

Très bon outils, mais souffre de quelques inconvénients :

- Manque de souplesse :
  - De (trop) nombreux paramètres lors des phases de définition et d'écriture due à la gestion des méta-données.
  - Nécessité de conserver de nombreux indices (handle) liés aux fichiers ou aux variables => concentration des appels I/O
  - Répétitions non nécessaires de nombreux paramètres
  - Nécessité de recompiler lors de chaque modification de paramètres I/O.
- Problèmes de performance
  - Aucune gestion du parallélisme ou du multi-threadisme.
  - 1 fichier par processus MPI, les fichiers doivent être reconstruits en post-traitement
  - De gros problèmes de performances lors du passage à l'échelle pour les sorties et les reconstructions des fichiers.

- La nouvelle approche : XIOS (XML-IO-SERVER)



2 principaux objectifs

Flexibilité, souplesse

- Externalisation de la description des I/O dans un fichier XML
  - Gestion hiérarchique avec concept d'héritage
  - Définitions plus simples et plus compactes
  - Évite les répétitions inutiles
- Simplification de la gestion des I/O au niveau du code
  - minimisation des appels liées au définitions des I/O
  - minimisation du nombre d'argument des appels
- Ecriture d'un champ : un identifiant et la donnée
  - `CALL send_field("field_id", field)`
- Permet de modifier la définition des I/O sans recompiler
  - Tout est dynamique, le fichier XML est analysé à l'exécution

XIOS

Performance

- L'écriture des données ne doit pas impacter l'exécution du code.
- Utilisation d'un ou plusieurs serveurs exclusivement dédiés au I/O
  - Transfert asynchrone des données des clients vers les serveurs.
  - Ecriture indépendante et asynchrone des données par chaque serveur.
- Utilisation des systèmes de fichiers parallèles via netcdf4/hdf5 => MPI_IO.
  - Écritures simultanées de plusieurs processus dans un même fichier
  - Plus de phase de reconstruction en post-traitement.

Historique du développement

Fin 2009 : Démonstration de faisabilité : XMLIO/SERVER

- Totalement écrit en fortran 90
- Implémente les fonctionnalités XML et client/serveur

Mi-2010 – fin 2011 : réécriture complète en C++ => XIOS

- Projet Européen IS-ENES.
- Programmation orientée objet.
  - interopérabilité C++/C/Fortran à travers la norme Fortran 2003
- 25000 ligne de codes sous SVN.

Description XML - « Hello Word »

```xml
<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="output_file">
        <field field_ref="field_A" />
      </file>
    </file_definition>

  </context>
</simulation>
```

Interfaçage fortran

- L'arborescence XML peut être crée ou complétée grâce à l'API fortran 2003
  - ex: ajout d'attributs au champ « toce »

```
CALL xios_set_field_attribut(id="toce",long_name="Temperature", unit="deg C", enabled=".TRUE.")
```

  - Les champs sont envoyés à chaque pas de temps

```
CALL xios_send_field(id="field_A",field_A)
```

Héritage

Principe d'héritage

- Héritage parent-enfant via les groupes

```
<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="ocean vertical velocity"            unit="m/s" />
      <field id="woce_eff"   long_name="effective ocean vertical velocity"  unit="m/s" />
    </field_group>

    <field id="aht2d"        long_name="lateral eddy diffusivity"           unit="m2/s" />

  </field_group>
</field_definition>
```

- Héritage par référence

```
<field id="toce" long_name="temperature (Celcius)" unit="degC" grid_ref="Grid_T" />
<field id="toce_K"  field_ref="toce" long_name="temperature (Kelvin)"  unit="degK" />
```

Fonctionnalités Client/Serveur

- "Serveur" XIOS :
  - Pool de processus MPI dédiés aux I/O
  - Les "clients" sont les processus MPI des codes de calcul

- Chaque code client communique avec les serveurs via la notion de "context"
  - Chaque pool client dispose de son propre communicateur MPI.
  - Un inter-communicateur MPI est crée entre le pool de client le pool serveur.
  - A chaque inter-communicateur est associé un "context"

- Véritable notion de "service" MPI
  - L'enregistrement est dynamique

```
CALL xios_context_initialize("context_id", comm)
```

  - Un même pool de serveur peut gérer plusieurs pools de clients

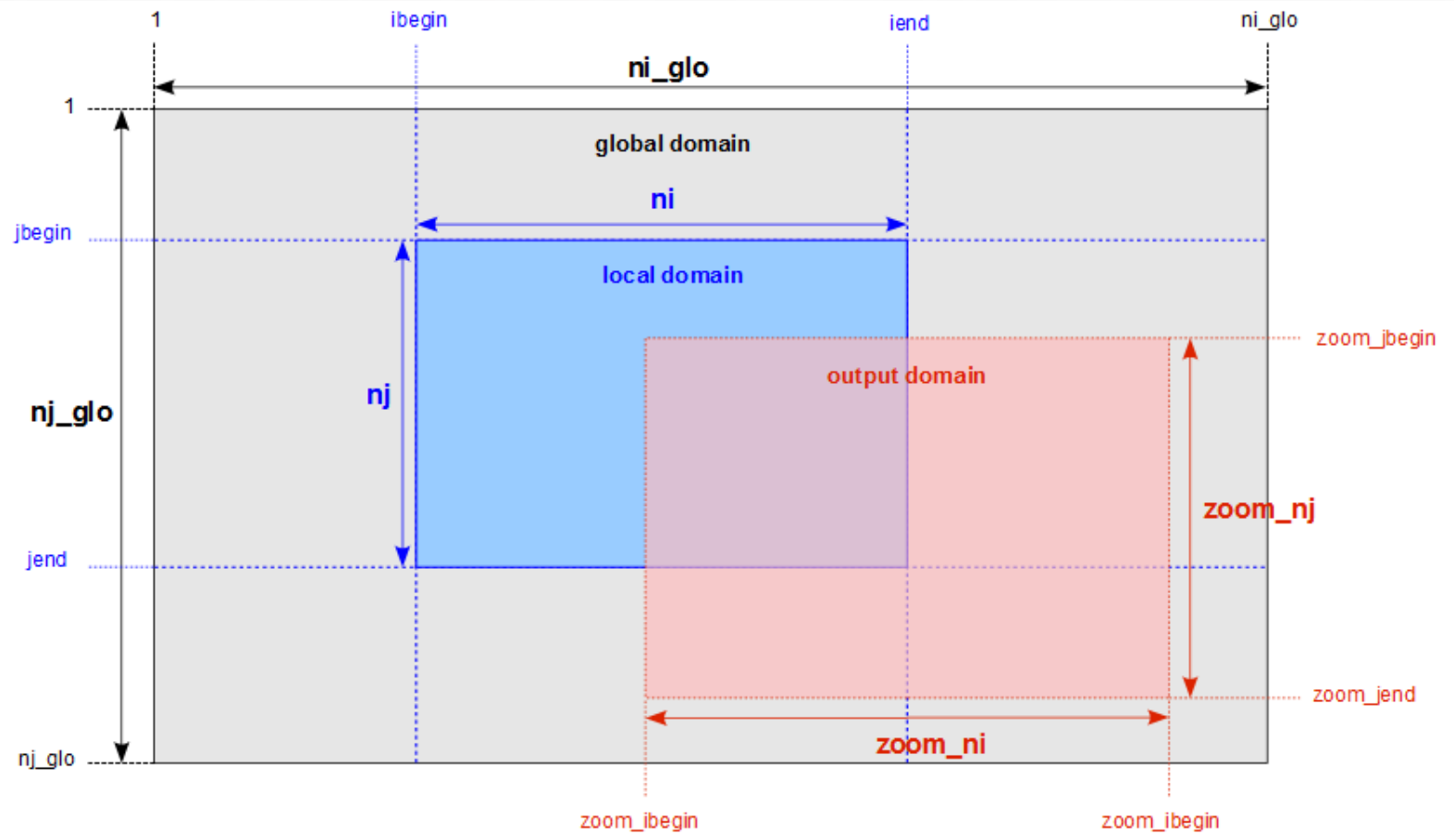
- Idée maîtresse : les codes de calculs ne doivent pas être impactés par les IOs
  - Communications point à point synchrone non bloquante entre clients et serveurs
  - MPI_ISSend, MPI_IRecv, MPI_Test, MPI_IProbe...
  - Permet le recouvrement Calcul/Communication/Écritures
  - Utilisation de buffers pour lisser les pics d'IO

Fonctionnalités Client/Serveur

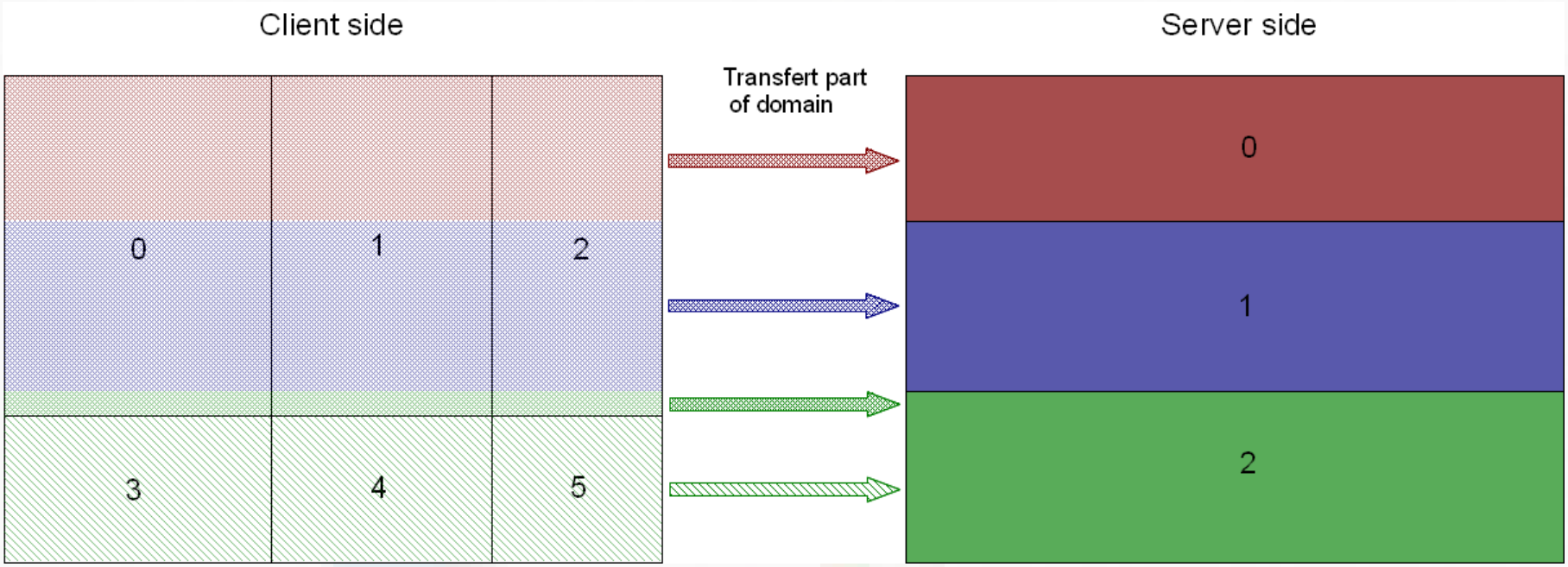
- Les communications entre clients et serveurs utilise des principes de programmation RPC (Remote Proceduring CALL) à travers MPI
  - Un message est auto-descriptif.
  - Un message est rempli coté client en empaquetant les arguments et les données.
  - Quand le message est reçu coté serveur, l'en-tête est analysé et le message est acheminée à destination
  - Le message est ensuite dépaqueté puis la méthode correspondante est appelée.

Coté client : Distribution des données

- Domaine global : ni_glo x nj_glo
- Domaine local : ni x nj



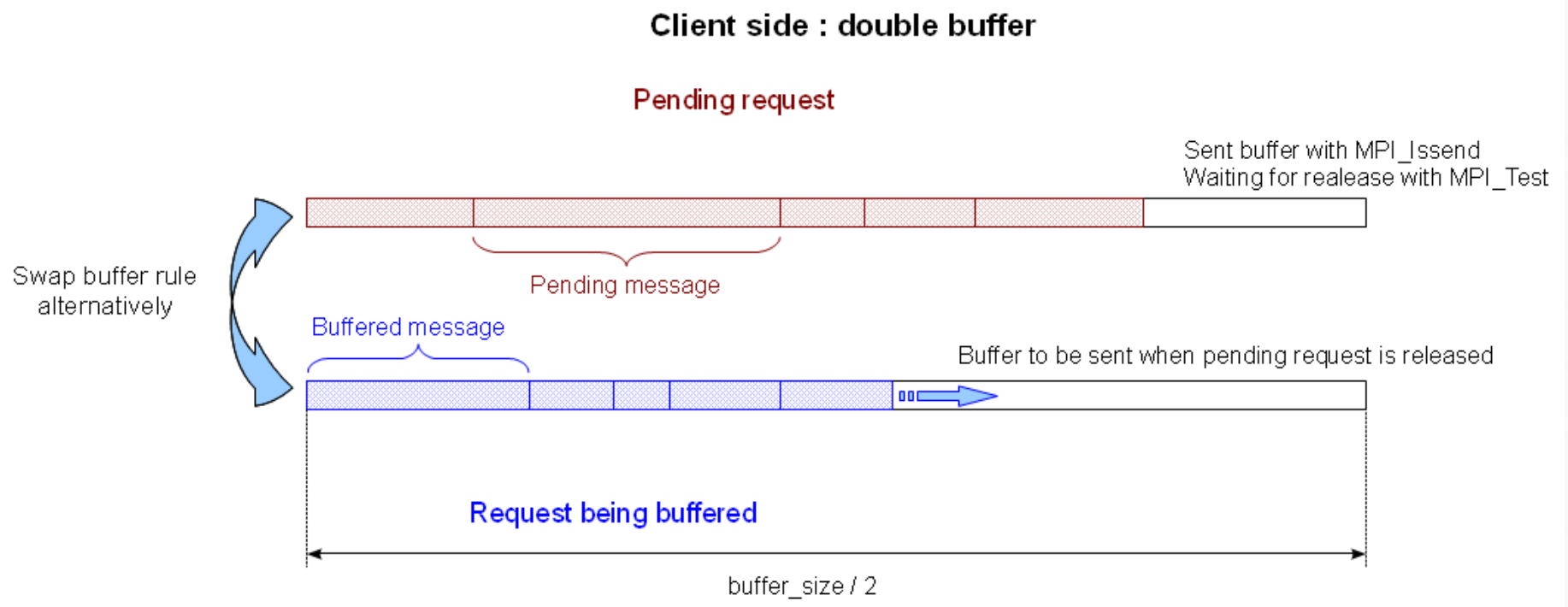
Coté serveur



- Les clients {0, 1, 2} envoient leur part de domaine aux serveurs {0, 1, 2}
- Les clients {3, 4, 5} envoient leur part de domaine au serveur 2

- La distribution des données coté serveur est équi-répartie suivant la seconde dimension.
- Un client peut communiquer avec plusieurs serveurs.
- Un serveur peut recevoir des données de plusieurs clients.

Coté client : double buffers



- Les messages sont concaténés puis envoyer en une seule une requête MPI lorsque la précédente requête est reçue.
- Communications non bloquantes : MPI_Issend/MPI_Test.
- Lorsque les buffers sont pleins => on rentre en mode bloquant

Protocole de communication

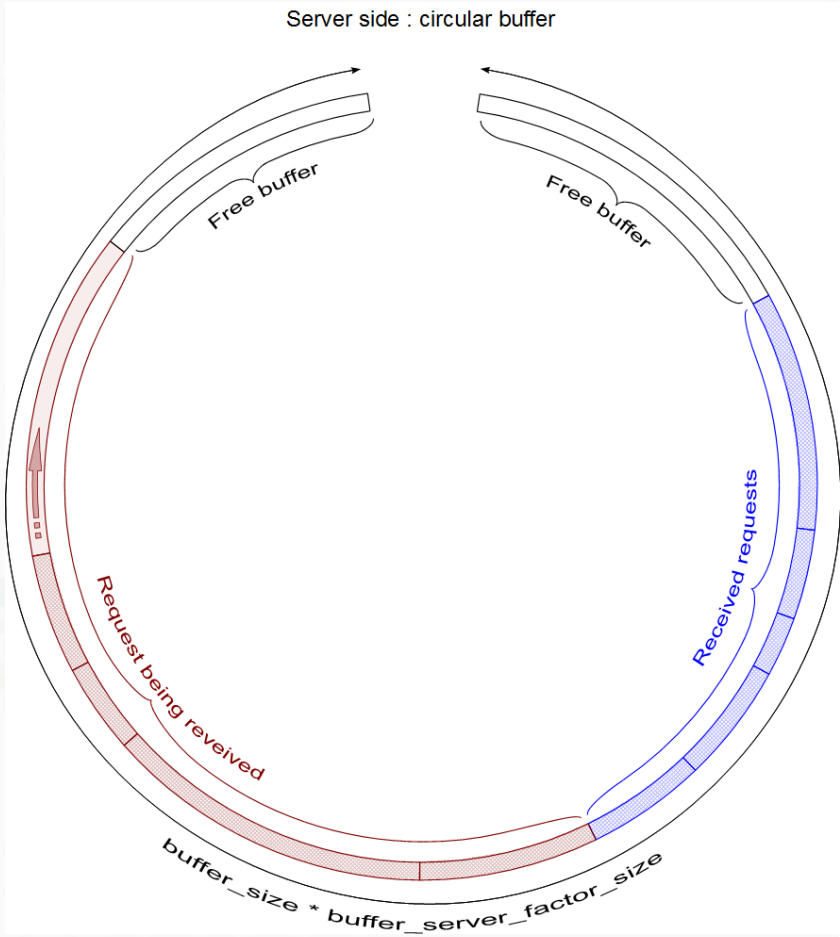
Coté serveur : buffer circulaire

- Arrivé d'un message => MPI_Iprobe
- Réception : MPI_Irecv / MPI_Test

- On reçoit en priorité les messages afin de libérer les buffers coté client.

- On traite les messages et effectue les écritures lorsque qu'il n'y a plus de requête en attente de réception.

- Lorsque les buffers sont pleins, on passe en mode bloquant :
  - traitement des messages pour libérer les buffers
  - Ajout de serveur XIOS pour alléger la charge.



Sortie au format NETCDF4/HDF5

- Utilisation des écritures parallèle via HDF5//
  - MPI/IO en bout de chaine

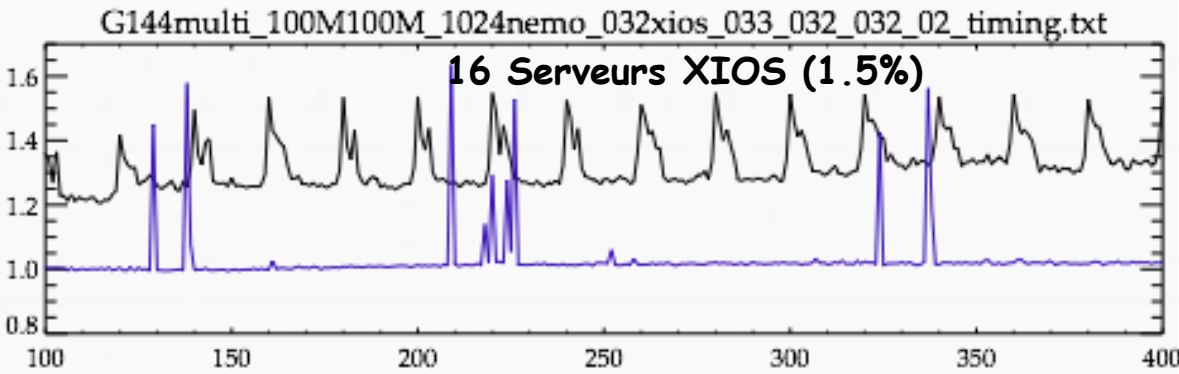
2 options possibles : "multiple_file" ou "one_file"

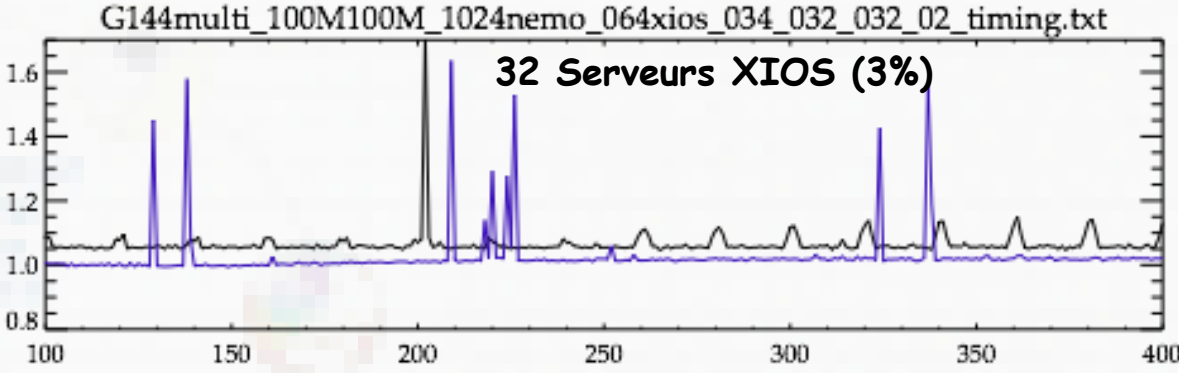
- multiple_file : un fichier par serveur
  - Pas d'écriture parallèle
  - Phase de reconstruction nécessaire en post-traitement
- one_file : un fichier unique
  - Utilisation des écritures parallèles, accès indépendants ou collectifs.

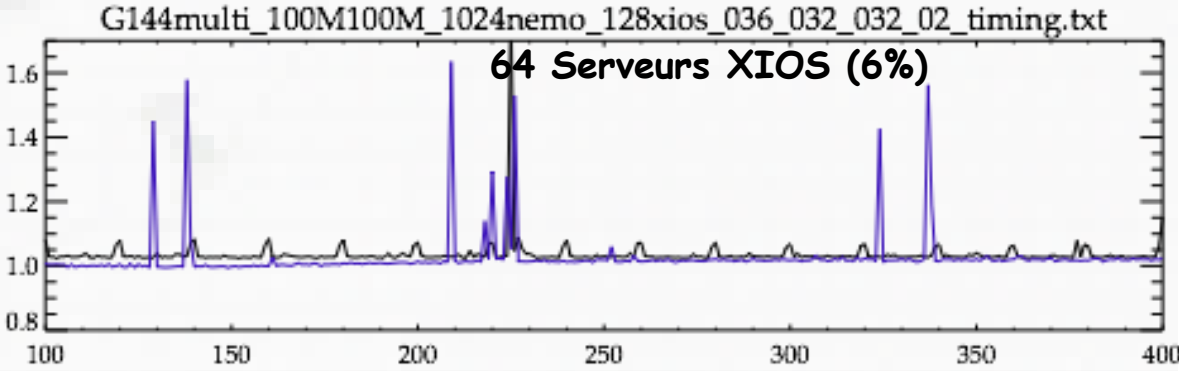
Cas test : XIOS/NEMO

- Configuration GYRE 144 (4322x2882)
  - pdt=180s, 720 pdt (36 h)
  - NEMO : 1024 proc. MPI
- Temps par itération
  - **Sortie horaire**
  - **Sans IO**
  - 246 Go écrit en 4 fichiers

```
62G Feb 25 04:36 BIG1h_st32_1h_grid_T.nc
28G Feb 25 04:37 BIG1h_st32_1h_grid_U.nc
26G Feb 25 04:31 BIG1h_st32_1h_grid_V.nc
130G Feb 25 04:35 BIG1h_st32_1h_grid_W.nc
```

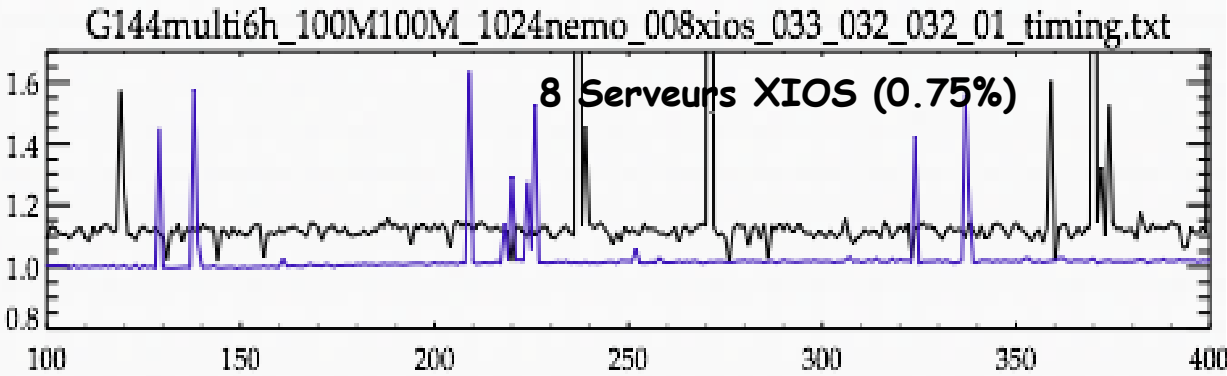


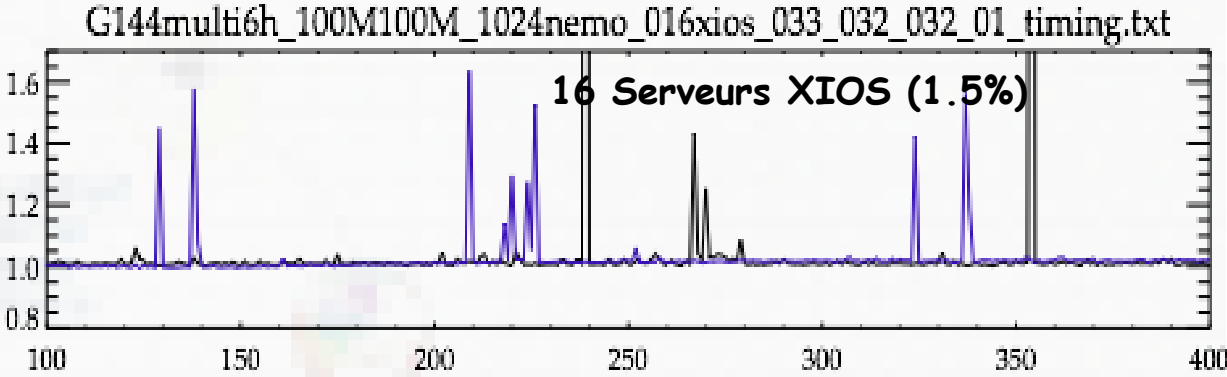


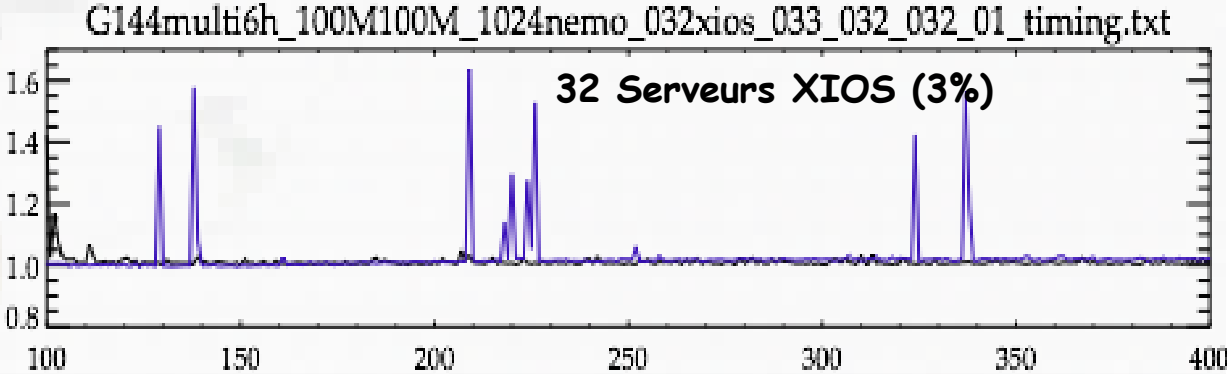


Cas test : XIOS/NEMO

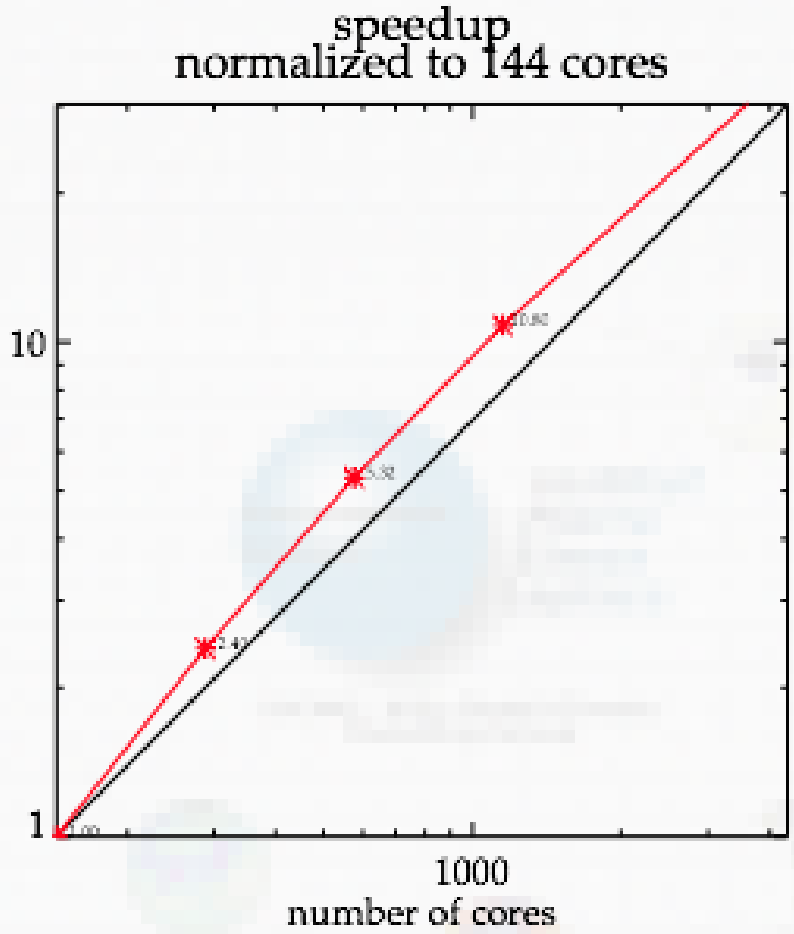
- Cas test plus léger
  - Sortie à 6h
  - Sans IO

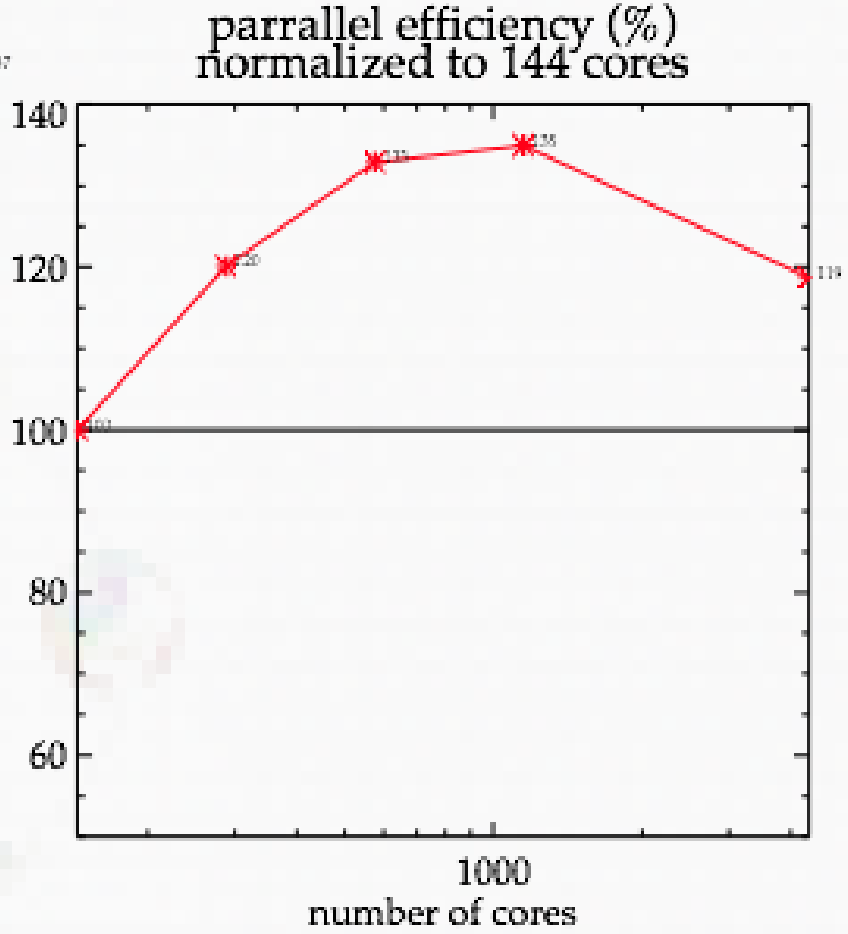






CURIE Fat Nodes: NEMO 3_4_b GYRE Big IO multi_file, jp_cfg = 144





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