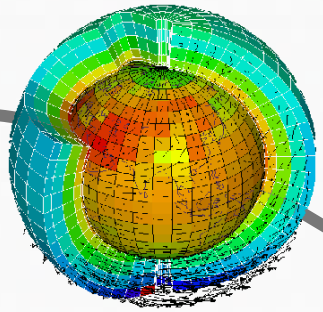
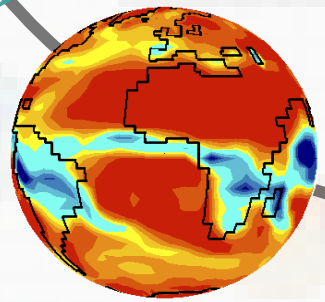
$$\frac{\partial \mathbf{U}}{\partial t} + 2\Omega \times \mathbf{U} = .$$

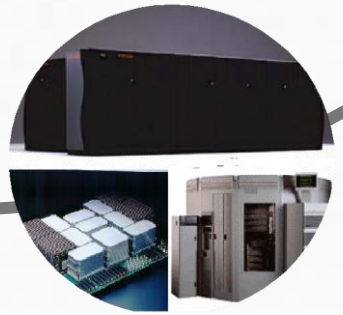


Gestion des IO parallèles en modélisation du climat

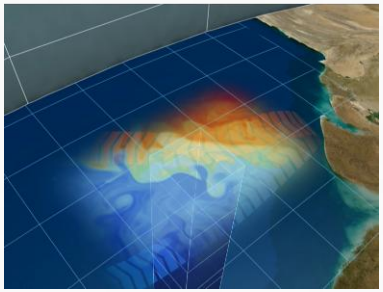
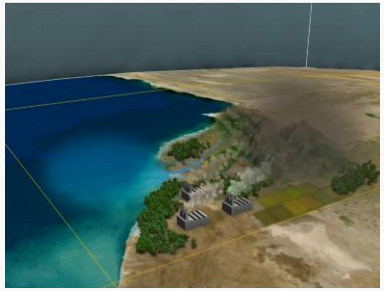
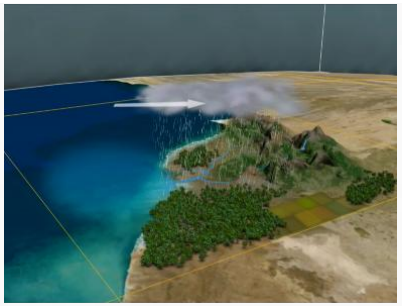
La bibliothèque XIOS

```
...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
...ER*30 :: timenow
```





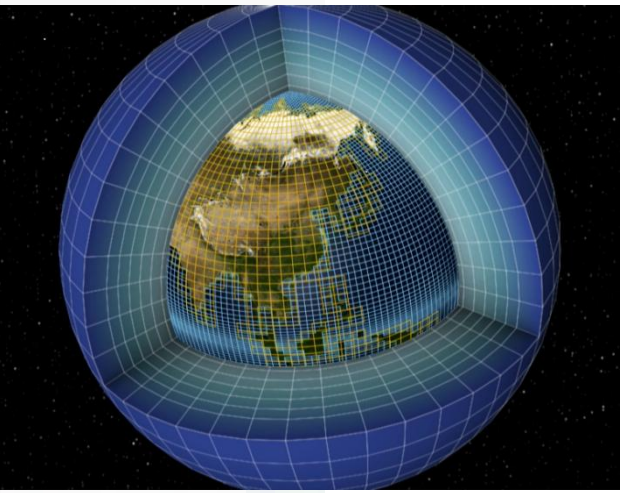
Le modèle couplé "Système Terre" de l'IPSL

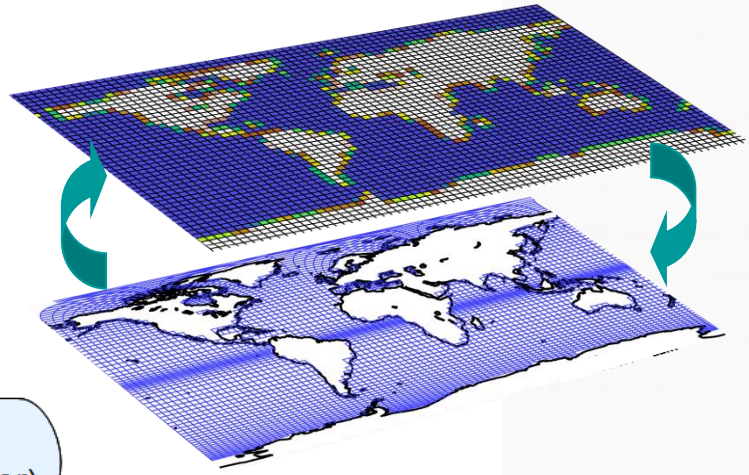
INCA / REPROBUS
(chimie atmosphérique)
(aérosol)

ORCHIDEE
(surfaces continentales)
(végétation)

LMDZ
(atmosphère)



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/TGCC)
  - 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)

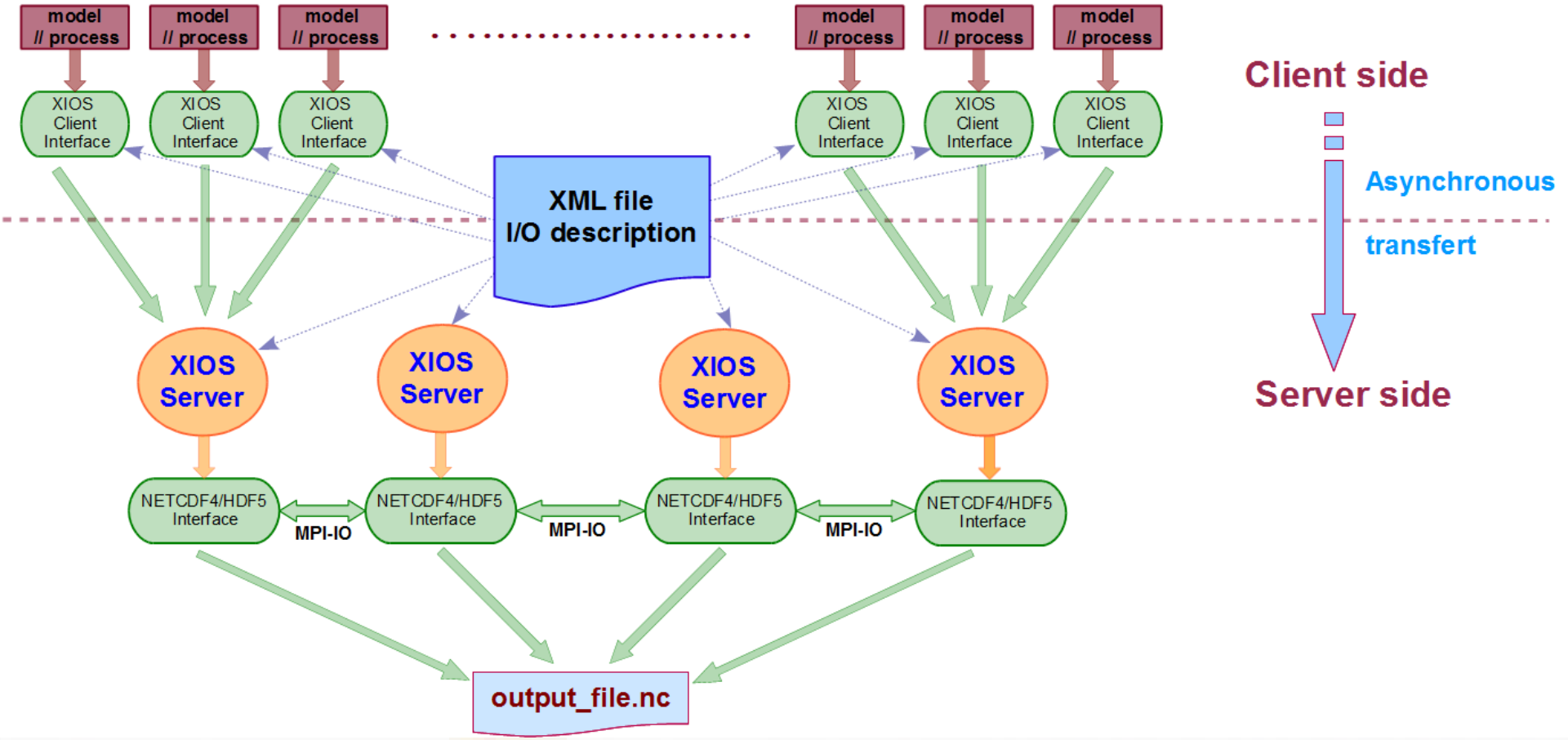
Approche actuelle : la bibliothèque IOIPSL

- Sortie des fichiers au format netcdf, écrite en fortan.
- Gestion des calendriers, des fichiers de redémarrage et des sorties histoires.
- 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 et 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 liés aux définitions des I/O.
  - Minimisation du nombre d'arguments des appels.
- Écriture d'un champ : un identifiant et la donnée
  - `CALL xios_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.
    - Écriture 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.
    - Mais utilise toujours IOIPSL comme couche de sortie.
  - Mi-2010 – fin 2011 : réécriture complète en C++ => XIOS
    - Projet Européen IS-ENES.
    - Programmation orientée objet.
      - 35000 lignes de codes sous SVN.
  - Mi-2012... : passage en production

Description XML - « Hello Word »

```
<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

- Interfaçage Fortran/C/C++ à travers l'interopérabilité Fortran/C de la norme 2003
- Le code fortran pilote l'initialisation et la finalisation de XIOS.
- L'arborescence XML peut être crée ou complétée grâce à l'API fortran.
  - ex : ajouter le champ "toce" dans l'arborescence XML

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

  - ex: ajout d'attributs au champ « toce ».

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

- Le code pilote la boucle temporelle pendant laquelle les champs sont envoyés à chaque pas de temps :

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

"Hello World" - Fortran

```
PROGRAM test_cs
IMPLICIT NONE
  INCLUDE "mpif.h"
  INTEGER :: rank
  INTEGER :: size
  INTEGER :: ierr

  CALL MPI_INIT(ierr)
  CALL MPI_COMM_RANK(MPI_COMM_WORLD,rank,ierr)
  CALL MPI_COMM_SIZE(MPI_COMM_WORLD,size,ierr)

  IF (rank<8) THEN
   CALL client(rank,8)
  ELSE
    CALL server
  ENDIF

  CALL MPI_FINALIZE(ierr)
END PROGRAM test_cs

SUBROUTINE server
  USE xios
  IMPLICIT NONE
    CALL xios_init_server
END SUBROUTINE server
```

"Hello World" - Fortran

```
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")

  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()
```

"Hello World" - Fortran

```
! 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 : généralité

- Context : `<context />`
  - Isole les définitions des différents codes
    - Pas d'interférence possible
    - 2 identifiants identiques peuvent être réutilisés

- Calendrier et durée
  - XIOS peut gérer plusieurs calendriers via l'attribut de context : "`calendar_type`"
    - `Gregorian, D360, NoLeap, AllLeap, Julian`
    - Format : "`2012-02-27 15:30:00`"
  - XIOS gère plusieurs unités de temps pour les durées
    - year : y, month : mo, day : d, hour : h, minute : mi, second : s
    - ex : "`1mo 2d 1.5h 30s`"

- 7 familles d'éléments
  - "context", "axis", "domain", "grid", "field", "file" et "variable"
  - Regroupée sous forme hiérachique :
    - Racine : ex : <field_definition> ... </field_defintion>
    - Groupe : ex : <field_group> ... <field_group>
    - Élément final : ex : <field> ... </field>

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" />
```

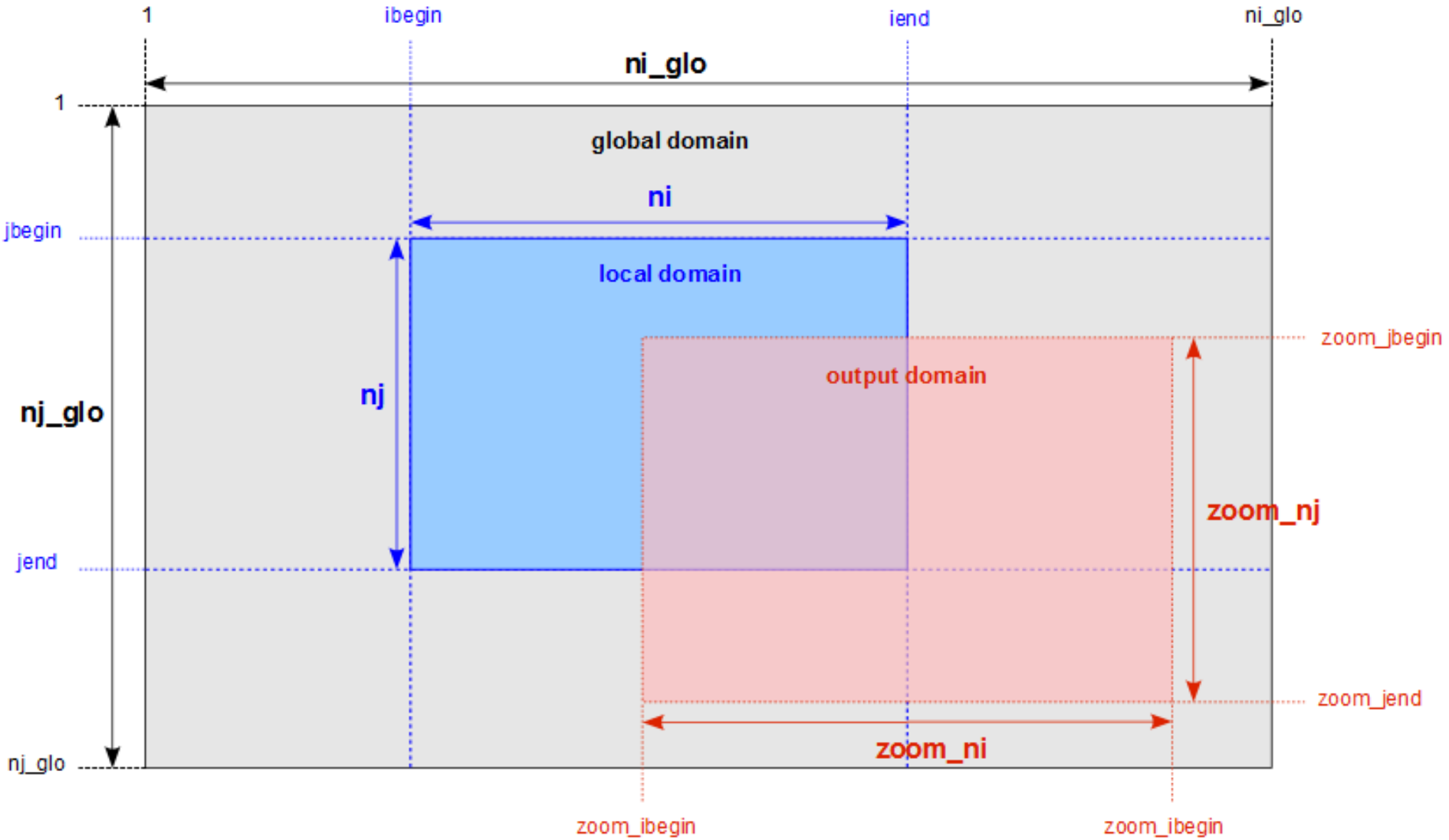
Définition des grilles

- Grille : <grid/>
  - Seules les grilles cartésiennes ou curvilinéaires sont gérées actuellement par XIOS
  - Définie par l'association d'un domaine horizontal et (option) d'un axe vertical
    - `<grid id="grid_A" domain_ref="domain_A"  axis_ref="axis_A" />`

- Axe vertical : <axis />
  - Attribut : size, value et unit.

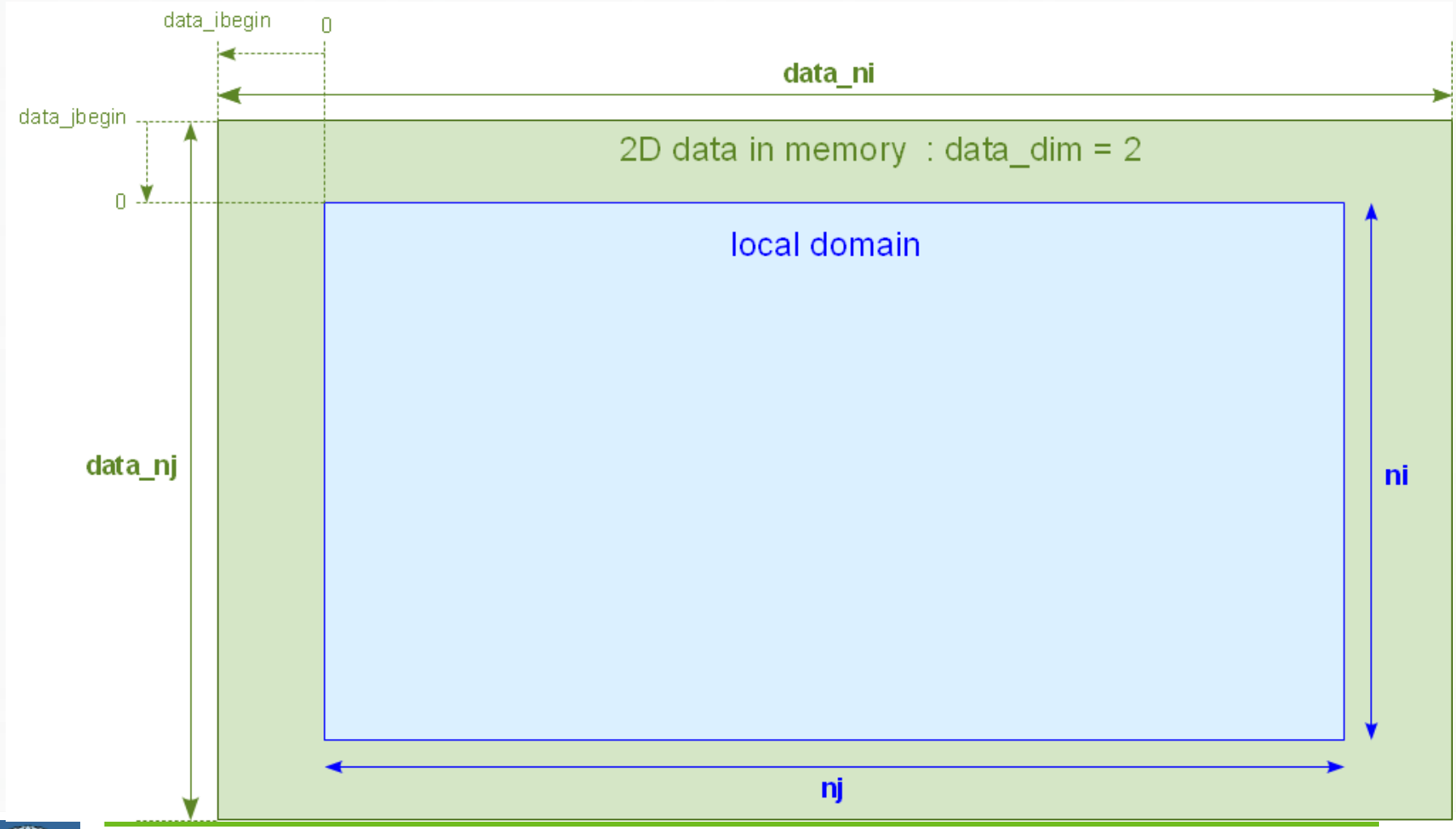
- Domaine horizontal : <domain />
  - Découpé et distribué sur les différents processus.
  - Domaine global : ni_glo x nj_glo
    - ni_glo, zoom_ibegin, zoom_ni,
    - nj_glo, zoom_jbegin, zoom_nj
  - Domaine local : ni x nj
    - **ibegin, ni, [iend]**
    - **jbegin, nj, [jend]**

Définition des grilles



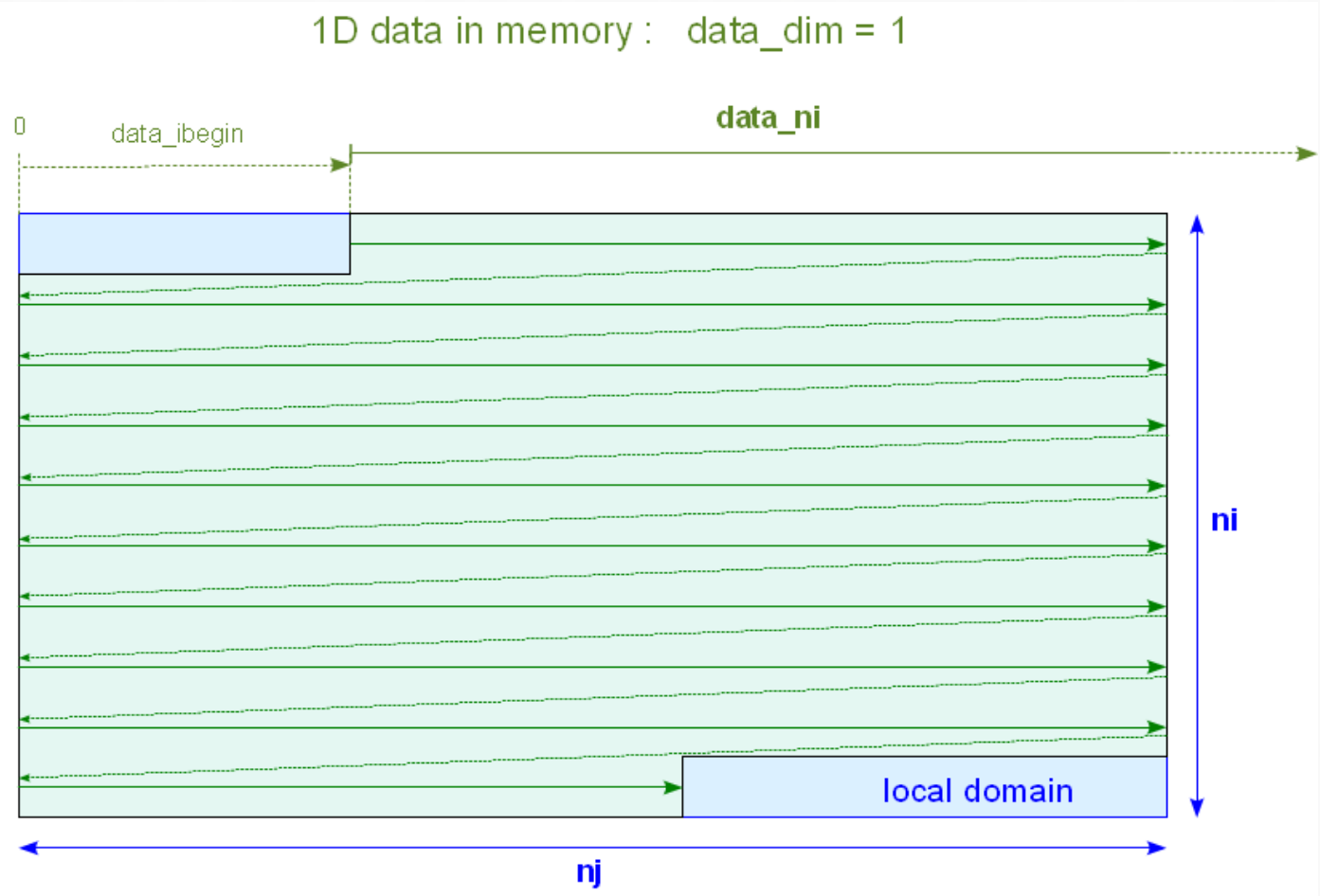
Définition des grilles

- Description des données en mémoire et extraction :
  - Gestion des halos de recouvrement
  - Attributs : data_dim, data_ibegin, data_ni, data_j_begin, data_nj



Définition des grilles

- Grille 1D en mémoire (vecteurs de points)



- Grilles indexées (compressées) : ex : grilles en points de terre
  - data_nindex, data_i_index, data_j_index

Définition des champs

Champ : <field />

- La grille associée définie les dimensions du champs
  - Attribut : grid_ref
  - Dimension définie conformement à la grille : dimension : data_ni x data_nj x vertical_size
- Un champ peut être envoyé à chaque pas de temps à travers l'interface fortran
  - `CALL xios_send_field("field_id",field)`

- Des opérations temporelles (moyennes, min, max...) peuvent être effectuées à partir des valeurs de chaque pas de temps.
  - Attribut "operation" : once, instant, average, minimum, maximum

- Un champ est associé à un ou plusieurs fichiers dans lequel il sera sorti à la fréquence du fichier :

```
<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>
```

Opérations entre champs

- XIOS permet d'effectuer des opérations sur les champs "à la volée"
  - Ex : Conversion d'unité

```
<field name="T_Celcius" unit="C"> T_kelvin-273.15 </field>
```

  - De manière plus générale toutes combinaisons de champs, incluant les fonctions arithmétiques classiques :

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

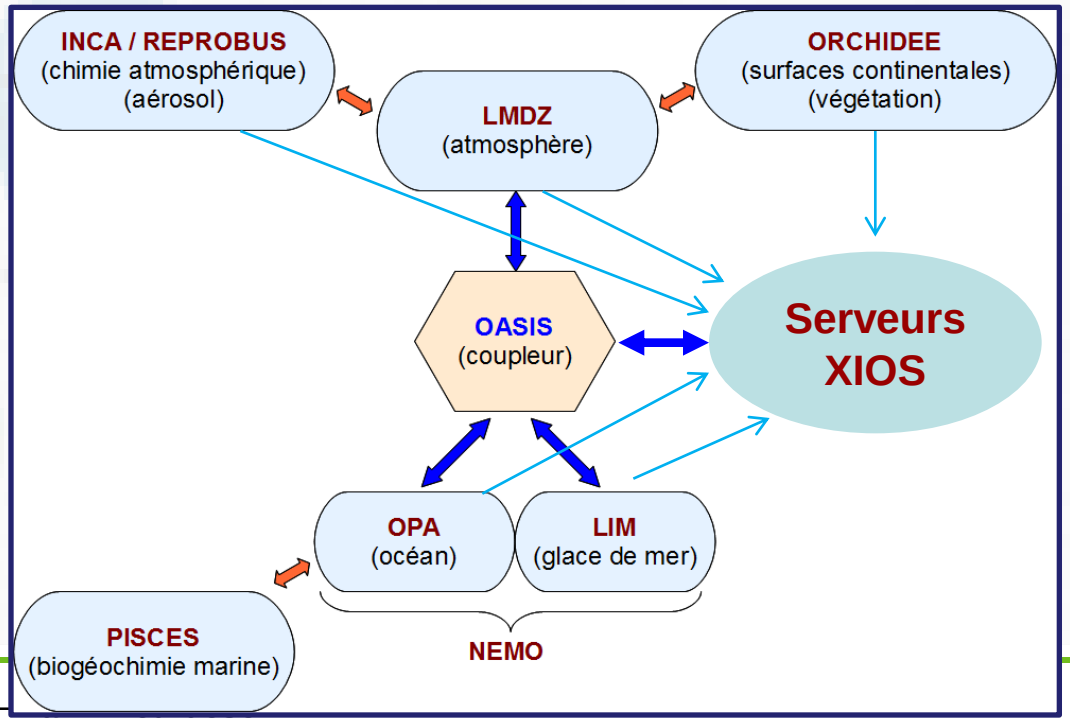
  - Fonctionne également avec les opérations temporelles : préfix : @
    - ex : sortie de la moyenne mensuelle du maximum journalier de la température

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

Fonctionnalités Client-Serveur
Couche de transport

Fonctionnalités Client/Serveur

- "Serveurs" XIOS :
  - Pool de processus MPI dédiés aux I/O.
  - Les "clients" sont les processus MPI des codes de calcul.
- Idée maîtresse : les codes de calculs ne doivent pas être impactés par les IOs
  - Communications point à point synchrones non bloquantes 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'I/O.
  - Rassemblement des données sur quelques serveurs => moins d'accès disque



Fonctionnalités Client/Serveur

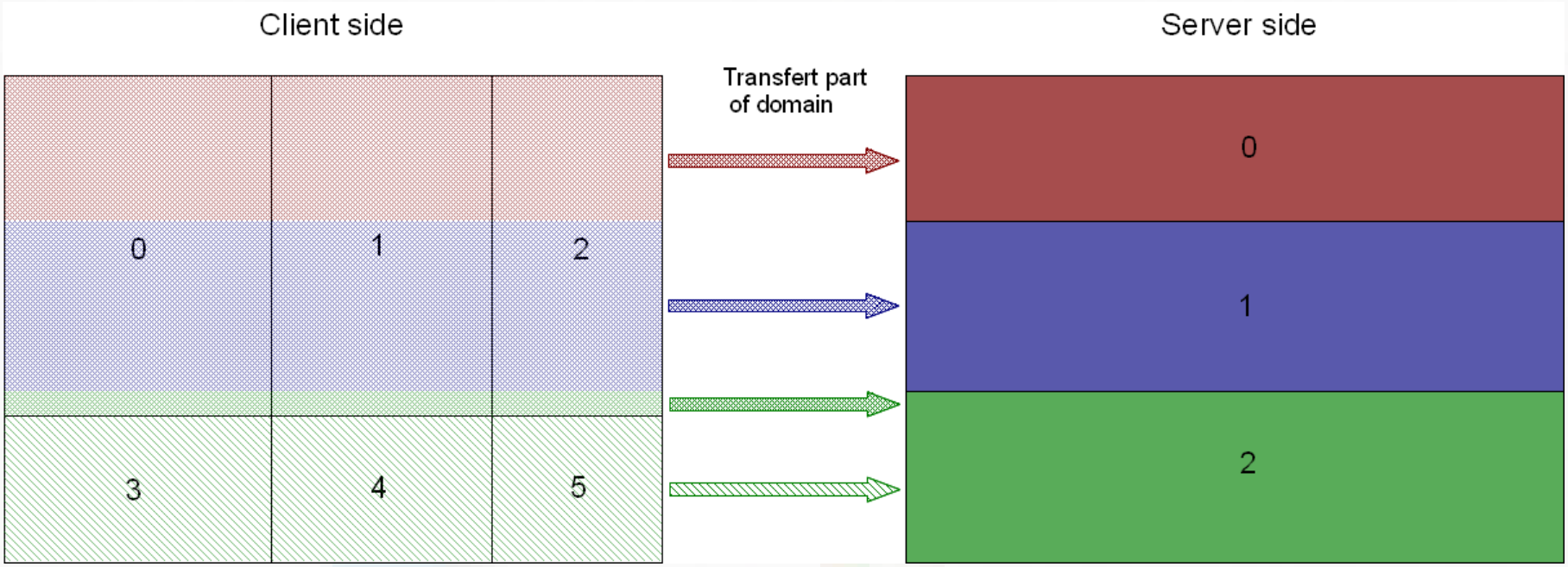
- XIOS est conçu pour fonctionner avec des modèles couplés
  - Un pool de serveur pour l'ensemble des codes.
  - Véritable notion de "service" MPI

- "Splitting" des communicateurs
  - Les clients et les serveurs XIOS on besoin de leur propre communicateur
  - Le communicateur global (MPI_COMM_WORLD) peut être "splitté" soit par XIOS, soit par le coupleur OASIS (si présent).
  - Effectué durant l'initialisation de XIOS, chaque code est identifié par un Id unique.
  - `CALL xios_initialise("code_id", return_comm)`
  - Un communicateur est retourné.
  - Les serveurs sont initialisés et écoute pour l'enregistrement des contexts.

- 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 clients et le pool de serveurs.
  - A chaque inter-communicateur est associé un "context".
  - Chaque processus client peut gérer plusieurs context.
  - L'enregistrement est dynamique et peut être réalisé à tout moment.

    `CALL xios_context_initialize("context_id", comm)`
  - Les messages peuvent ensuite être acheminés vers le bon contexte.

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.

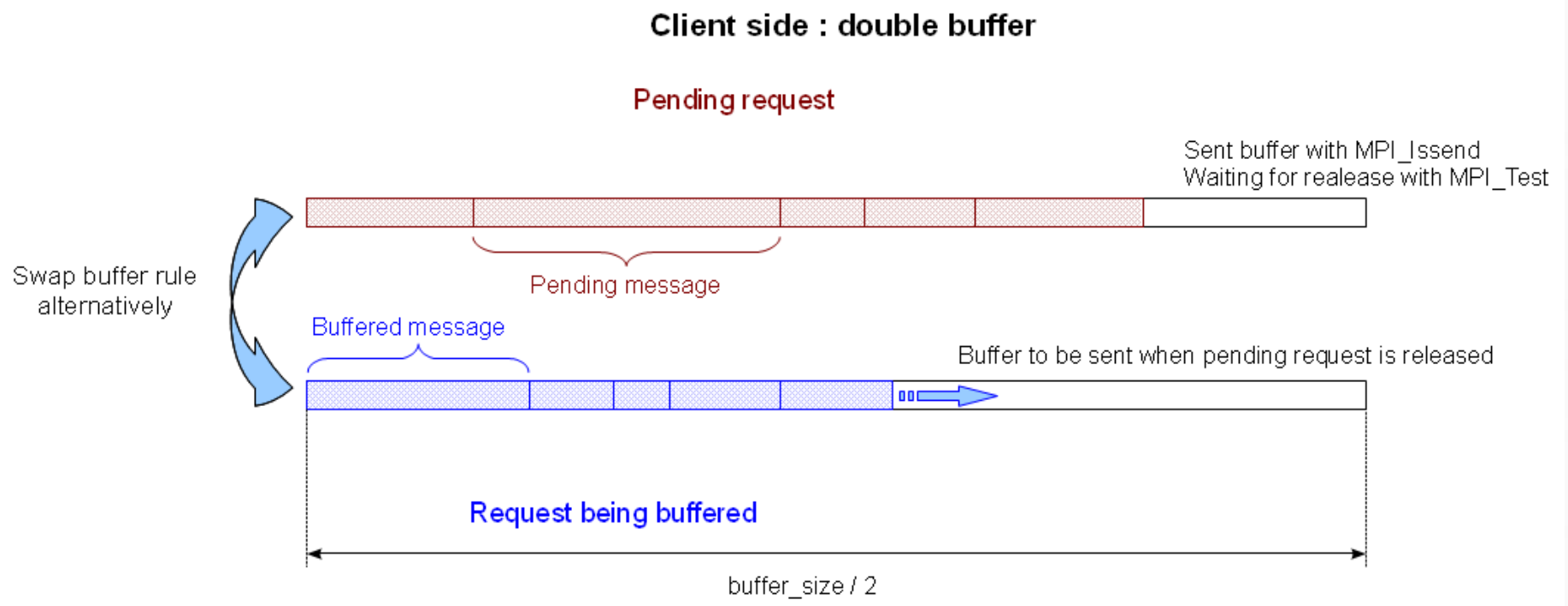
- La distribution des données coté serveur est équi-répartie suivant la seconde dimension.
- Un client peut communiquer avec plusieurs serveurs.
- Un serveur peut recevoir des données de plusieurs clients.

Protocole de communication

- 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 analysée et le message est acheminé à destination.
  - Le message est ensuite dépaqueté puis la méthode correspondante est appelée.

- Zoologie
  - **Message** : concaténation dans un buffer de différents arguments d'appel.
  - **Requête** : concaténation of plusieurs messages qui seront envoyés/reçus à travers la couche MPI par des appels asynchrones.
  - **Evènement** : Ensemble de plusieurs messages de différents clients mais définissant une action collective.
    - Comme les messages peuvent être reçu dans le désordre, un message d'un même évènement est identifié par un même unique identifiant : "timelineId" .
    - Après réception, les évènements sont triés suivant leur "timelineId" puis, s'il sont complets, acheminés vers les méthodes correspondantes et exécutés.

Coté client : double buffers



- Les messages sont concaténés puis envoyer en une seule 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

Protocole coté serveur

1- Boucle sur les contexts enregistrés

2- Boucle sur les clients des contexts

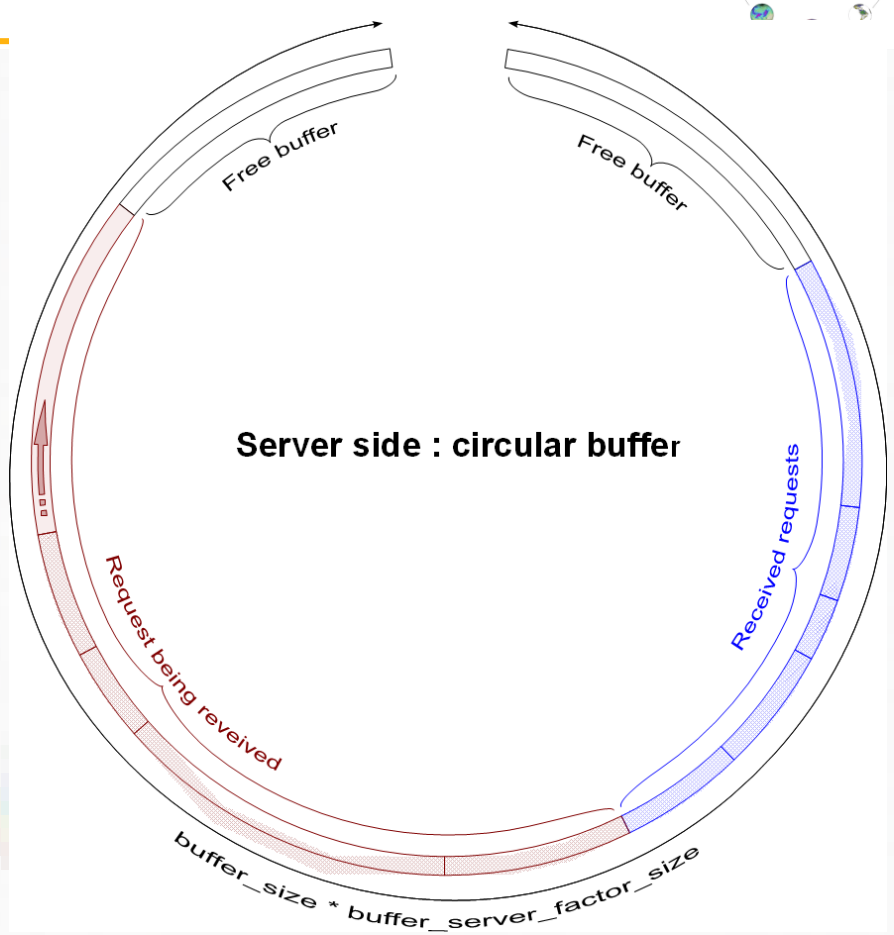
3- Une requête est en cours de réception ?
- passe au prochain client

3- Sinon vérifie si un message est disponible
- Utilise la fonction asynchrone MPI_Iprobe
- sinon passe au prochain client

3- Si oui, vérifie la taille du buffer
- Si le buffer est plein, passe au prochain client

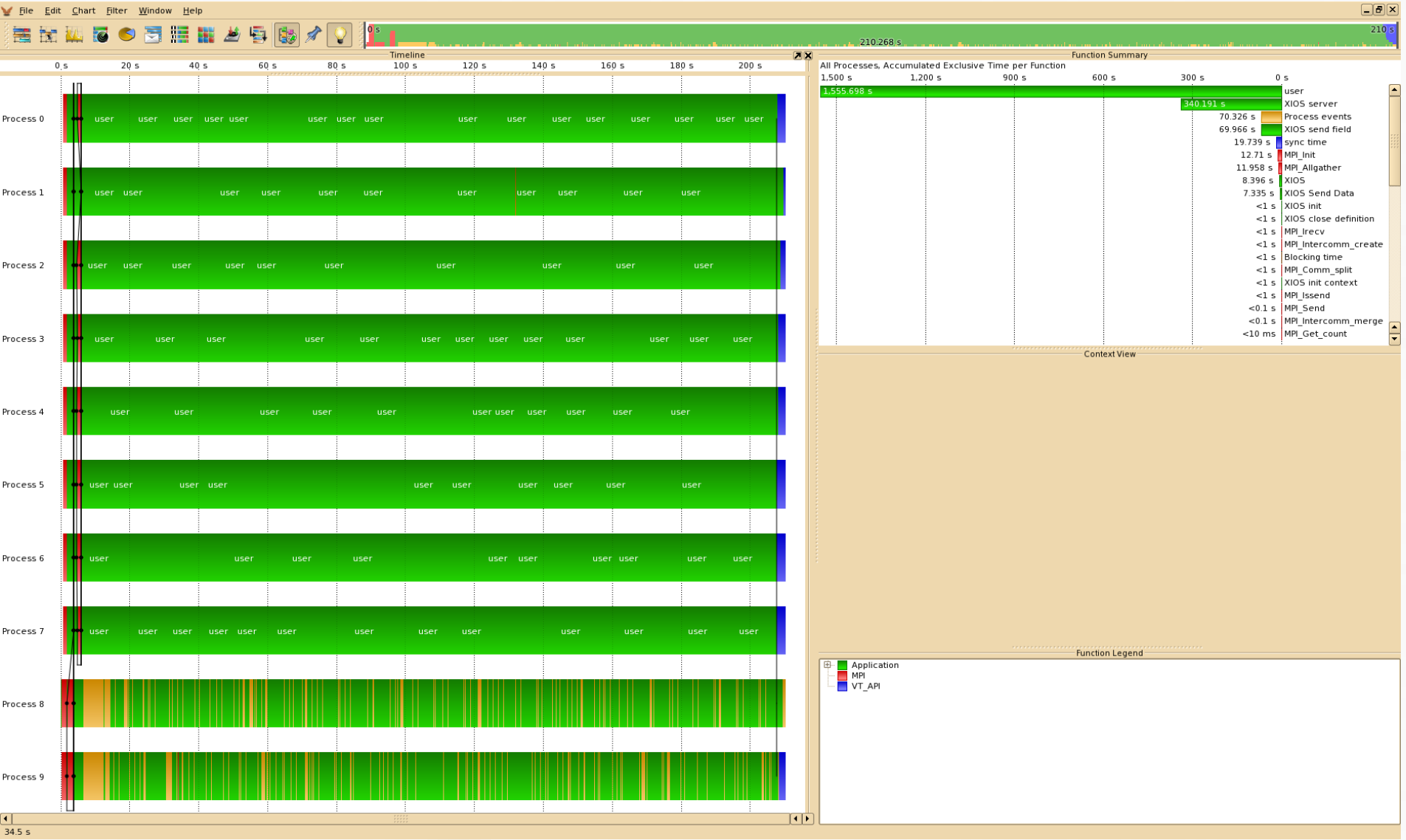
3- Si il y a assez de place, reçois la requète
- utilise la fonction asynchrone MPI_IRECV

3- Prochain client... => goto 2

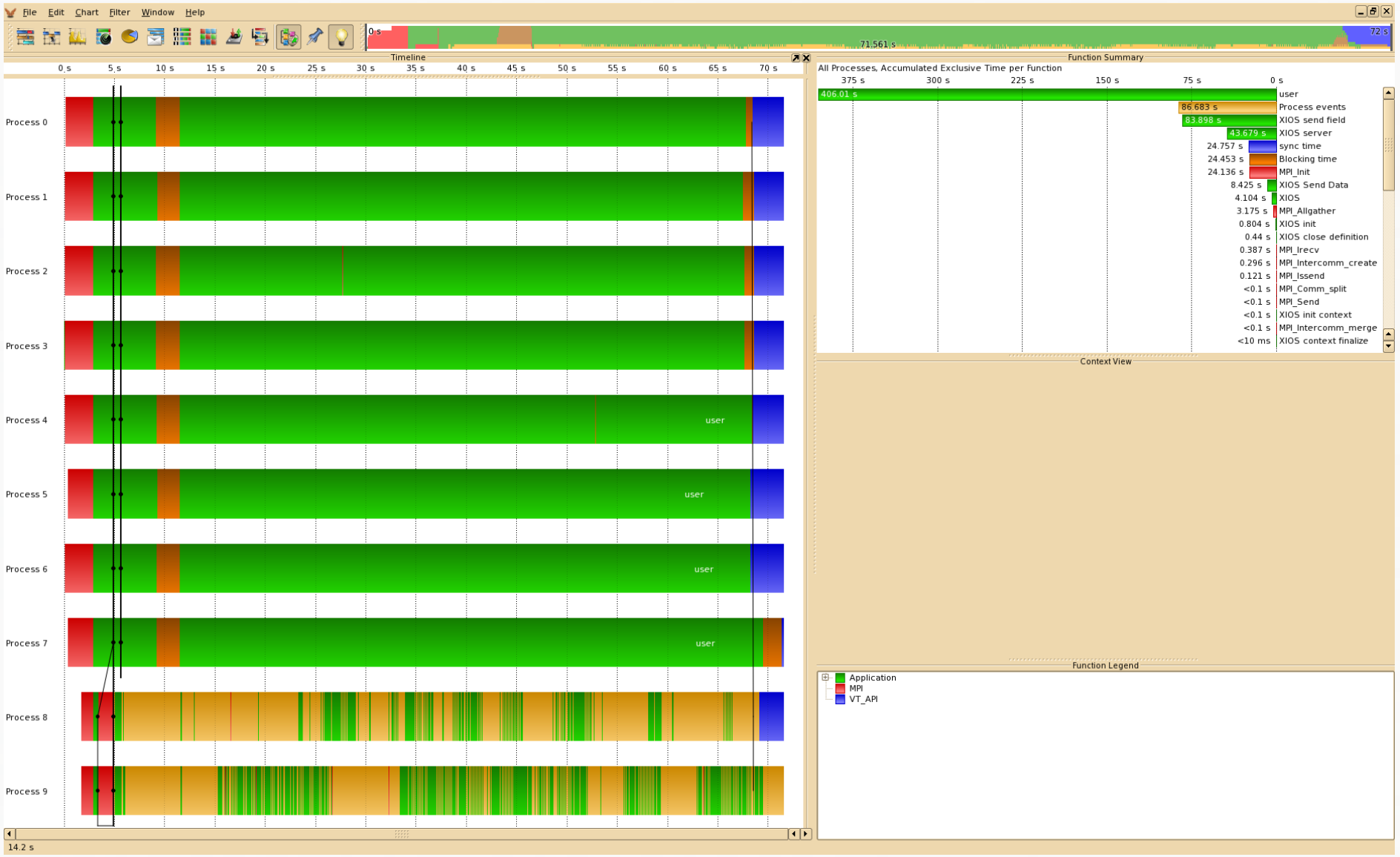


2- Si plus aucune requète n'est disponible, alors vérifie les évènements reçus.
- Vérifie si le prochain évènement identifié par son "timelineId" est complet.
- Achemine l'évènement vers la méthode cible puis l'exécute
- Libère l'espace mémoire correspondant dans le buffer.

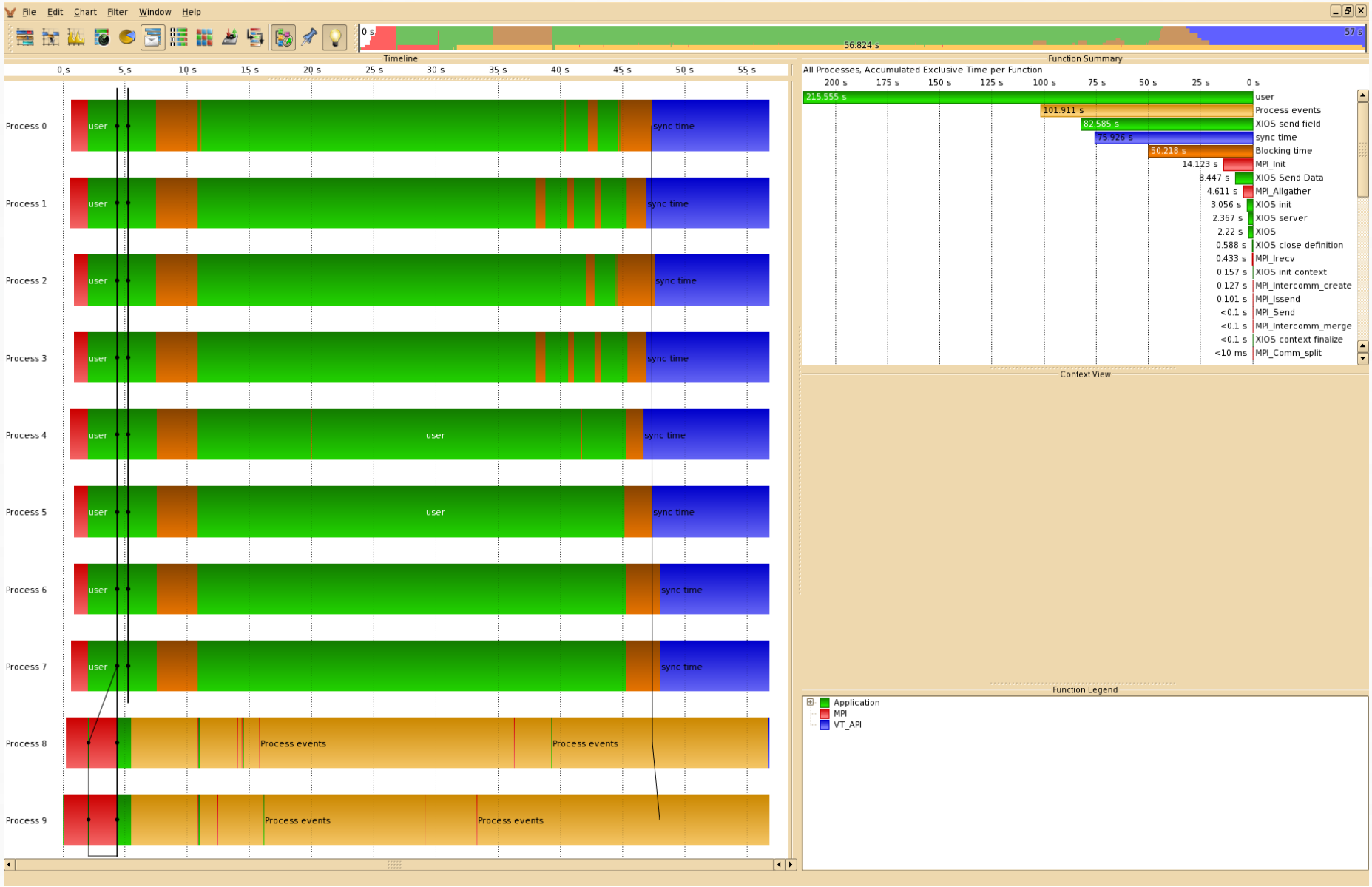
8 clients – 2 serveurs : temps par itération 80 ms



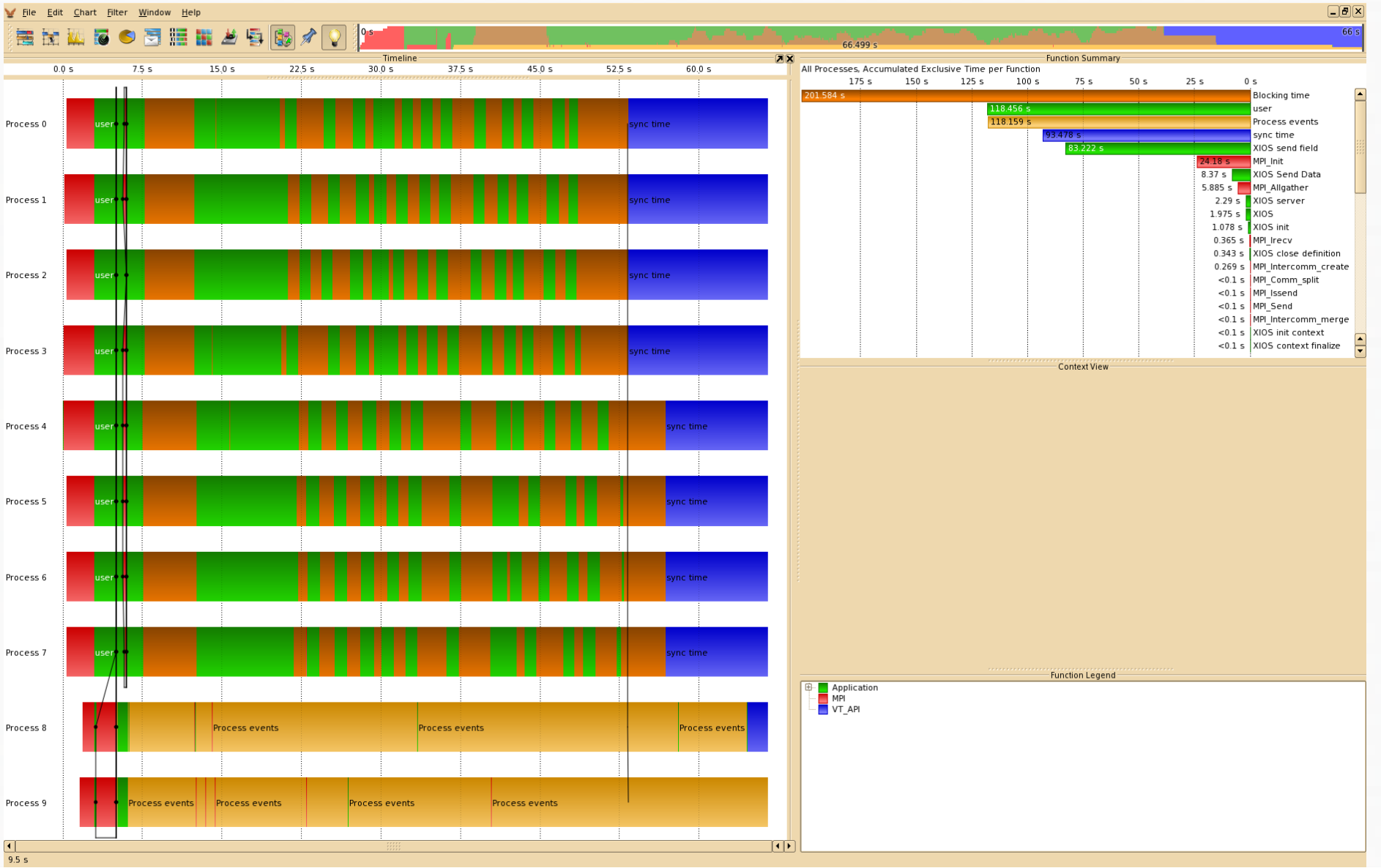
8 clients – 2 serveurs : temps par itération 20 ms



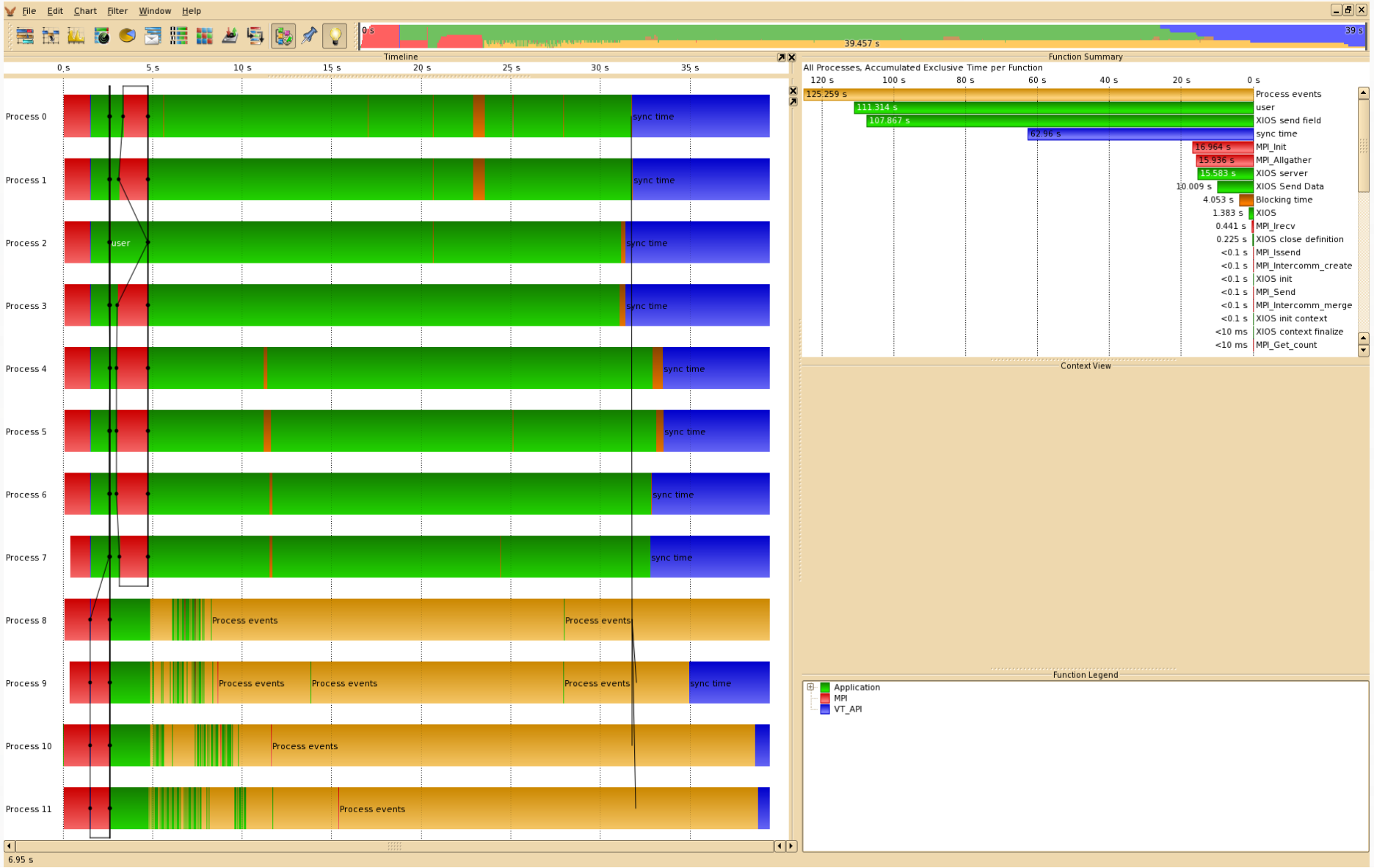
8 clients – 2 serveurs : temps par itération 10 ms



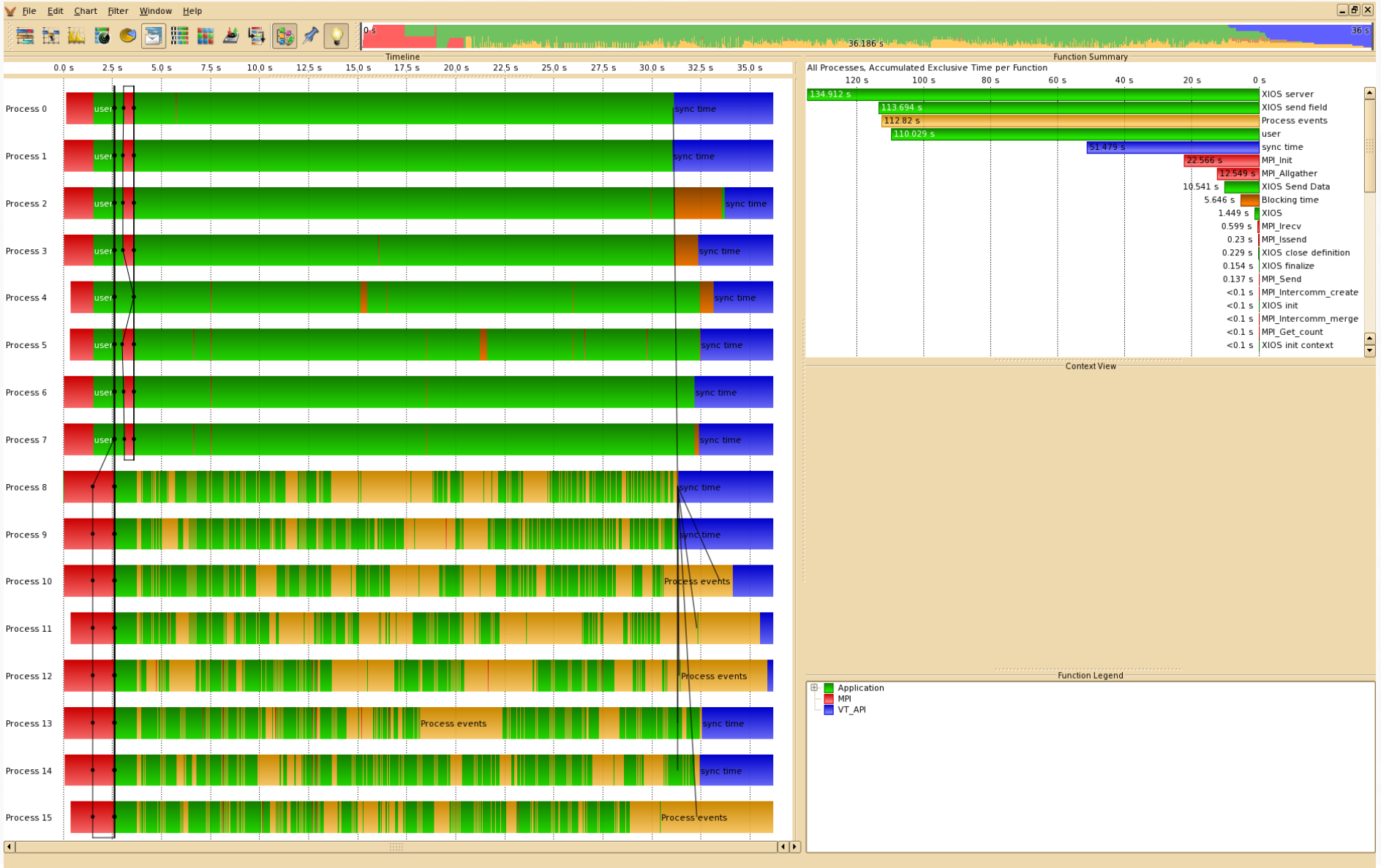
8 clients – 2 serveurs : temps par itération 5 ms



8 clients – 4 serveurs : temps par itération 5 ms



8 clients – 8 serveurs : temps par itération 5 ms



Sortie au format NETCDF4/HDF5

- Utilisation des écritures parallèle via HDF5//
  - MPI/IO en bout de chaîne.
- Couche IO très modulaire
  - Une nouvelle couche peut très facilement être ajoutées.

2 options possibles : "multiple_file" ou "one_file"

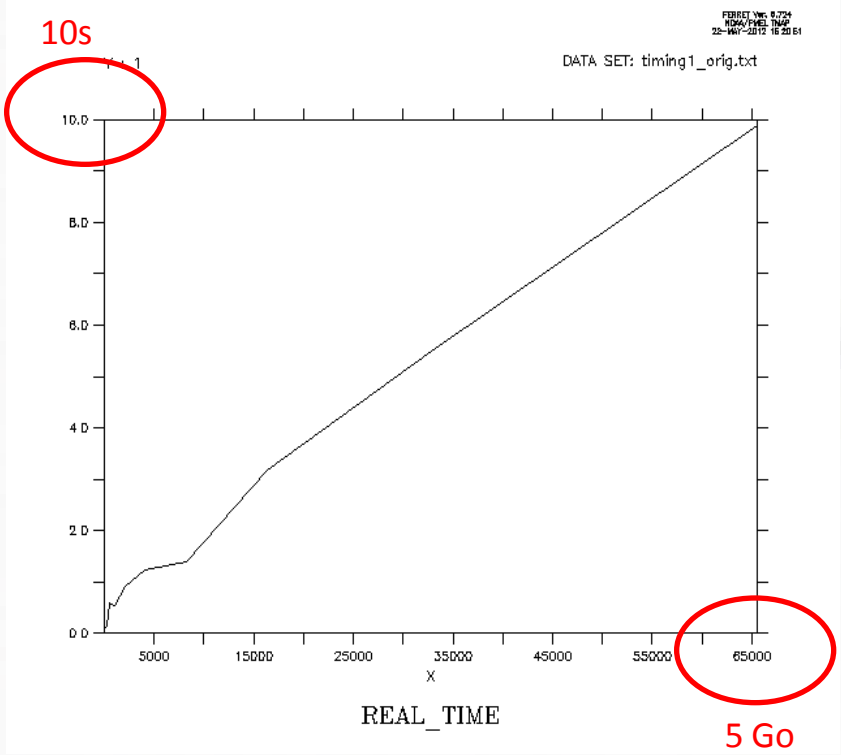
- Le mode peut être sélectionné par l'attribut de fichier : type = "multiple_file" / "one_file"

- "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.

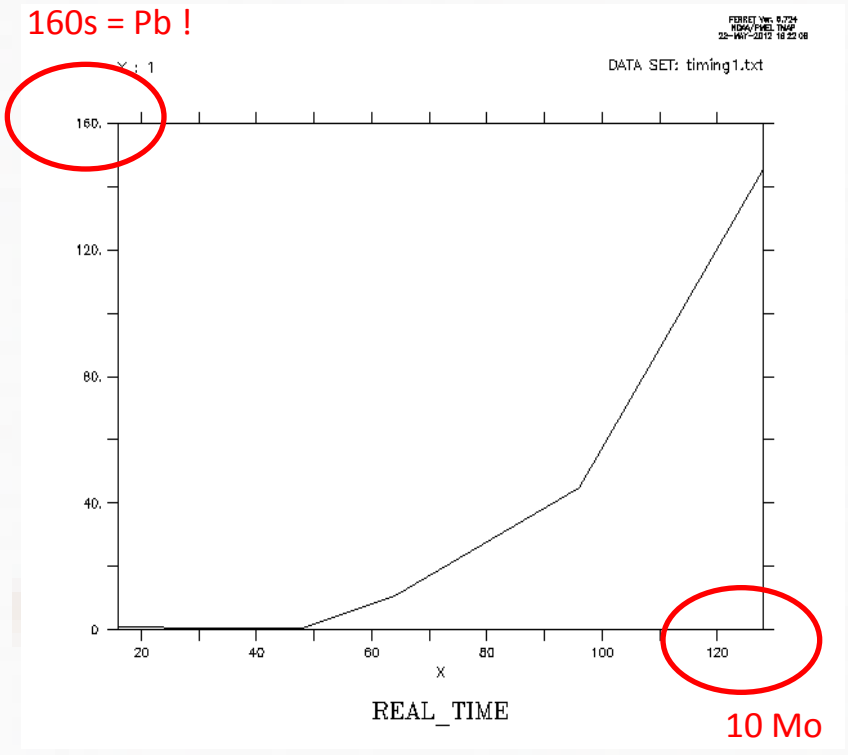
Test IO// sur Curie/Lustre

10s



5 Go

160s = Pb !



10 Mo

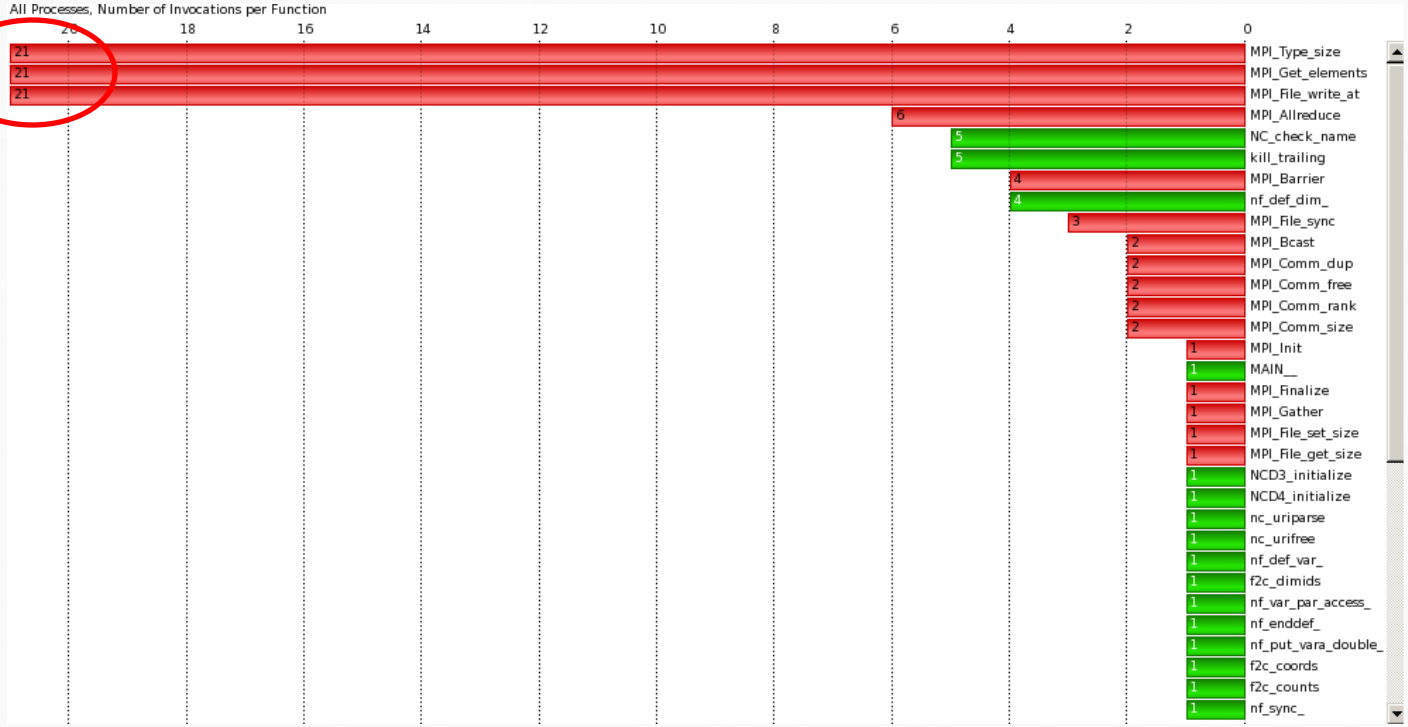
–*Multiple file sur 16 CPUs : 1 fichier par process = 16 fichiers*

–*Single file sur 16 CPUs : 1 fichier final (collective access ou independent access)*

Test IO// sur Curie/Lustre

Grille 100 x 32 x 100

21 appels
MPI_write_file

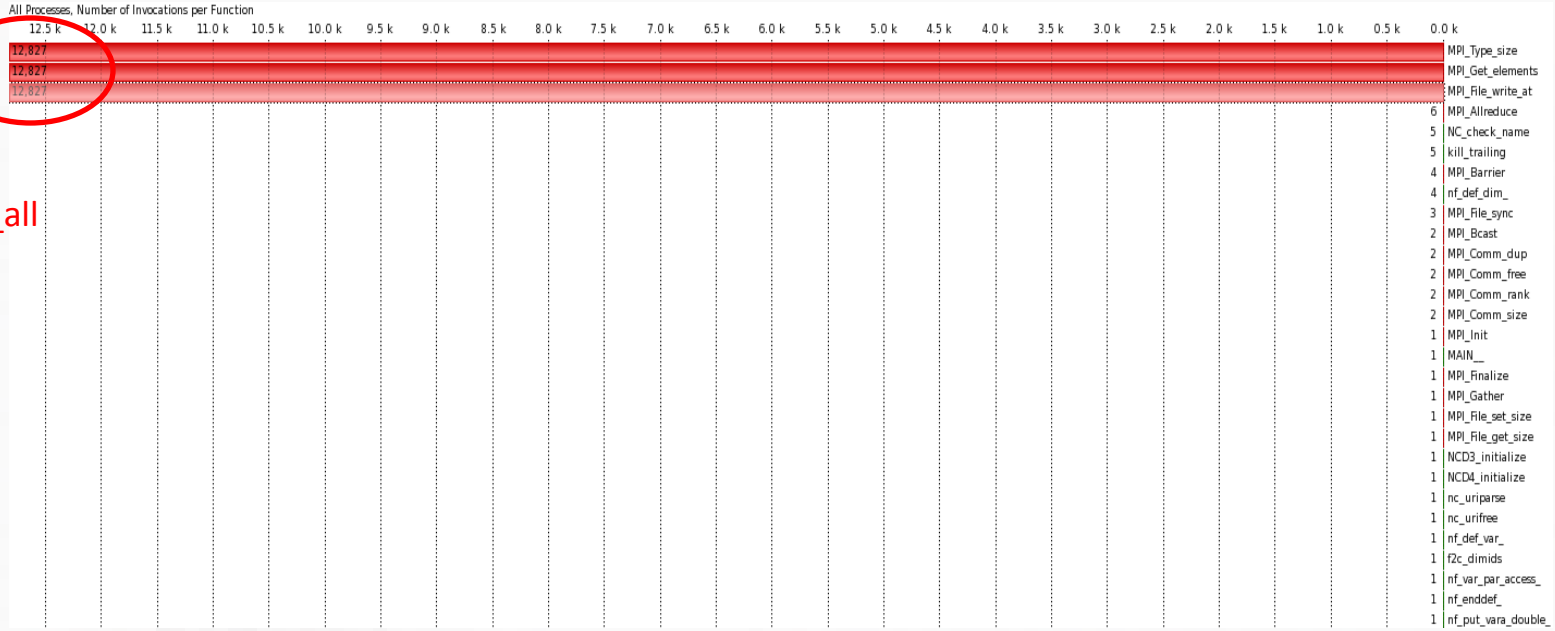


Single file sur 1 CPU: 1 fichier final (collective access)

Test IO// sur Curie/Lustre

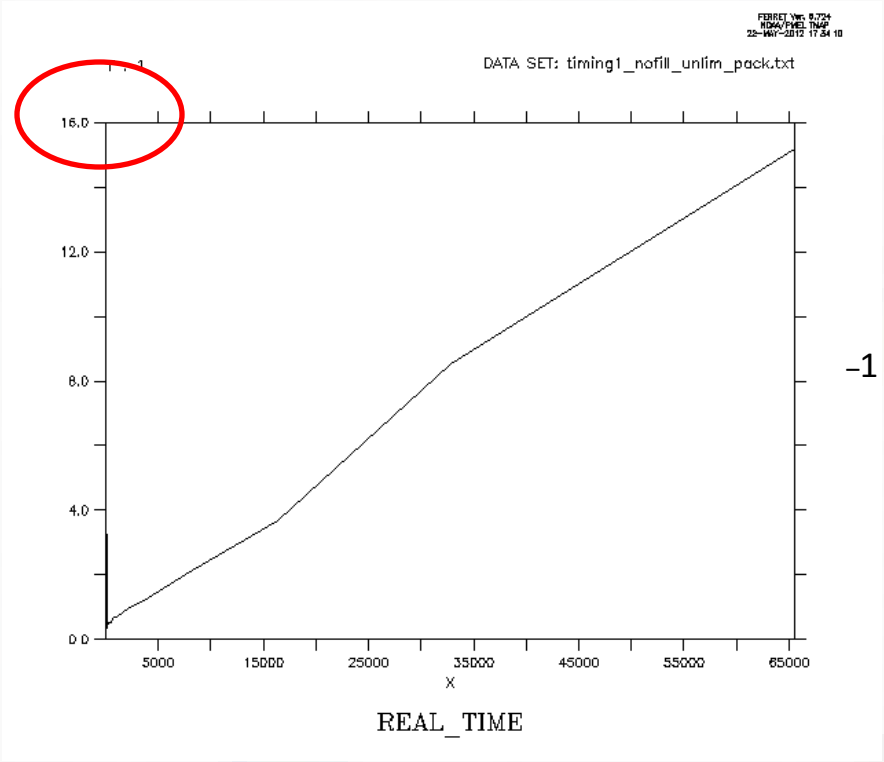
–Grille 100 x 64 x 100

12 827 appels
MPI_write_file_at_all

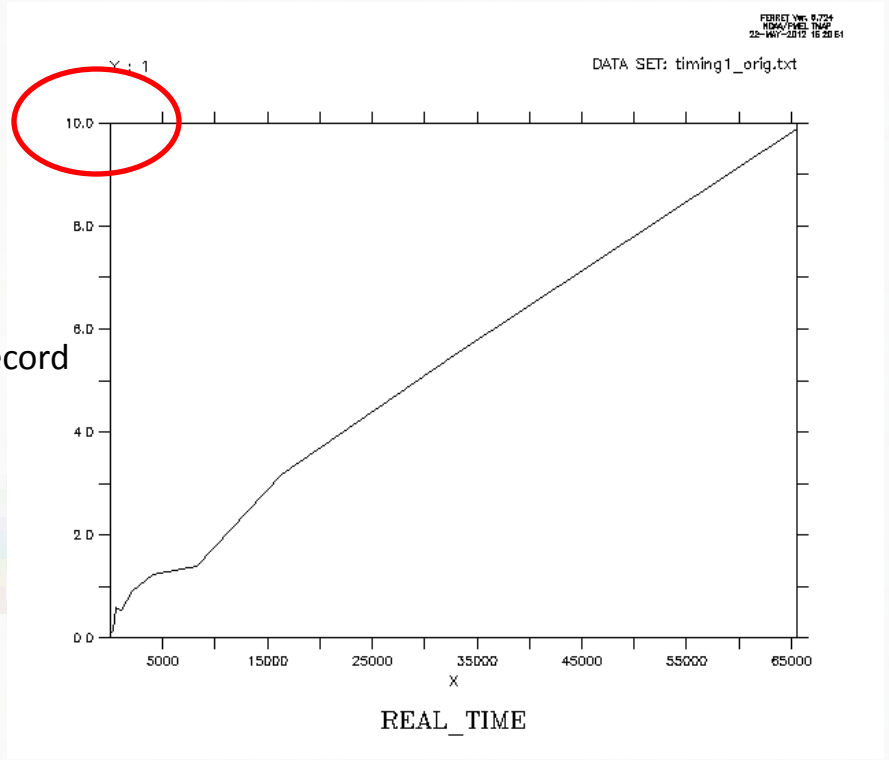


–*Single file sur 1 CPU: 1 fichier final (collective access)*

Test IO// sur Curie/Lustre



-1 time record



- **Single, AVEC unlimited, nf90_def_var_fill, nf90_def_var_chunking**

10 time records : 196 s (1 fichier de 50Go)

=> Bande passante : 270 Mo/s

- **Multiple**

10 time records : 76 s (16 fichiers de 3Go)

=> Bande passante : 600 Mo/s

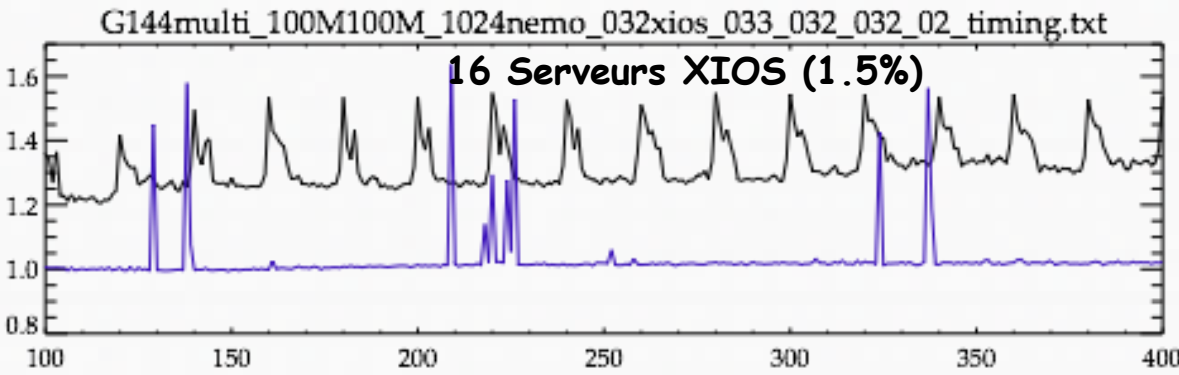
Ecriture NETCDF4/ HDF5 : état de l'art

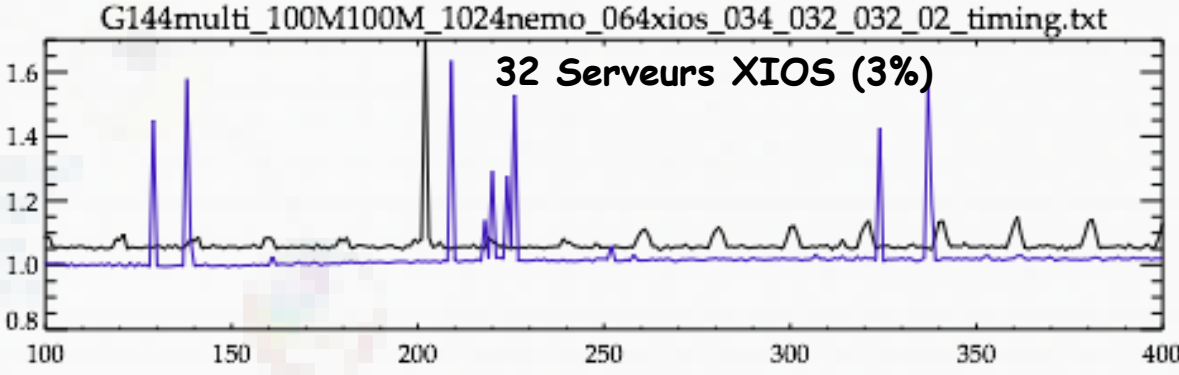
- Constat : difficile d'aggréger de la bande passante au-delà de 0.3 Go/s
- Simple bench MPI/IO : MPI_File_write_at_all, sans vue :
  - Avec 256 OST et 256 noeuds (1 process écrivant par noeuds)
  - 10 Go/s avec des pointes à 20 Go/s
- Déboggage HDF5
  - Probablement un problème dans l'utilisation des vues.
  - Ou mauvaise performance MPIIO de la bibliothèque bullxMPI ?
- En cours d'investigation...

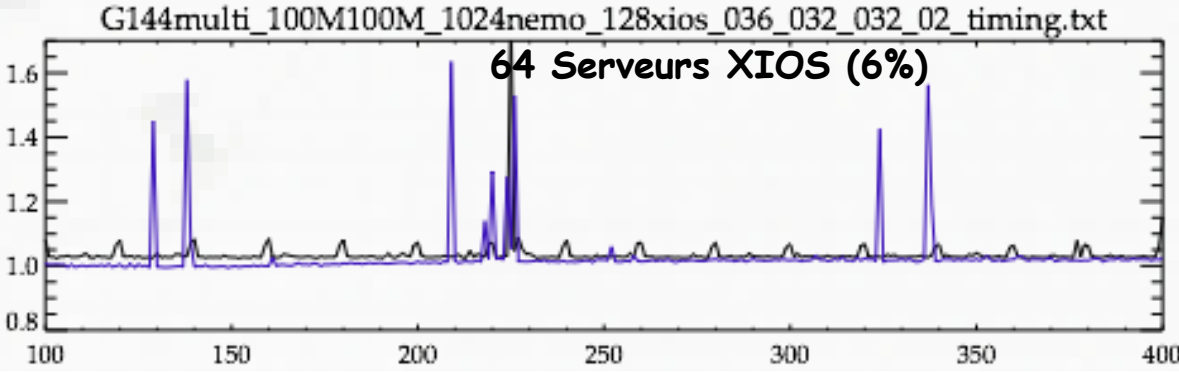
Cas test : XIOS/NEMO

- 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
```

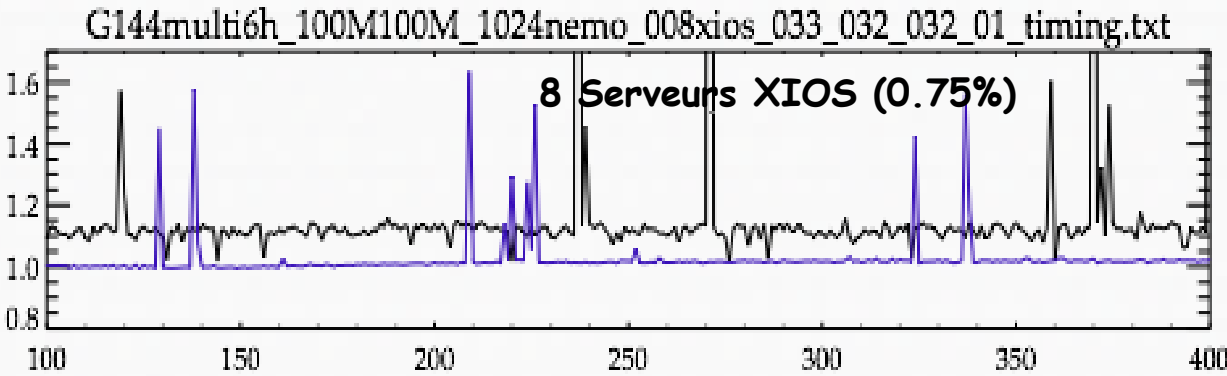


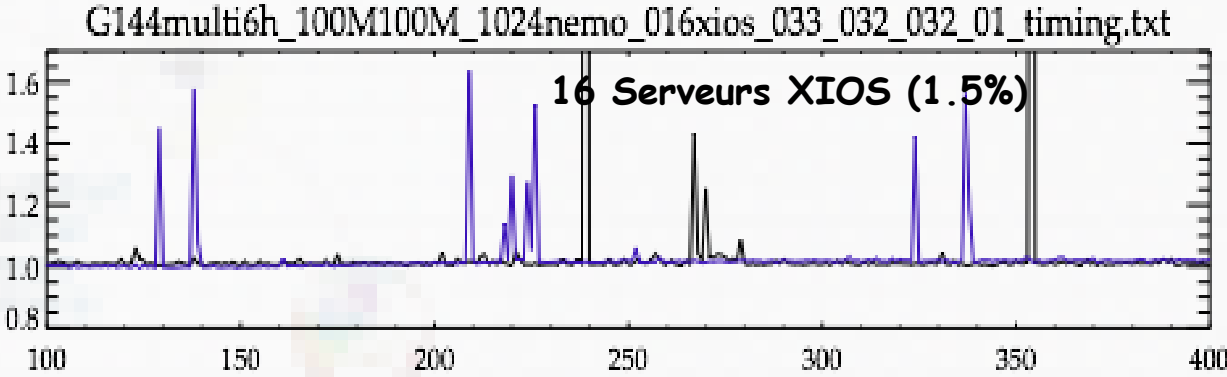


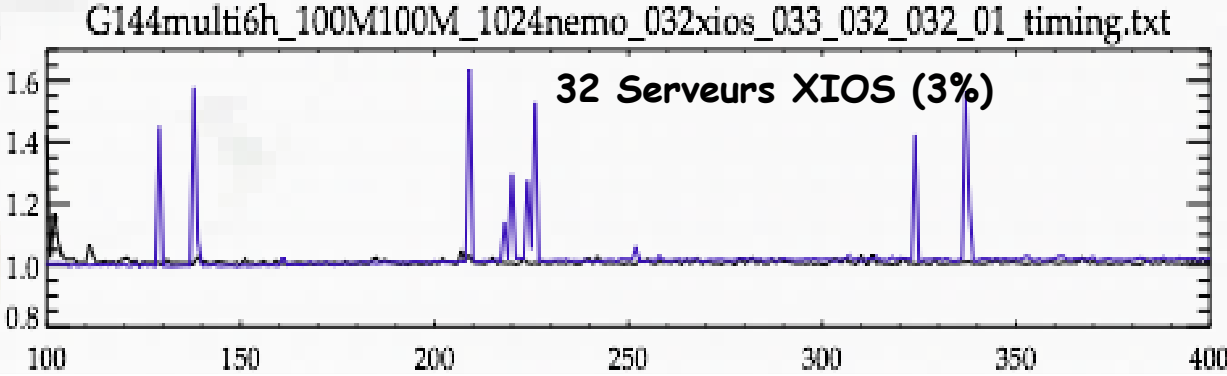


Cas test : XIOS/NEMO

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

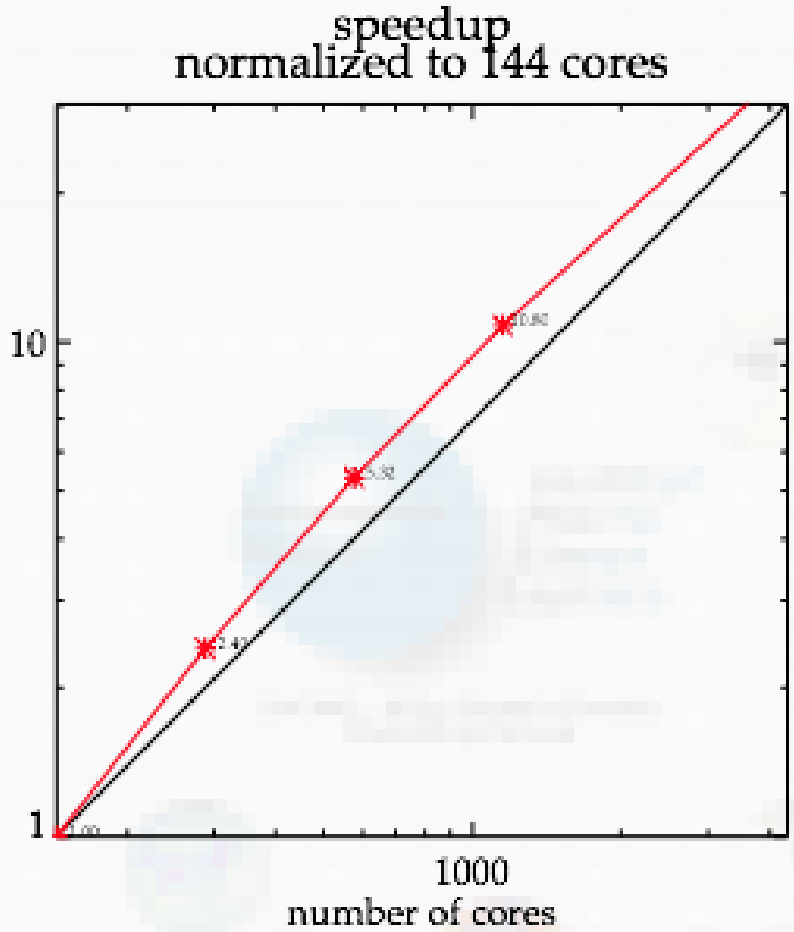


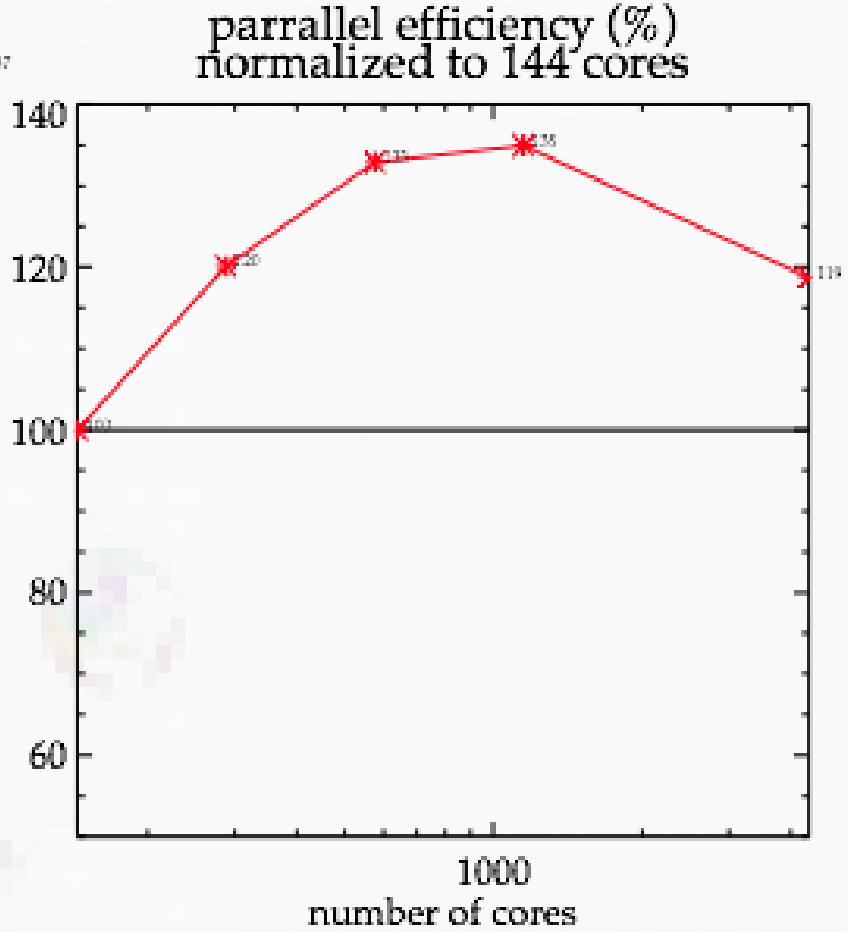




Cas test : XIOS/NEMO : speed-up

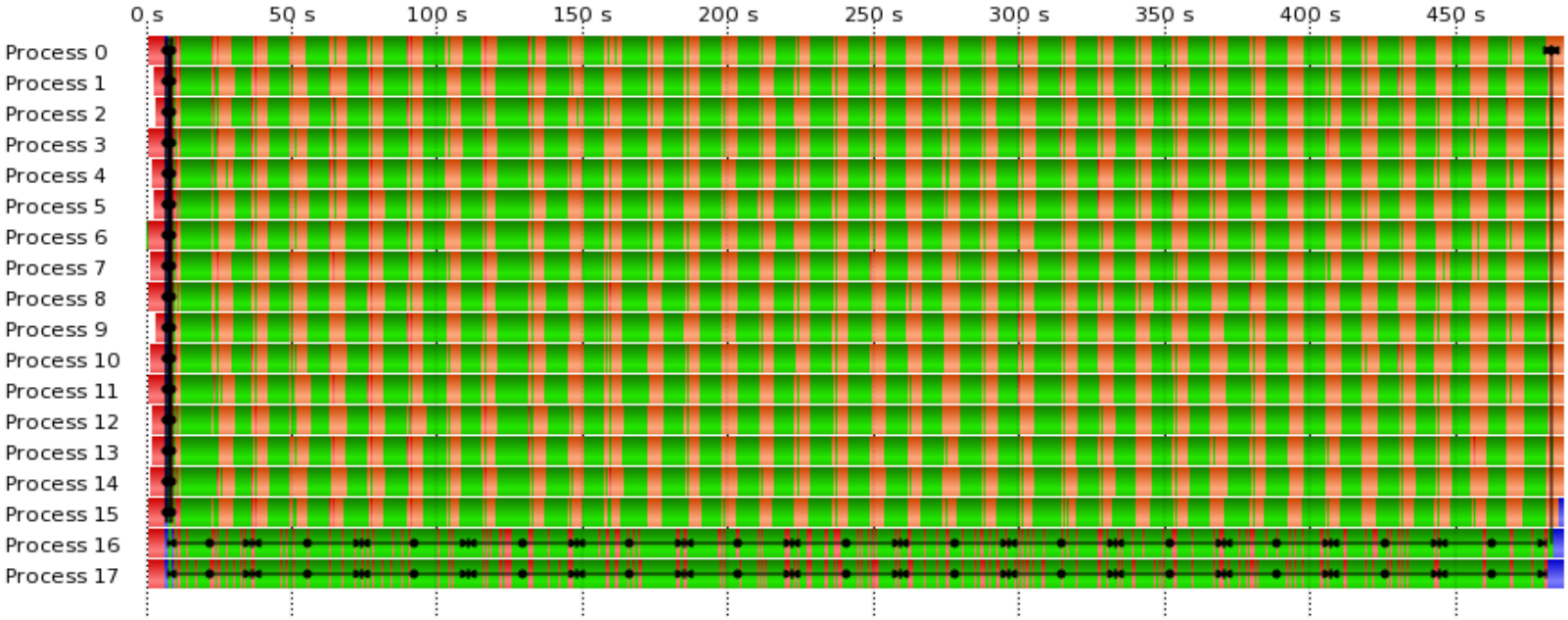
CURIE Fat Nodes: NEMO 3_4_b GYRE Big IO multi_file, jp_cfg = 144



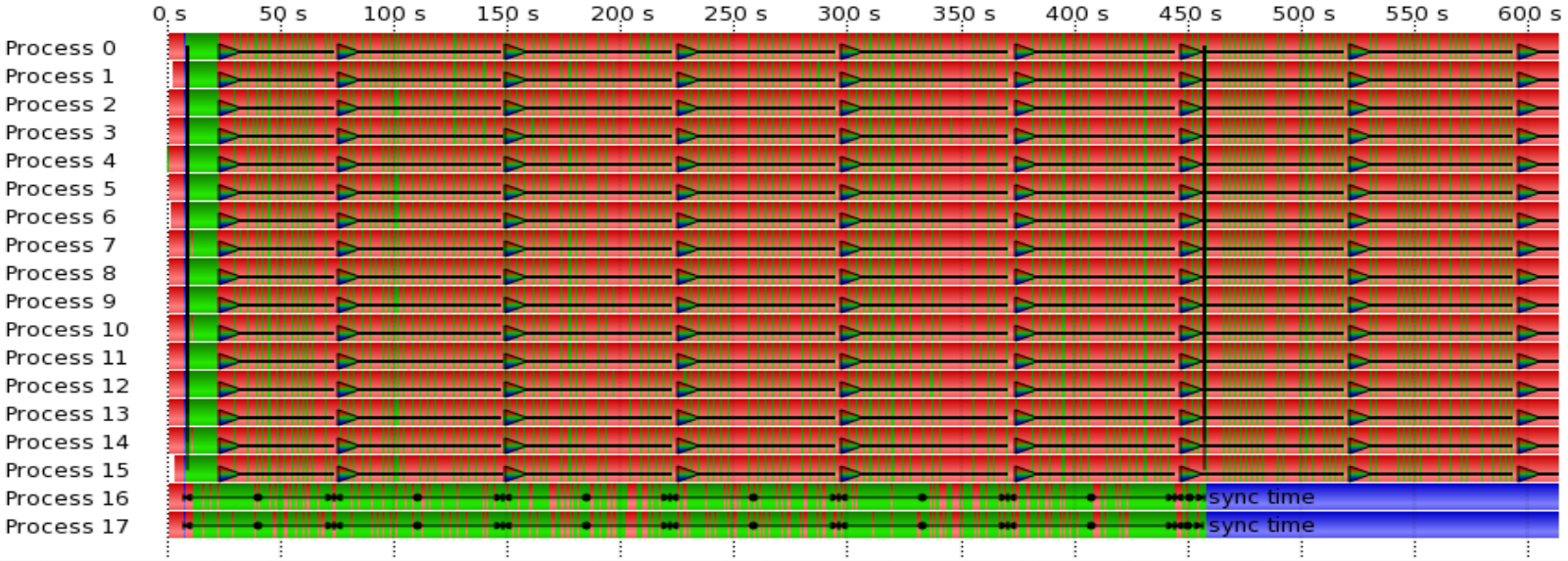


ncores:	144	288	576	1152	4352
timing:	8000	3330	1505	741	223

XIOS-NEMO : trace Vampir



1024 proc
16 serveurs XIOS
multiple_file



1024 proc
16 serveurs XIOS
one_file

Conclusions et perspectives

- XIOS fonctionne plutôt bien
  - Amélioration des performance des écritures HDF5 parallèles.
  - Test de parallel-netcdf ?

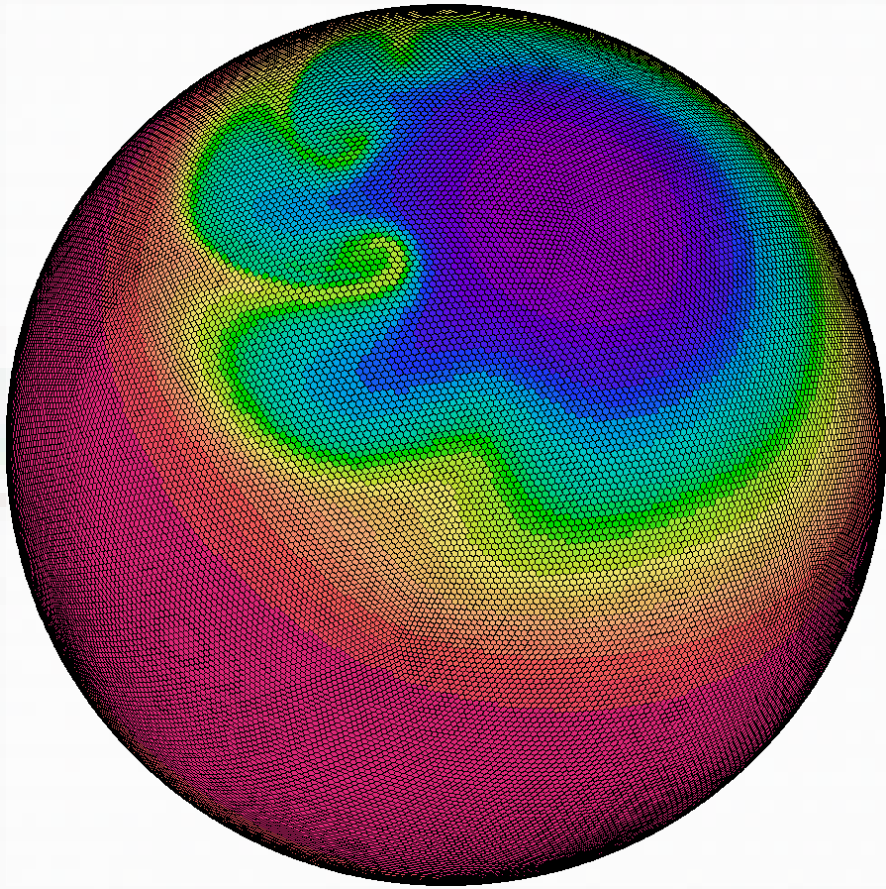
- XIOS : intégration dans l'ensemble des composantes du modèle couplé de l'IPSL.
  - NEMO : officiellement adopté par le consortium NEMO (France, UK, Italie)
    - Intégré dans la prochaine release, début 2013.
  - ORCHIDEE : en cours..
  - INCA
  - LMDZ

- Collaboration extérieure
  - LGGE (Grenoble) modèle régional MAR (H. Gallée) : en cours
  - Météo-France/CNRM (S. Sénesi)

- Collaboration Européenne ?
  - L'IPSL a-t-elle les moyens de fournir du support au-delà de sa communauté ?

Conclusions et perspectives

- Traitements des entrées
  - Lectures des fichiers en avance de phase et envoie asynchrone
    - Gestion des forçages.
    - Gestion des restarts.
- Gestion des grilles non-structurée
  - Projet G8 ICOMEX
    - gestion des grilles icosaédrique.
  - Grilles gaussiennes (CNRM)
- Opération de regrillage
  - Opération de réduction
    - moyennes globales
    - moyennes zonales
  - Dégradation de résolution
  - Interpolation et projections vers d'autres grilles à la volée
    - E. Kritsikis (G8-ICOMEX)



Interpolation (E. Kritsikis)

- Développement de méthode d'interpolation conservative sur la sphère
  - Ordre 2
  - Conservation exact de la masse sur les champs scalaires.
  - Grille géodésique ⇔ grille géodésique
  - Grille lon-lat ⇔ grille lon-lat
  - Grille géodésique ⇔ grille lon-lat
- Calcul des poids en ~ n log n
  - algorithme de recherche basé sur le SSTREE