# Parallélisation par l'intermédiaire d'une fenêtre à mémoire partagée (MPI 3.0) : application à un code de mécanique des fluides

1. Contexte et objectifs
2. Méthode RMA
3. Résultats

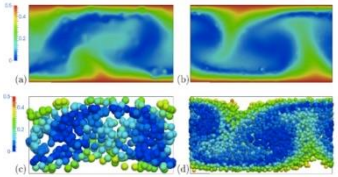
Elyakime P. – Beauvillier M.

JCAD 2023, 3 Octobre 2023

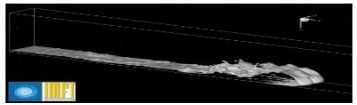
# 1.1 - JADIM

## Code de calcul volumes finis du 2nd ordre @IMFT et @LGC

- Ecoulements instationnaires de fluides incompressibles
- Maillages structurés cartésiens ou curvilignes 2D-3D
- Fortran 90/95 parallélisé hybride MPI-GP-GPU



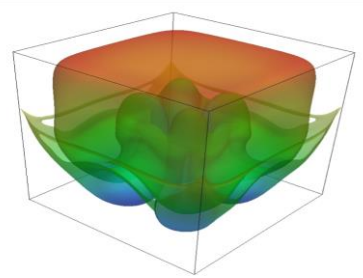
Inertial finite-size particles in turbulent Couette flow (Force Coupling Method). G. Wang.



Gravity current. Y. Hallez

## Modules physiques :

- Multiphasique par une approche de type Volume Of Fluid ou Level-set
- IBM : Ecoulement autour d'objets (Immersed Boundary Method)
- LES (Large Eddy Simulation)
- Transfert thermique
- Réaction chimique, …



Crustal polydiapirs
(Convection dans la croûte terrestre)
A. Louis-Napoléon

# 1.2 - Méthode de projection & temps de calcul

CPU

CPU/GPU

CPU

- Calcul de l'avancement en temps (schéma de Runge-Kutta / Crank-Nicolson)

$$\frac{\hat{V}_i^{n+1} - V_i^n}{\Delta t}\vartheta = -\left(\frac{1}{\rho}\frac{\partial P}{\partial \xi_i}\right)^n \vartheta + \text{Gravité} + \text{Advection} + \text{Diffusion} + \text{Effets capillaires}$$

- Résolution d'un **système linéaire** (pseudo-équation de Poisson)

$$\frac{1}{\Delta t}\nabla.\hat{\mathbf{V}}^{n+1} = \nabla.\left(\frac{1}{\rho}\nabla\Phi\right)$$

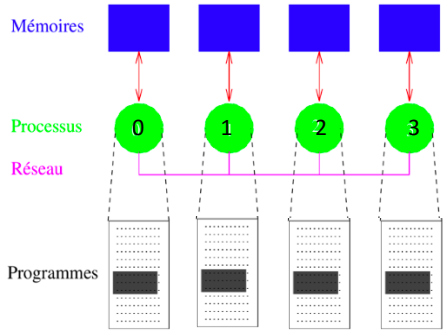
***Solveurs : PETSc , Mumps, Fourier [MPI] AmgX [GP-GPU]***

- Mise à jour de la vitesse et de la pression

$$\frac{V_i^{n+1} - \hat{V}_i^{n+1}}{\Delta t} = -\frac{1}{\rho}\frac{\partial \Phi}{\partial \xi_i} \qquad P^{n+1} = P^n + \Phi$$

# 1.3 – Parallélisation de JADIM

La parallélisation de JADIM se base sur la technique à mémoire distribuée (SPMD)




Calcul parallèle à mémoire distribuée

Schéma de la programmation parallèle (Idris)

- Chaque processus a sa propre mémoire
- Plusieurs processus exécutent le programme sur des sous domaines MPI du maillage
- Echanges des données entre les processus via des appels utilisant la bibliothèque MPI

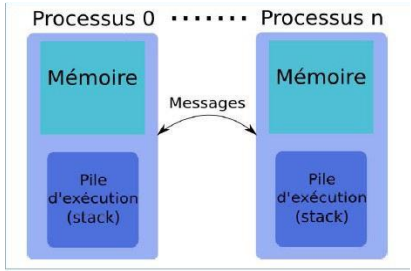
# 1.3 – Parallélisation de JADIM

1. Définition de la distribution de la mémoire en discrétisant le domaine : chaque processus construit les bornes de son sous domaine

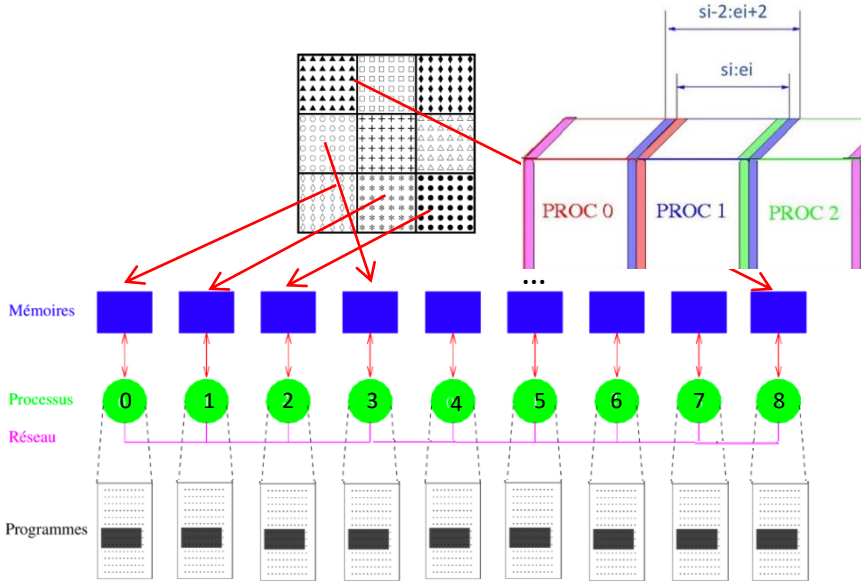
2. Les variables des sous domaines sont stockés dans la mémoire privée de chacun des processus

3. Les processus MPI calcul une partie du programme sur son sous domaine en utilisant les données des variables de ce sous domaine

4. Les processus MPI échangent les données




```
$ mpirun -np 9 ./jadimles.exe 3 3 1
```



# 1.4 – Mémoire & Parallélisation

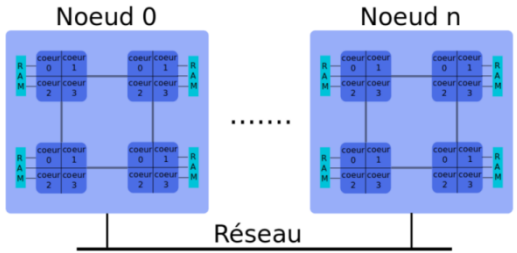



Schéma d'un supercalculateur (Idris)

Variable parallélisée : les données sont distribuées à tous les proc. MPI

Variable non parallélisée : proc. MPI possèdent la variable sur tout le domaine

ex: un code lancé sur 2 nœuds de calcul et 32 processus MPI par nœud aura donc 32 fois une variable sur tous les points du maillage en mémoire

# 1.5 – JADIM : mémoire non parallélisée

| Tableau | Dimension | Mémoire $(512^3)$ | Mémoire $(1024^3)$ |
|---|---|---|---|
| **nfa** | (nx + 2) × (ny + 2) × (nz/Nz + 2)) × 4 × 4 bits | 4,19 x (nz/Nz + 2) MB **= 0,54 GB *** | 16,8 x (nz/Nz + 2) MB **= 4,3 GB *** |
| **coor** | (nx + 2) x (ny + 2) x 8 x 8 bits | 16,9 MB | 67,4 MB |
| **area** | (nx + 2) x (ny + 2) x 16 x 8 bits | 33,8 MB | 135 MB |
| **dist** | (nx + 6) x (ny + 6) x 16 x 8 bits | 23,6 MB | 94,5MB |
| **curvn** | (nx + 2) x (ny + 2) x 6 x 8 bits | 17,2 MB | 50,5 MB |
| **curv** | (nx + 2) x (ny + 2) x 20 x 8 bits | 42,3 MB | 168 MB |
| **fac** | ((2 x nx + 2 x ny) x nz + 1) x 7 x 4 bits | 29,1 MB | 117 MB |
| **facl** | ((2 x nx + 2 x ny) x nz + 1) x 3 x 4 bits | 12,5 MB | 50,1 MB |
| **tang** | ((2 x nx + 2 x ny) x nz + 1) x 4 x 8 bits | 33,2 MB | 133,8 MB |
| **IJP** | (nx + 6) × (ny + 6) × 4 bits | 1,07 MB | 4,23 MB |
| ***Total** * | | **23,9 GB** | **164 GB** |

* Pour 1 nœud de calcul avec 32 cœurs et une décomposition par Nz=4 dans la direction Z

# 1.6 – Objectif de l’étude

## Problème

**Consommation de la mémoire des variables non parallélisées trop importante**

# 1.6 – Objectif de l'étude

## Problème

**Consommation de la mémoire des variables non parallélisées trop importante**

## Solutions

- **Dépeupler les nœuds de calculs** => perte en efficacité de calcul

# 1.6 – Objectif de l'étude

## Problème

**Consommation de la mémoire des variables non parallélisées trop importante**

## Solutions

**Dépeupler les nœuds de calculs** => perte en efficacité de calcul

**Parallélisation à mémoire distribuée** des variables => modification de tout le code

# 1.6 – Objectif de l'étude

## Problème

**Consommation de la mémoire des variables non parallélisées trop importante**

## Solutions


**Dépeupler les nœuds de calculs** => perte en efficacité de calcul


**Parallélisation à mémoire distribuée** des variables => modification de tout le code


**Partager une zone mémoire** à l'ensemble des processus au sein d'un même nœud

# 1.6 – Objectif de l'étude

## Problème

**Consommation de la mémoire des variables non parallélisées trop importante**

## Solutions


**Dépeupler les nœuds de calculs** => perte en efficacité de calcul


**Parallélisation à mémoire distribuée** des variables => modification de tout le code

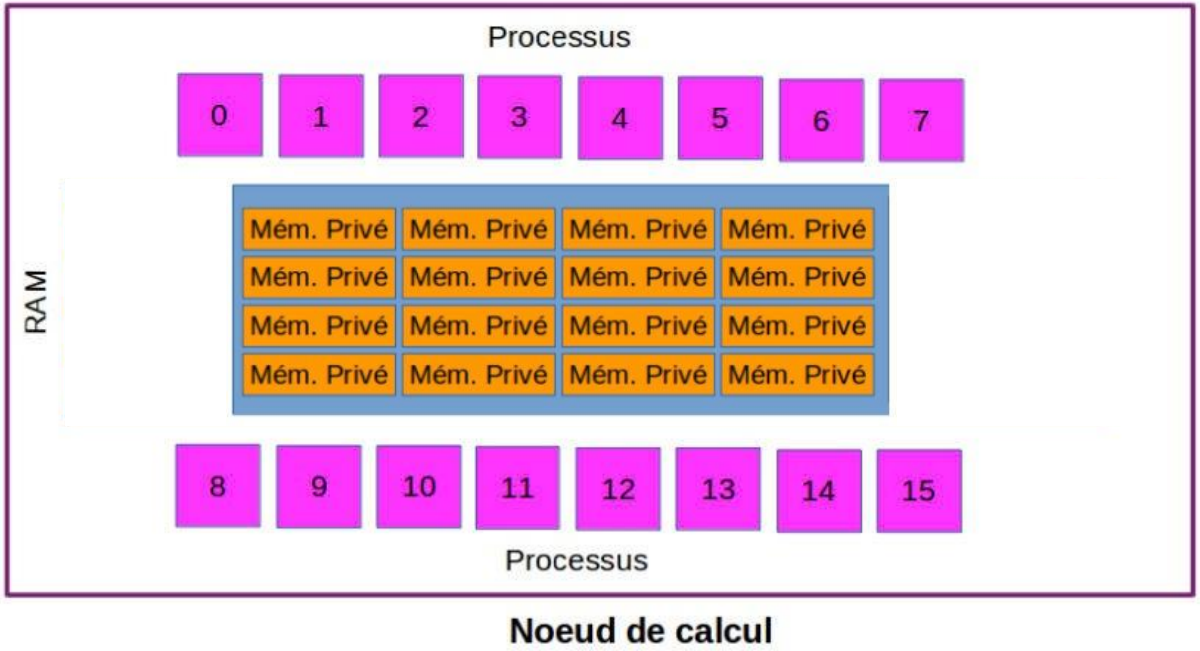

**Partager une zone mémoire** à l'ensemble des processus au sein d'un même nœud

## Comment faire ?

Utiliser une nouvelle approche de programmation introduite dans la norme **MPI 3.0 nommée Remote Memory Access** (RMA) avec la fenêtre à mémoire partagée

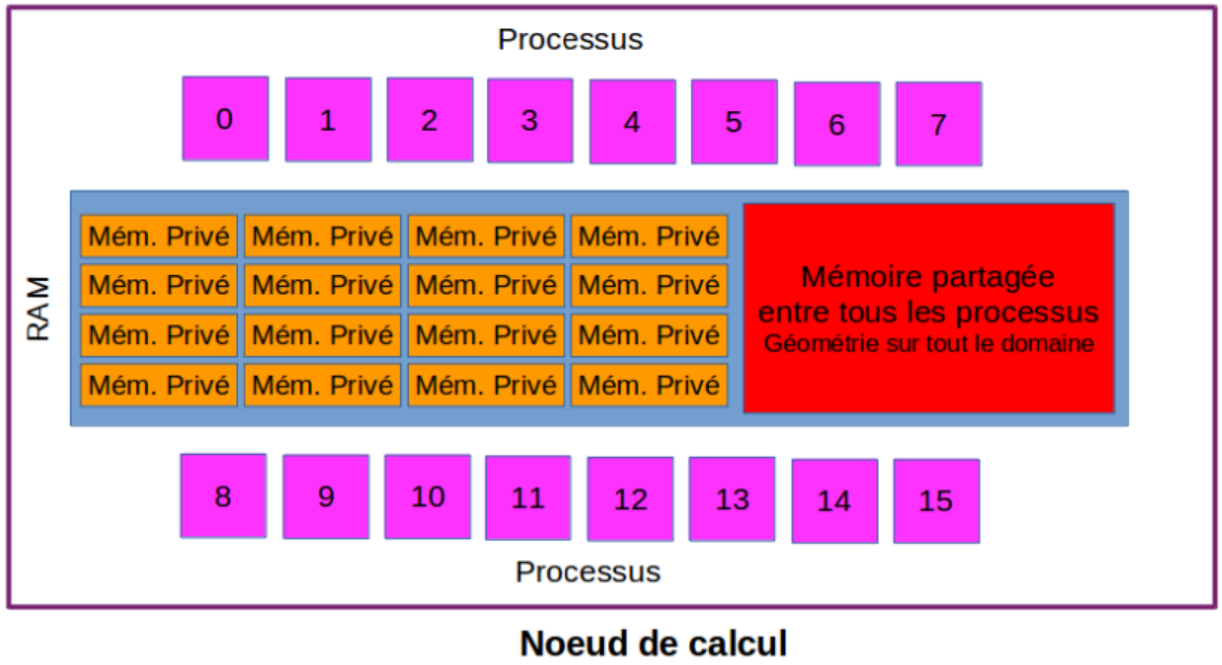
# 2. La méthode RMA

# 2.1 – Approche avec la méthode RMA




Mémoire privée uniquement

# 2.1 – Approche avec la méthode RMA

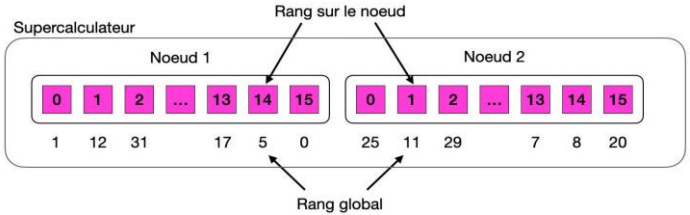



Mémoire privée + partagée

# 2.2 – Implémentation dans le code JADIM

## 1- Construction du communicateur sur les noeuds

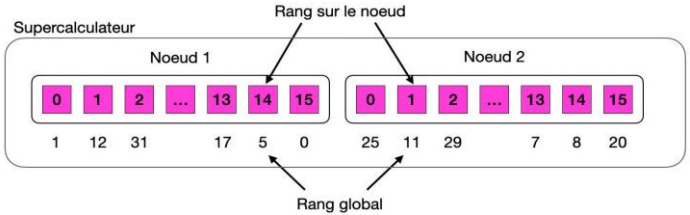
```
!For RMA
call MPI_COMM_SPLIT_TYPE(mpi_comm_world, MPI_COMM_TYPE_SHARED,&
                         t_p, mpiinfo, mpi_comm_node, mpierr)
call MPI_COMM_RANK(mpi_comm_node, t_p_node, mpierr)
call MPI_COMM_SIZE(mpi_comm_node, nb_procs_node, mpierr)
```



# 2.2 – Implémentation dans le code JADIM

## 1- Construction du communicateur sur les noeuds

```
!For RMA
call MPI_COMM_SPLIT_TYPE(mpi_comm_world, MPI_COMM_TYPE_SHARED,&
                         t_p, mpiinfo, mpi_comm_node, mpierr)
call MPI_COMM_RANK(mpi_comm_node, t_p_node, mpierr)
call MPI_COMM_SIZE(mpi_comm_node, nb_procs_node, mpierr)
```

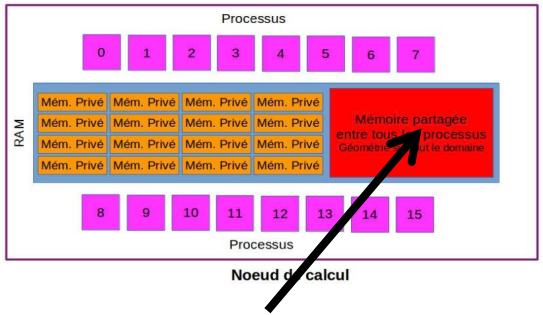



## 2- Construction de la fenêtre mémoire et accès à cette fenêtre

```
! Definition of the windows size
size_mem_coor=0_MPI_ADDRESS_KIND
if (t_p_node==0) size_mem_coor=int((nx+2)*(ny+2), &
                 MPI_ADDRESS_KIND)*64_MPI_ADDRESS_KIND ! 8*8

! Allocation of the memory window.
call MPI_WIN_ALLOCATE_SHARED(size_mem_coor,1,MPI_INFO_NULL, &
                  mpi_comm_node,baseptr_coor,win_coor,mpierr)

! Other processes query access to the memory window of t_p_node==0
if (t_p_node /= 0) call MPI_Win_shared_query(win_coor, 0, &
                   size_mem_coor, disp_unit,baseptr_coor,mpierr)
```

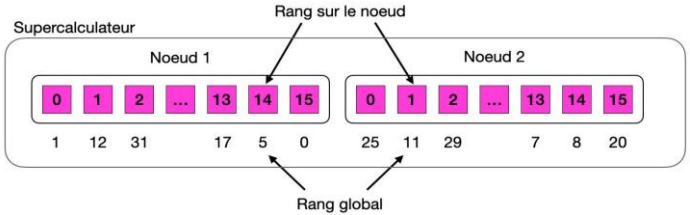



nfa, coor, dist, area, curvn, curv, fac, facl, tang, ijp

# 2.2 – Implémentation dans le code JADIM

## 1- Construction du communicateur sur les noeuds

```
!For RMA
call MPI_COMM_SPLIT_TYPE(mpi_comm_world, MPI_COMM_TYPE_SHARED,&
                         t_p, mpiinfo, mpi_comm_node, mpierr)
call MPI_COMM_RANK(mpi_comm_node, t_p_node, mpierr)
call MPI_COMM_SIZE(mpi_comm_node, nb_procs_node, mpierr)
```

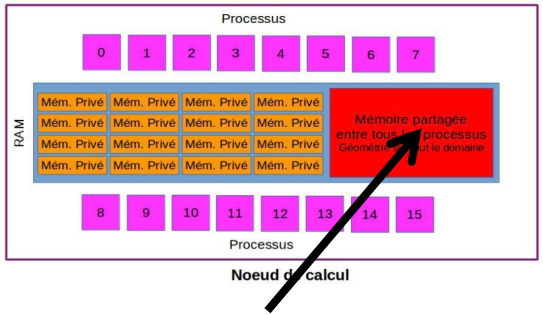



## 2- Construction de la fenêtre mémoire et accès à cette fenêtre

```
! Definition of the windows size
size_mem_coor=0_MPI_ADDRESS_KIND
if (t_p_node==0) size_mem_coor=int((nx+2)*(ny+2), &
                 MPI_ADDRESS_KIND)*64_MPI_ADDRESS_KIND ! 8*8

! Allocation of the memory window.
call MPI_WIN_ALLOCATE_SHARED(size_mem_coor,1,MPI_INFO_NULL, &
                 mpi_comm_node,baseptr_coor,win_coor,mpierr)

! Other processes query access to the memory window of t_p_node==0
if (t_p_node /= 0) call MPI_Win_shared_query(win_coor, 0, &
                   size_mem_coor, disp_unit,baseptr_coor,mpierr)
```

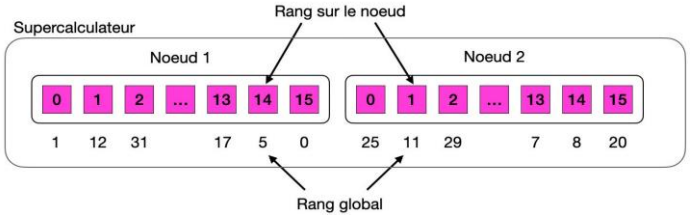



nfa, coor, dist, area, curvn, curv, fac, facl, tang, ijp

# 2.2 – Implémentation dans le code JADIM

## 1- Construction du communicateur sur les noeuds

```
!For RMA
call MPI_COMM_SPLIT_TYPE(mpi_comm_world, MPI_COMM_TYPE_SHARED,&
                         t_p, mpiinfo, mpi_comm_node, mpierr)
call MPI_COMM_RANK(mpi_comm_node, t_p_node, mpierr)
call MPI_COMM_SIZE(mpi_comm_node, nb_procs_node, mpierr)
```

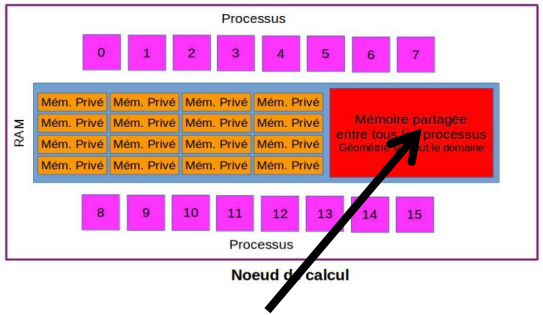



## 2- Construction de la fenêtre mémoire et accès à cette fenêtre

```
! Definition of the windows size
size_mem_coor=0_MPI_ADDRESS_KIND
if (t_p_node==0) size_mem_coor=int((nx+2)*(ny+2), &
                 MPI_ADDRESS_KIND)*64_MPI_ADDRESS_KIND ! 8*8

! Allocation of the memory window.
call MPI_WIN_ALLOCATE_SHARED(size_mem_coor,1,MPI_INFO_NULL, &
                 mpi_comm_node,baseptr_coor,win_coor,mpierr)

! Other processes query access to the memory window of t_p_node==0
if (t_p_node /= 0) call MPI_Win_shared_query(win_coor, 0, &
                 size_mem_coor, disp_unit,baseptr_coor,mpierr)
```




nfa, coor, dist, area, curvn, curv, fac, facl, tang, ijp

# 2.2 – Implémentation dans le code JADIM

**3 - Transformation d'un pointeur C en un tableau Fortran**

```
! Transformation of the C pointer to a fortran array
allocate(arrayshape(2)); arrayshape=(/ nx+2,ny+2 /)
CALL C_F_pointer(baseptr_coor,coor_tmp,arrayshape)

! Reshaping of the Fortran to beginning
! at the correct position
coor(0:,0:) => coor_tmp
```

# 2.2 – Implémentation dans le code JADIM

**3 - Transformation d'un pointeur C en un tableau Fortran**

```
! Transformation of the C pointer to a fortran array
allocate(arrayshape(2)); arrayshape=(/ nx+2,ny+2 /)
CALL C_F_pointer(baseptr_coor,coor_tmp,arrayshape)

! Reshaping of the Fortran to beginning
! at the correct position
coor(0:,0:) => coor_tmp
```

# 2.2 – Implémentation dans le code JADIM

**3 - Transformation d'un pointeur C en un tableau Fortran**

```
! Transformation of the C pointer to a fortran array
allocate(arrayshape(2)); arrayshape=(/ nx+2,ny+2 /)
CALL C_F_pointer(baseptr_coor,coor_tmp,arrayshape)

! Reshaping of the Fortran to beginning
! at the correct position
coor(0:,0:) => coor_tmp
```

**4 – Modification du tableau**

```
! Fill coor%x array
if (t_p_node == 0 ) then
  coor(i,j)%x = xc1(nfa(i,j,2,2))
endif
```

**coor est rempli exclusivement par le processus 0 du nœud**

# 3. Les résultats

# 3.1 – Les résultats

## Présentation du supercalculateur Olympe de CALMIP

Supercalculateur OLYMPE
[site:https://www.calmip.univ-toulouse.fr/]

### Deux partitions CPU

- 360 nœuds bi-socket (2 sockets de 18 cœurs)
  - 192 Go de mémoire par nœud
- 2 nœuds de large mémoire (1 socket de 18 cœurs)
  - 750 Go de mémoire par nœud

« nœud »

« mesca »

### Une partition GP-GPU:

- 12 nœuds GPU (2 sockets de 18 cœurs + 4 cartes GPU)
  - 377 Go de mémoire par nœud

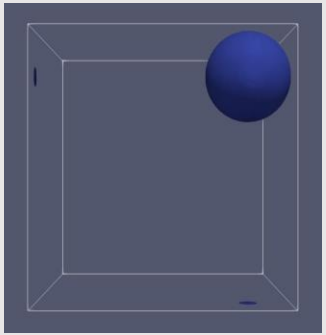
« volta »

# 3.2 – Maillage maximum

**Configuration des tests :**

Cas test : diphasique (VoF) – dynamique  d'une BULLE animée par de la gravité
Maillage : cartésien orthogonal régulier
Condition aux limites : périodiques
JADIM version « RMA » vs JADIM version « master »
Nombre de processus : 1 Node, 8 Nodes, 16 Nodes, 24 Nodes

Simulation sur **2 pas de temps**



- **Recherche du maillage de taille maximum**

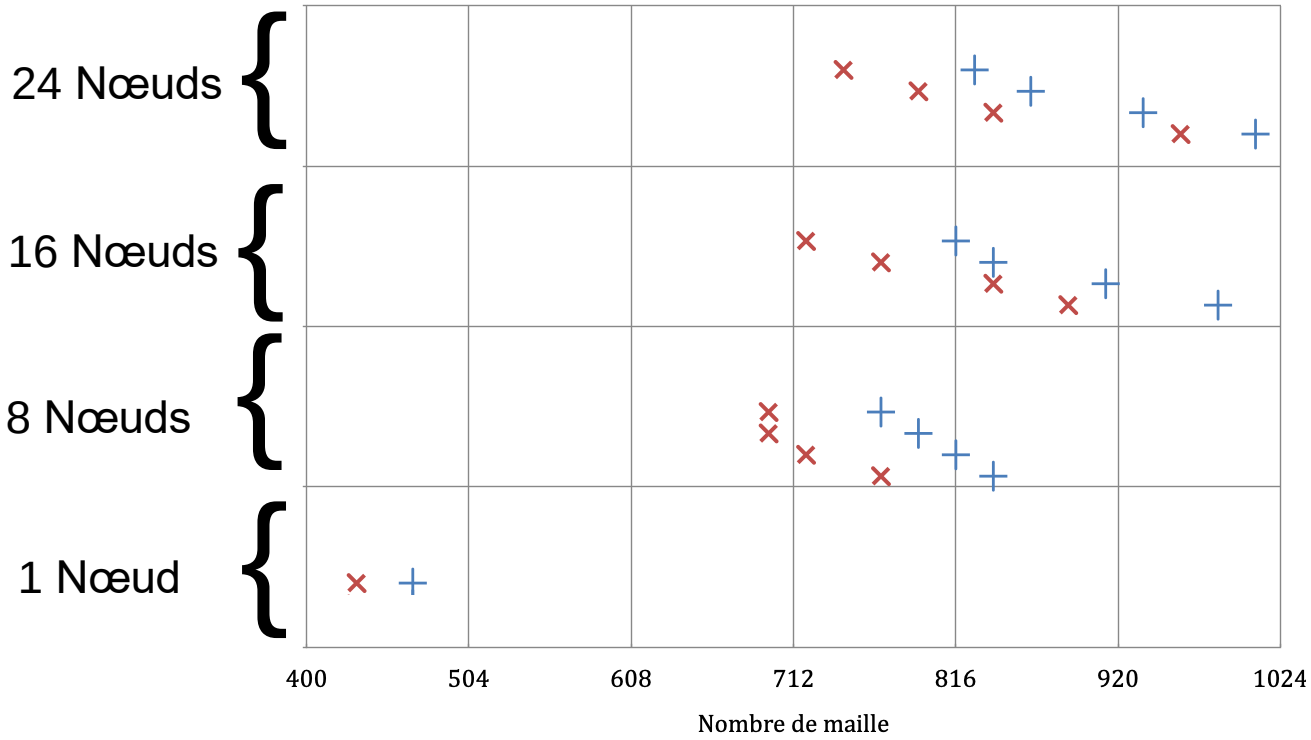
# 3.2 – Maillage maximum

**Légende :**

+ RMA

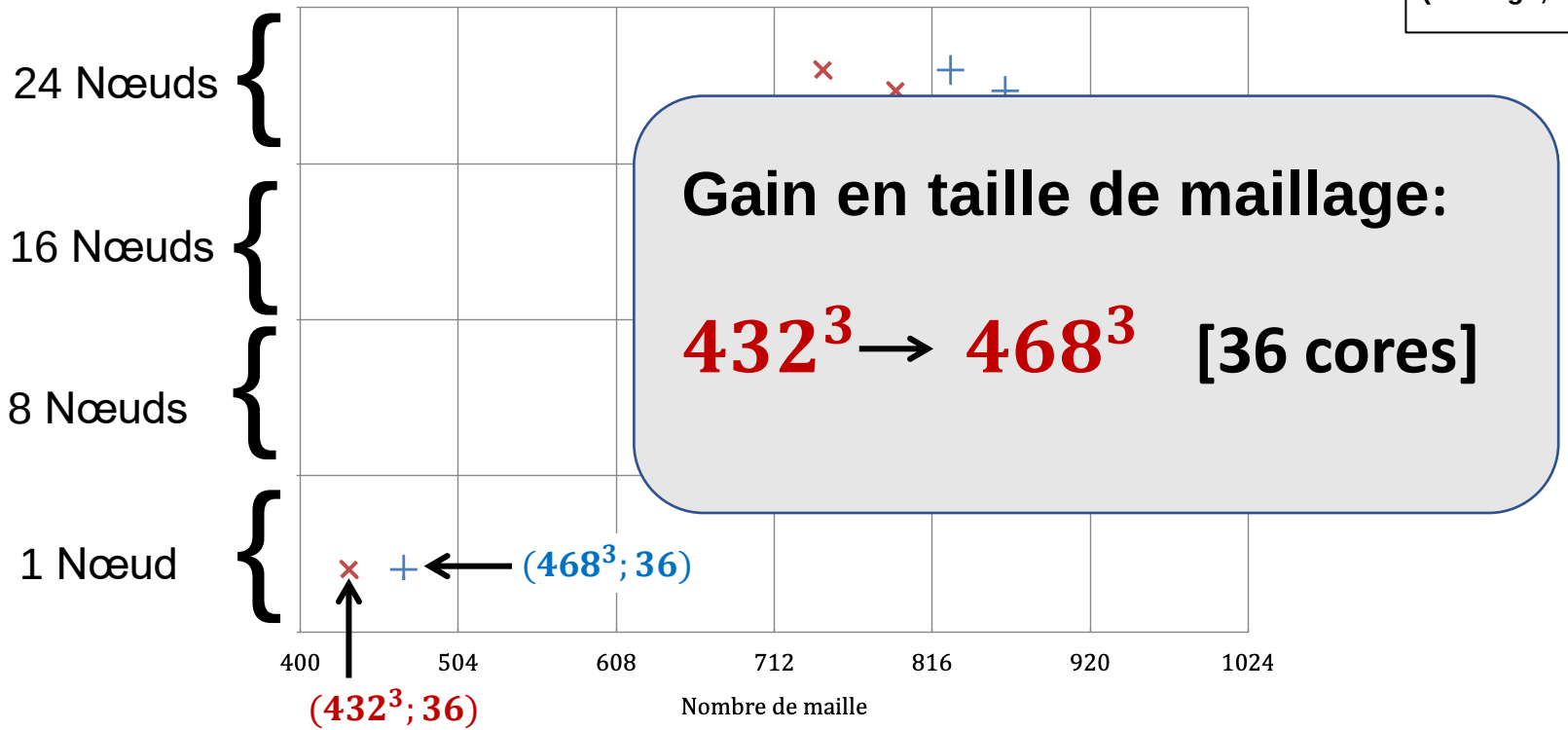
× Master

**(Maillage; Nb_Proc_MPI/Node)**



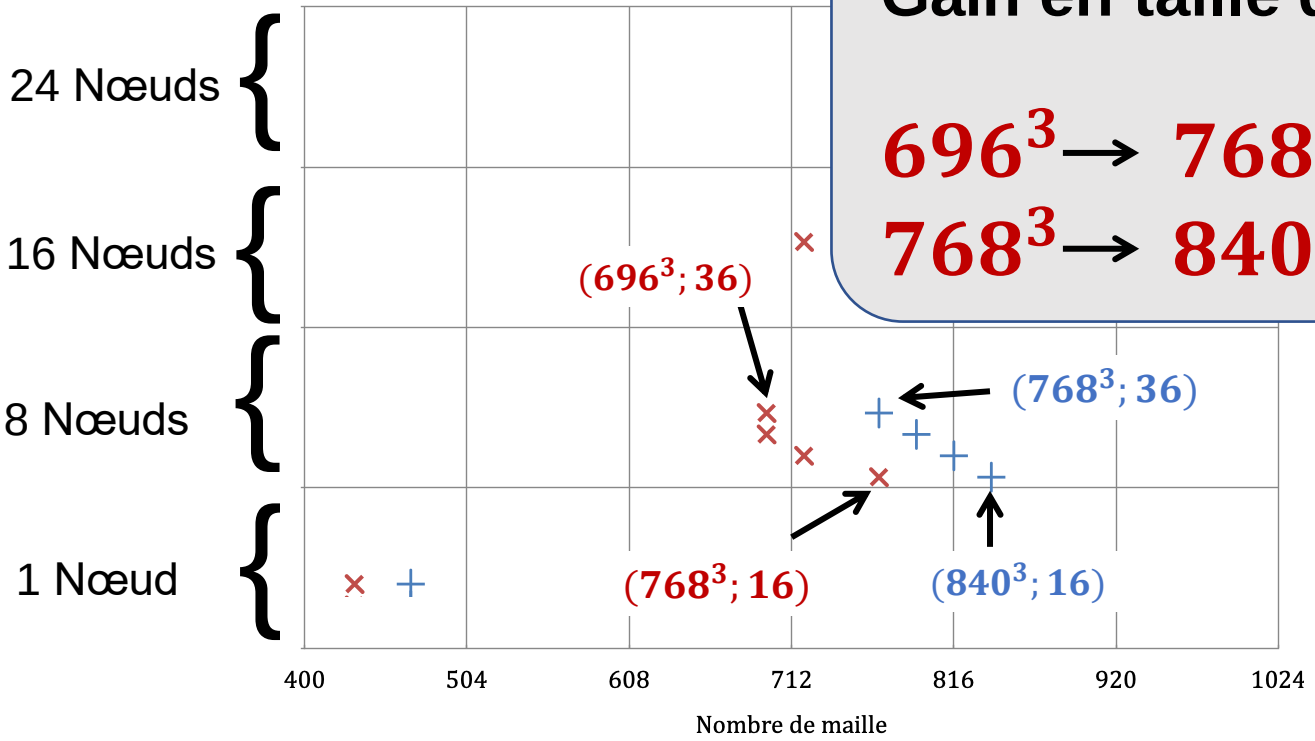

Evolution de la taille de maillage maximum

# 3.2 – Maillage maximum




Evolution de la taille de maillage maximum

# 3.2 – Maillage maximum



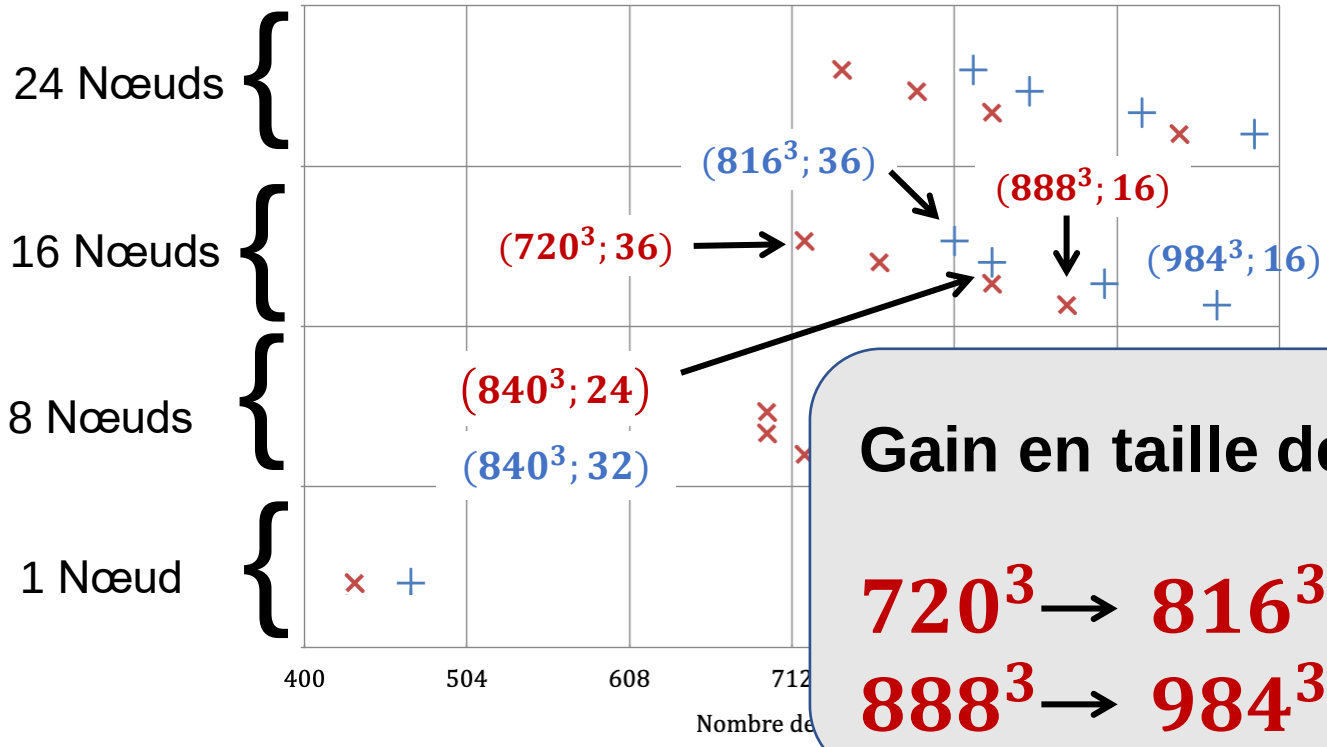

**Gain en taille de maillage:**

$696^3 \rightarrow 768^3$ [36 cores]

$768^3 \rightarrow 840^3$ [16 cores]

Evolution de la taille de maillage maximum

# 3.2 – Maillage maximum




Evolution de la taille de maillage maximum

**Gain en taille de maillage:**

$720^3 \rightarrow 816^3$ **[36 cores]**

$888^3 \rightarrow 984^3$ **[16 cores]**

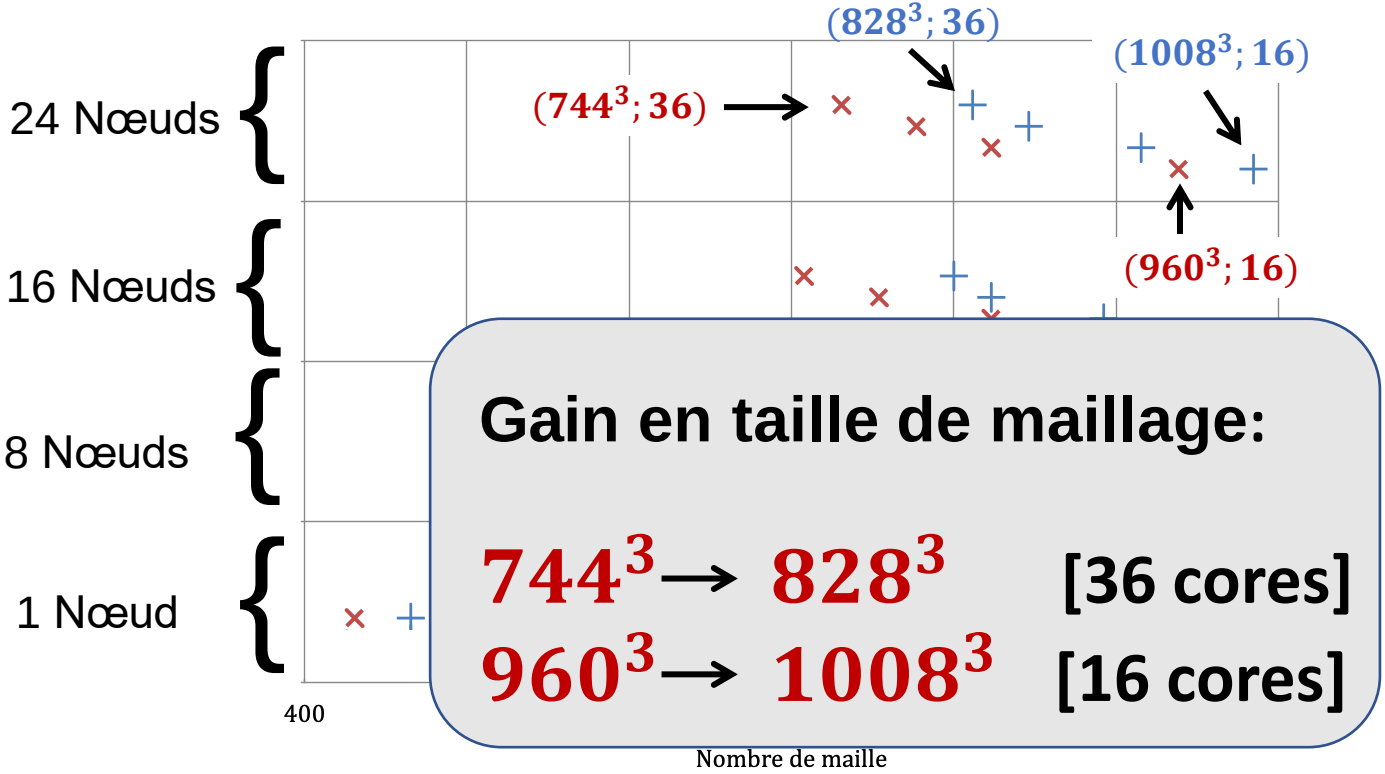
# 3.2 – Maillage maximum

**Légende :**

+ RMA

× Master

**(Maillage; Nb_Proc_MPI/Node)**



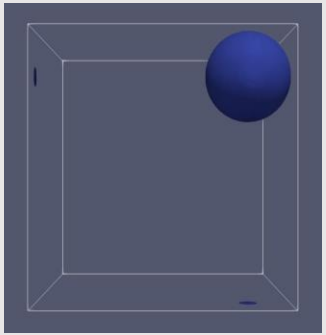

Evolution de la taille de maillage maximum

# 3.3 – Gain en mémoire et en temps de calcul

**Configuration des tests :**

Cas test : diphasique (VoF) - dynamique d'une BULLE animée par de la gravité
Maillage : cartésien orthogonal régulier
Condition aux limites : périodiques
JADIM version « RMA » vs JADIM version « master »
Nombre de processus : 1 Node, 8 Nodes, 16 Nodes, 24 Nodes

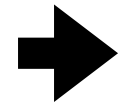
Simulation sur **2 pas de temps**



- **Mesure du gain en mémoire et en temps de calcul sur maillage de même taille**

# 3.3 – Gain en mémoire

| | RMA | MASTER |
|---|---|---|
| 1 Nœud<br>36 Processus | **Maillage $432^3$:** | |
| | 119 Go | 144 Go |



Gain en mémoire :
**18%**

# 3.3 – Gain en mémoire

| | | RMA | MASTER | |
|---|---|---|---|---|
| 1 Nœud<br>36 Processus | **Maillage $432^3$:** | 119 Go | 144 Go | Gain en mémoire :<br>**18%** |
| 8 Nœuds<br>36 Processus | **Maillage $696^3$:** | 101 Go | 137 Go | Gain en mémoire :<br>**26%** |

# 3.3 – Gain en mémoire

| | | RMA | MASTER | |
|---|---|---|---|---|
| 1 Nœud<br>36 Processus | **Maillage $432^3$:** | 119 Go | 144 Go | Gain en mémoire :<br>**18%** |
| 8 Nœuds<br>36 Processus | **Maillage $696^3$:** | 101 Go | 137 Go | Gain en mémoire :<br>**26%** |
| 16 Nœuds<br>36 Processus | **Maillage $720^3$:** | 85 Go | 125 Go | Gain en mémoire :<br>**32%** |

# 3.3 – Gain en mémoire

| | | RMA | MASTER | |
|---|---|---|---|---|
| 1 Nœud<br>36 Processus | **Maillage $432^3$:** | 119 Go | 144 Go | Gain en mémoire :<br>**18%** |
| 8 Nœuds<br>36 Processus | **Maillage $696^3$:** | 101 Go | 137 Go | Gain en mémoire :<br>**26%** |
| 16 Nœuds<br>36 Processus | **Maillage $720^3$:** | 85 Go | 125 Go | Gain en mémoire :<br>**32%** |
| 24 Nœuds<br>36 Processus | **Maillage $720^3$:** | 75 Go | 115 Go | Gain en mémoire :<br>**35%** |

# 3.3 – Gain en temps de calcul

| | | RMA | MASTER | |
|---|---|---|---|---|
| 8 Nœuds | **Maillage $768^3$:** | 36 cœurs<br>412 sec | 16 cœurs<br>1300 sec | **Gain en temps : x 3.55** |

# 3.3 – Gain en temps de calcul

| | RMA | MASTER | |
|---|---|---|---|
| 8 Nœuds | **Maillage $768^3$:**<br>36 cœurs<br>412 sec | 16 cœurs<br>1300 sec | **Gain en temps : x 3.55** |
| 16 Nœuds | **Maillage $840^3$:**<br>32 cœurs<br>424 sec | 24 cœurs<br>793 sec | **Gain en temps : x 1.87** |

# 3.4 – Validation des modules physiques

**Comparaison des solutions physiques CPU « sans RMA » vs CPU « avec RMA » : JADIM+Intel+PETSc**

| Module(s) associé(s) | Cas testés | Maillage | Solveur de Pression sous CPU |
|---|---|---|---|
| IBM | IBM_SEDIMENTATION | $128^3$ | PETSc |
| Diphasique VOF | BULLE_3D | $128^3$ | PETSc |
| Diphasique Level-set | PARASITE_3D | $216^3$ | PETSc |
| Turbulence LES | CANAL | $128^3$ | PETSc |
| LES+Thermique | CANAL2_THERM | $128^3$ | PETSc |
| IBM+DEM | RBD_3D_VOF | $160^3$ | PETSc |

# Conclusion

- Adapter la fonctionnalité de fenêtre à mémoire partagée de la norme MPI 3.0 à notre besoin de diminution de la mémoire
- Implémenter cette technique dans le code JADIM
- Valider cette parallélisation à de nombreux modules physiques du code

- **Diminution de la mémoire** consommée par JADIM d'environ **35%** **pour maillage de $720^3$**
- **Gain en temps de calcul** de **3,155** sur 8 nœuds grâce au repeuplage des noeuds

# Remerciements

Merci @**Team CALMIP** pour les heures de calculs sur le projet p19046
Merci @**Romain Casta, @Annaïg Pedrono** pour avoir initié ce travail
Merci @**Maxime Beauvillier** pour le stage qui a fourni les résultats