Institut du Développement et des Ressources en Informatique Scientifique

www.idris.fr

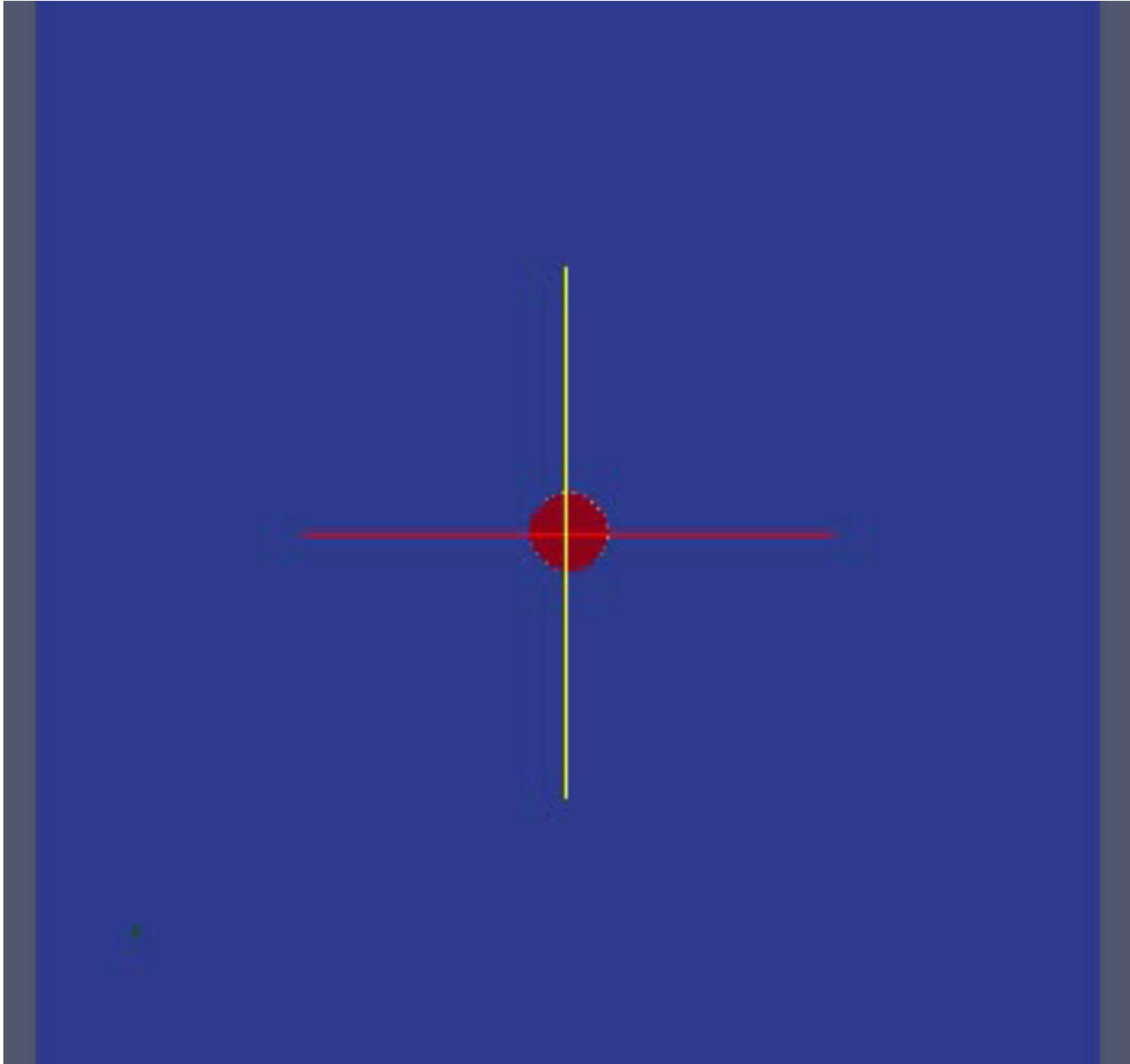
JDEV2013 - Calcul parallèle hybride avec MPI, OpenMP et OpenCL

HYDRO, de la version séquentielle à la version hybride MPI+OpenMP

Présentation du code HYDRO

- HYDRO est une version simplifiée du code d'astrophysique RAMSES.
- Code de mécanique des fluides (CFD), qui résout les équations d'Euler compressible de l'hydrodynamique en 2D.
- Méthode volume finis utilisant un schéma de Godunov d'ordre 2, avec résolution d'un problème de Riemann à chaque interface sur une grille régulière cartésienne 2D.
- 1500 lignes pour la version séquentielle F90. Disponible en Fortran ou en C.
- Facilement customizable (taille du domaine, nombre d'itération en temps, fréquence des IO, etc.), via un fichier d'input ASCII contenant des couples clées/valeurs.
- Par défaut, on va simuler une explosion de Sedov et suivre son évolution au cours du temps.

Simulation d’une explosion avec HYDRO



Équations et variables

Équations d'Euler pour un gaz idéal :

$$\partial_t U + \partial_x F(U) + \partial_y G(U) = 0$$

Avec U le vecteur des variables conservatives

$${}^{t}U = (\rho, \rho u, \rho v, E)$$

- ρ la densité
- u la vitesse suivant la première dimension x
- v la vitesse suivant la deuxième dimension y
- E l'énergie totale

Schéma numérique

Après discrétisation et intégration par la méthode des volumes finis, on obtient :

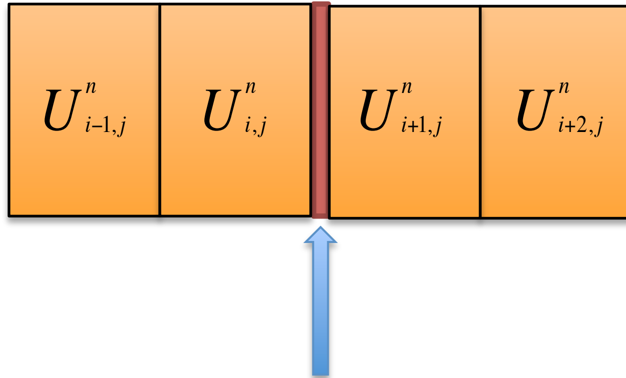
$$U_{i,j}^{n+1} = U_{i,j}^{n} + \frac{\Delta t}{\Delta x}\left(F_{i+1/2,j}^{n+1/2} - F_{i-1/2,j}^{n+1/2}\right) + \frac{\Delta t}{\Delta y}\left(G_{i,j+1/2}^{n+1/2} - G_{i,j-1/2}^{n+1/2}\right)$$

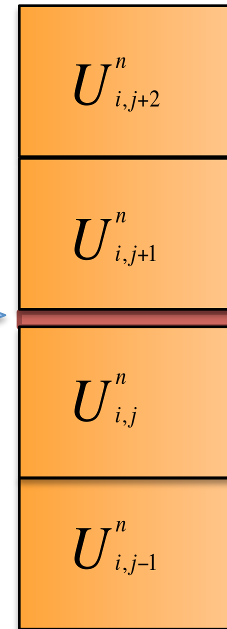
Où F et G sont les vecteurs flux suivant les deux dimensions.

Dans le code, le domaine discrétisé au n^{ieme} pas de temps est stocké dans le tableau uold(1:nx,1:ny,1:nvar)

$$U_{i,j}^{n} \approx uold(i,j,1:4)$$

Calcul du flux





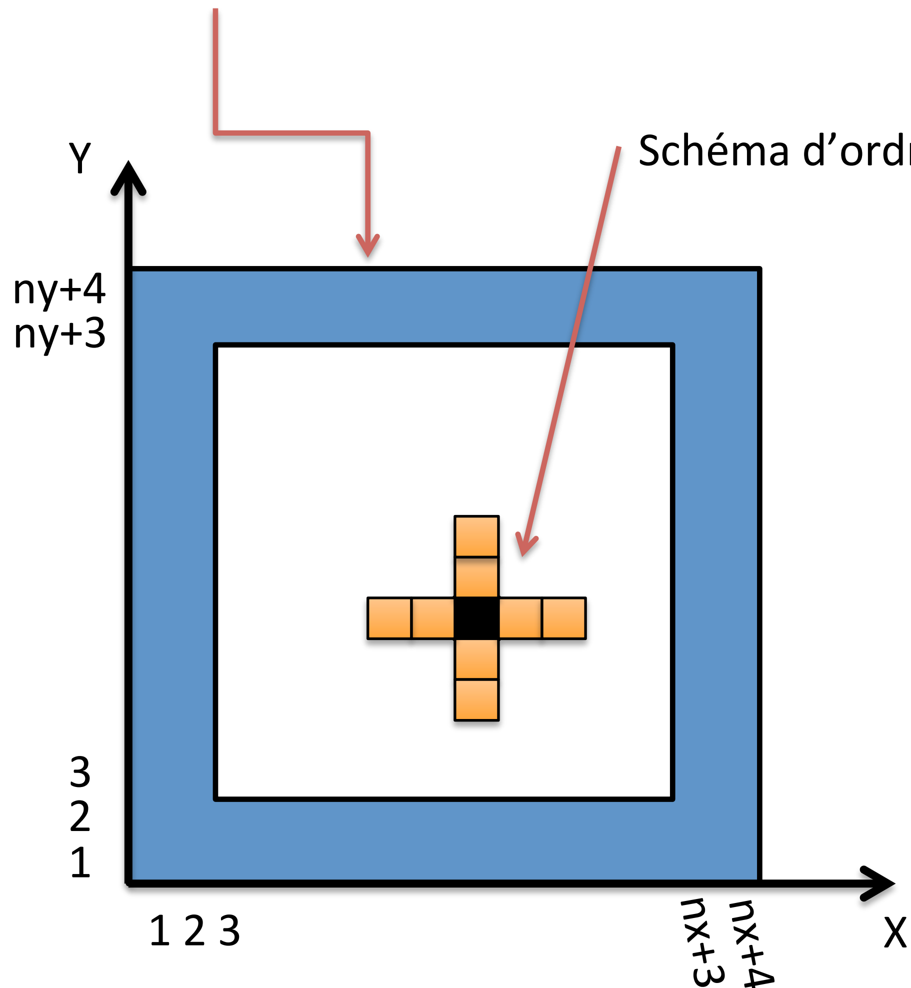
$$F^{n+1/2}_{i+1/2,j} = function\left(U^n_{i-1,j}, U^n_{i,j}, U^n_{i+1,j}, U^n_{i+2,j}\right)$$

$$G^{n+1/2}_{i,j+1/2} = function\left(U^n_{i,j-1}, U^n_{i,j}, U^n_{i,j+1}, U^n_{i,j+2}\right)$$

Dans le code, l'enchainement des subroutines *constoprim, trace, riemann* et *cmpflx* appelées depuis *godunov* permet de calculer ce flux…

HYDRO version séquentielle

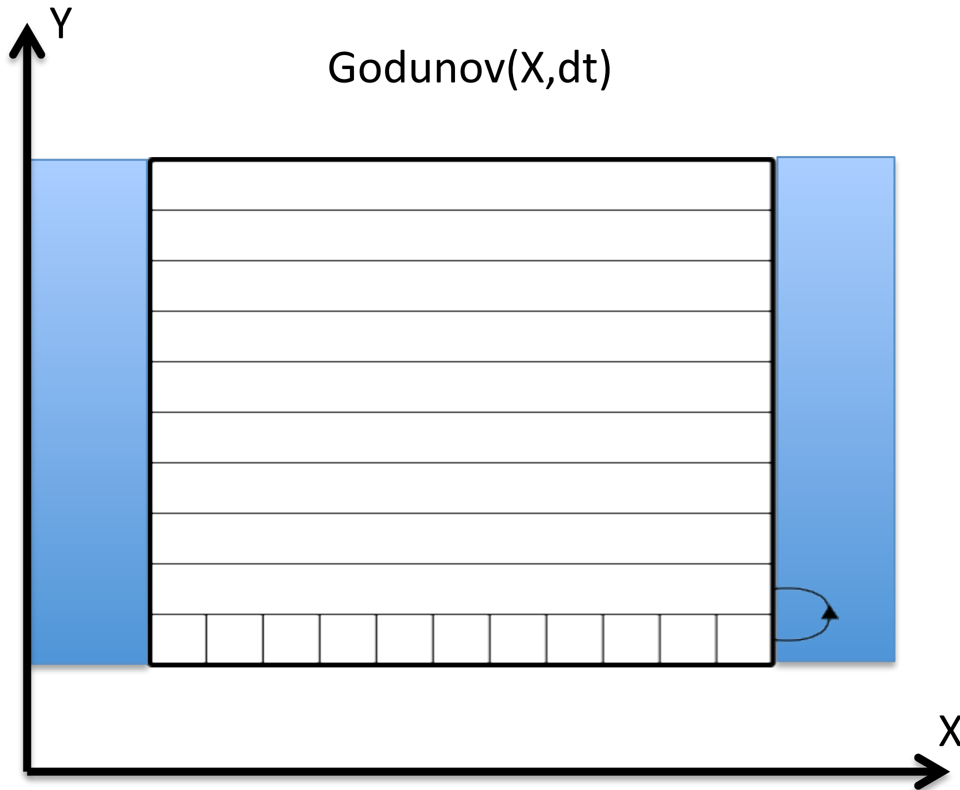
Mailles fantômes pour gérer les CL du domaine physique

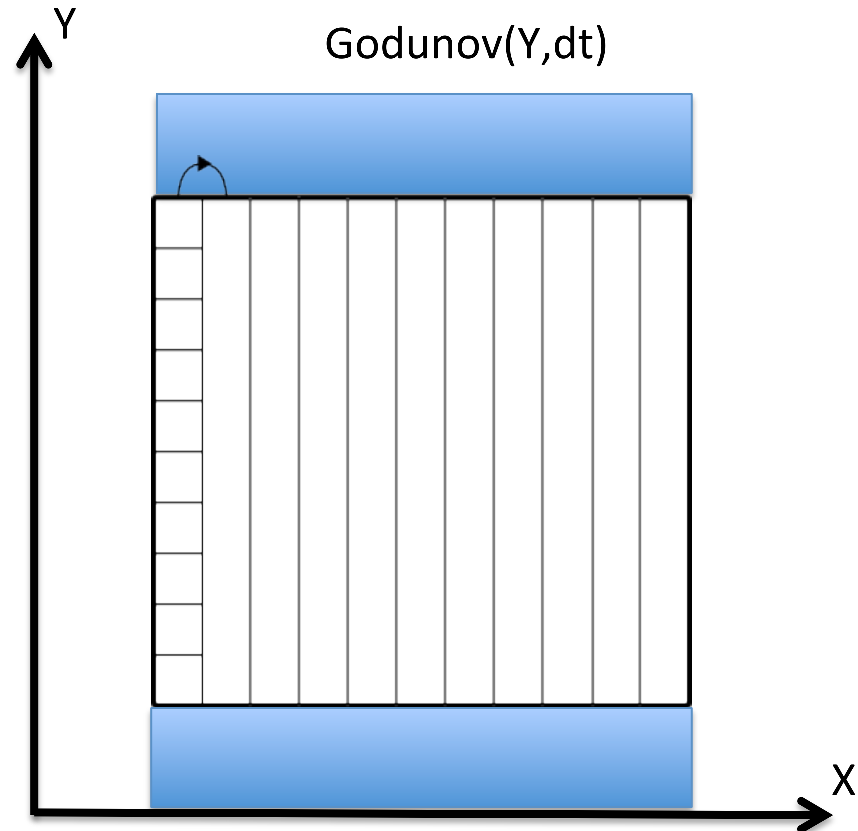


Algorithme HYDRO

initialize uold(:,:,:)
while *t<tend* **do**
dt=cmpdt(); // compute time step
if *n* %2 == 0 *step*
then
Godunov(X,dt); Godunov(Y,dt); // update uold
else
Godunov(Y,dt); Godunov(X,dt); // at t+dt
end if
t=t+dt
end while

HYDRO version séquentielle



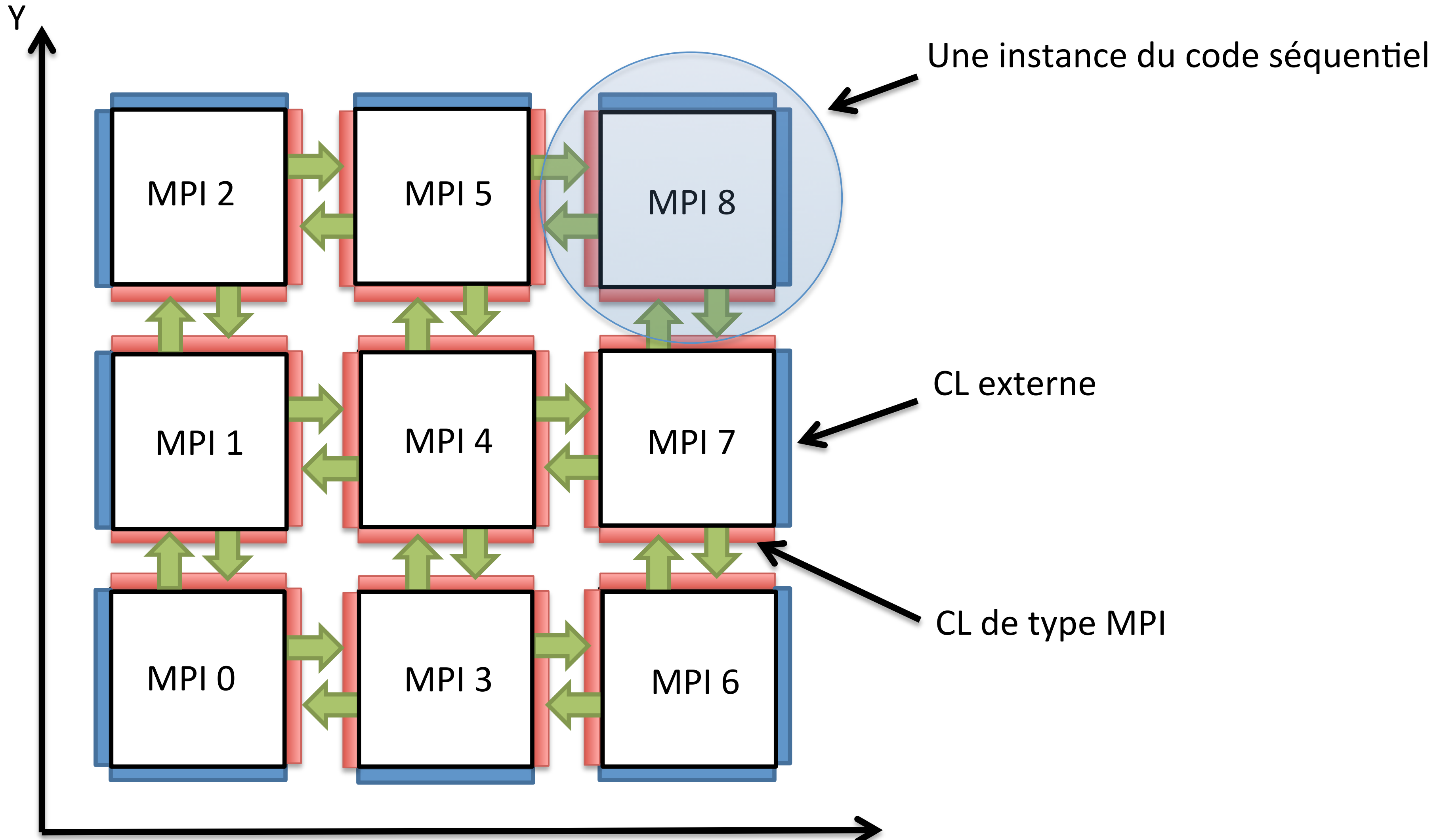


1. MAJ des CL pour les mailles fantômes verticales
2. Contribution des flux des interfaces verticales pour la MAJ de uold(:,:,:), ligne par ligne

1. MAJ des CL pour les mailles fantômes horizontales
2. Contribution des flux des interfaces horizontales pour la MAJ de uold(:,:,:), colonne par colonne

HYDRO version parallèle MPI

On reprend la version séquentielle qu'on duplique plusieurs fois et on ajoute un nouveau type de CL : la CL MPI (échange de valeurs avec ses voisins)



HYDRO version parallèle MPI

- Utilisation d'une topologie MPI 2D pour répartir les processus MPI dans chacune des directions et calculer la taille locale des sous-domaines associés à chaque processus MPI
- Détermination des voisins à chaque processus MPI (pour savoir à/de qui envoyer/recevoir !)
- Création de deux types dérivés MPI (un dans chaque direction) pour ne faire qu'un seul envoi/réception par voisin
- Le calcul du pas de temps nécessite une réduction pour calculer un maximum global sur la totalité des sous-domaines MPI

OpenMP *FineGrain* vs. *CoarseGrain*

- OpenMP *FineGrain*

Avantages	Inconvénients
Simplicité, approche incrémentale	Surcoût important qd granularité petite
Partage du travail et synchro implicite	Perf. et extensibilité limitée, pb NUMA
Équilibrage de charge dynamique	Certains algo. non parallélisables

- OpenMP *CoarseGrain*

Avantages	Inconvénients
Surcoûts limités même qd granularité petite	Approche non incrémentale et intrusive
Stratégie *"first touch"* implicite limitant pb NUMA	Synchronisation, partage du travail et équilibrage de charge entièrement à la charge du développeur
Perf. et extensibilité optimale	

Calcul de pi : *FineGrain* vs. *CoarseGrain*

Version *FineGrain*

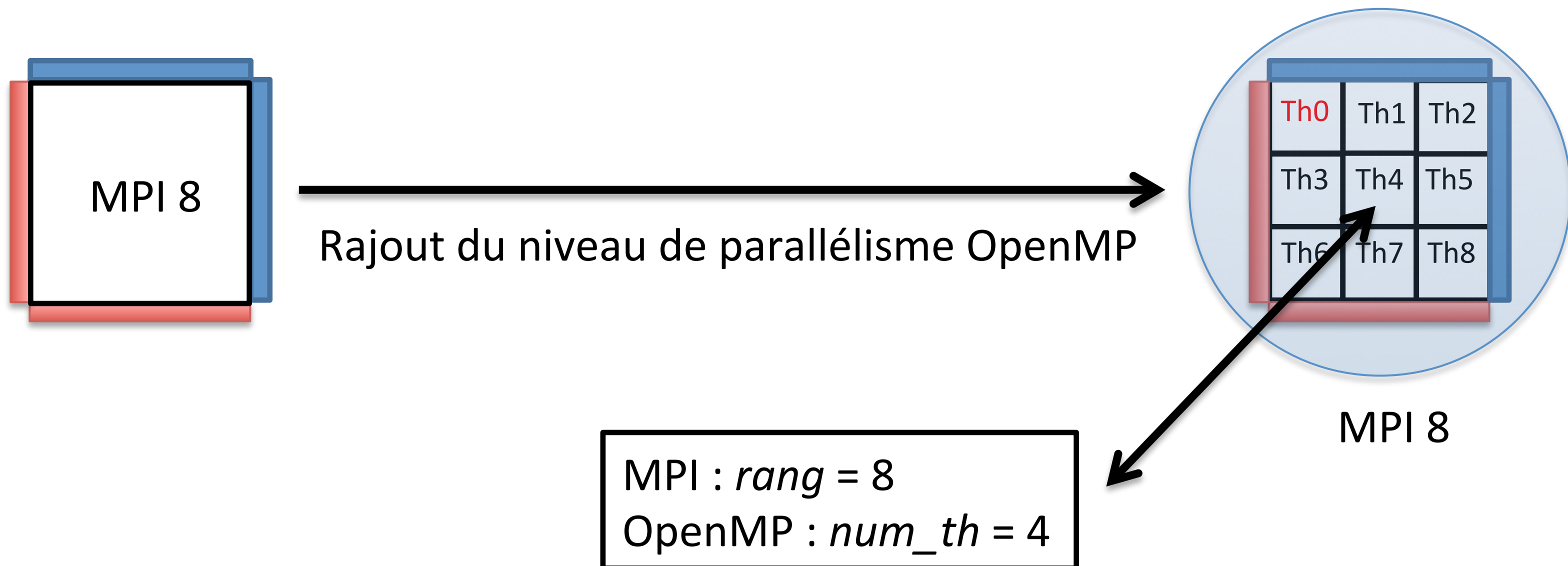
```
program pi
implicit none
integer, parameter :: n=30000000
real(kind=8) :: f, x, a, h, Pi_calcule
integer :: i
! Fonction instruction a integrer
f(a) = 4.0_8 / ( 1.0_8 + a*a )
! Longueur de l'intervalle d'integration.
h = 1.0_8 / real(n,kind=8)
! Calcul de Pi
Pi_calcule = 0.0_8
!$OMP PARALLEL DO PRIVATE(x) REDUCTION(+:Pi_calcule)
do i = 1, n
  x = h * ( real(i,kind=8) - 0.5_8 )
  Pi_calcule = Pi_calcule + f(x)
end do
!$OMP END PARALLEL DO
Pi_calcule = h * Pi_calcule
end program pi
```

Version CoarseGrain

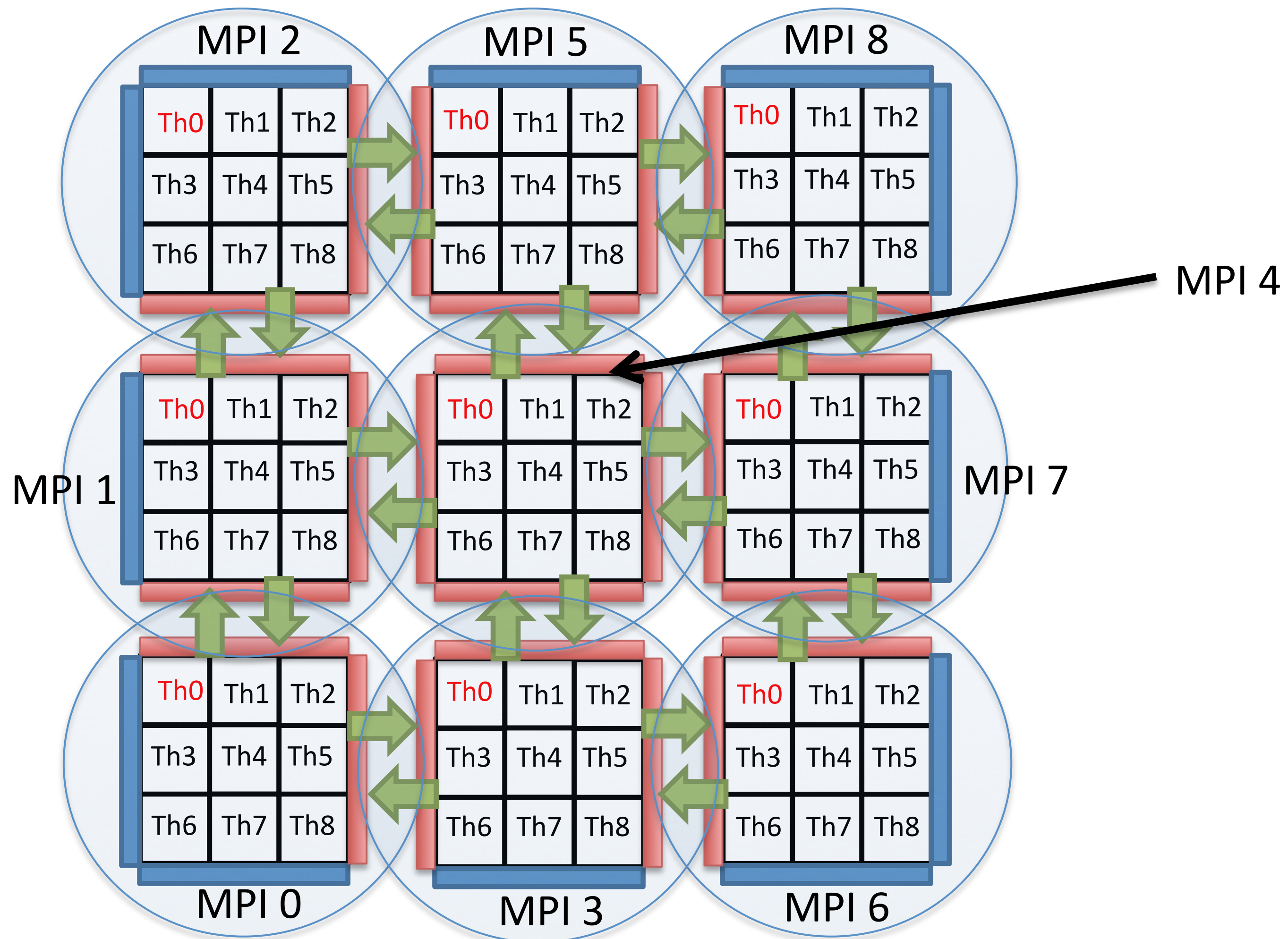
```
program pi
!$ use OMP_LIB
implicit none
integer, parameter :: n=30000000
real(kind=8) :: f, x, a, h, Pi_calcule, Pi_calcule_loc
integer :: i, iDeb, iFin, myOMPRank, nbOMPThreads
! Fonction instruction a integrer
f(a) = 4.0_8 / ( 1.0_8 + a*a )
! Initialisation de myOMPRank et nbOMPThreads
myOMPRank=0
nbOMPThreads=1
! Longueur de l'intervalle d'integration.
h = 1.0_8 / real(n,kind=8)
! Calcul de Pi
Pi_calcule = 0.0_8
Pi_calcule_loc = 0.0_8
!$OMP PARALLEL PRIVATE(x,myOMPRank) FIRSTPRIVATE(Pi_calcule_loc)
!$ myOMPRank = OMP_GET_THREAD_NUM()
!$ nbOMPThreads = OMP_GET_NUM_THREADS()
iDeb = 1+(myOMPRank*n)/nbOMPThreads
iFin = ((myOMPRank+1)*n)/nbOMPThreads
do i = iDeb, iFin
  x = h * ( real(i,kind=8) - 0.5_8 )
  Pi_calcule_loc = Pi_calcule_loc + f(x)
end do
!$OMP ATOMIC
Pi_calcule = Pi_calcule + Pi_calcule_loc
!$OMP END PARALLEL
Pi_calcule = h * Pi_calcule
end program pi
```

HYDRO version parallèle hybride

- Au sein de chaque processus MPI, on rajoute un niveau de parallélisme OpenMP (approche Coarse Grain de type décomposition de domaine)
- Support du multi-threading de type MPI_THREAD_FUNNELED, les communications sont faites par le thread maître (celui de rang 0) pendant que les autres threads attendent sur une barrière de synchronisation
- Chaque thread est identifié de façon unique par son numéro de processus MPI (*rang*) et par son numéro de thread (*num_th*)

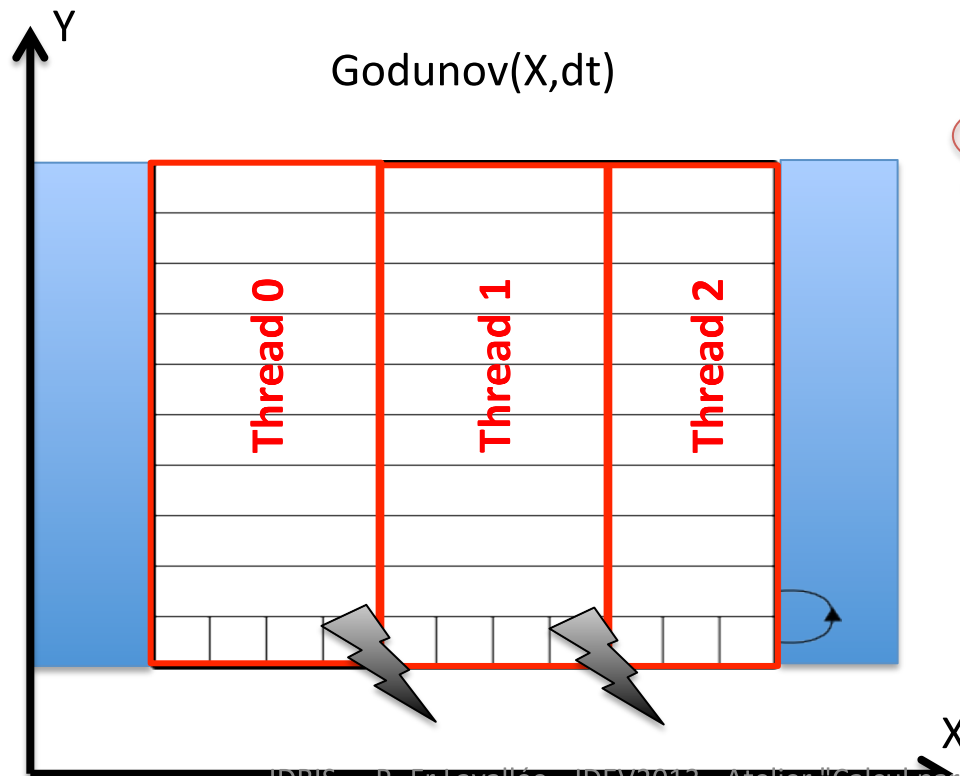


HYDRO version parallèle hybride



HYDRO version parallèle hybride

- Attention à la gestion des synchronisations (MPI et OpenMP) pour ne pas casser les dépendances et conserver ainsi la sémantique du code (i.e. obtenir les mêmes résultats que la versions séquentielle aux erreurs d'arrondis près…)
- Synchronisation de type exclusion mutuelle des threads pour calculer le pas de temps sur un sous-domaine MPI
- Synchronisation (MPI+OpenMP) avant de commencer la MAJ de uold(:,:,:)
- Synchronisation « fine » OpenMP nécessaire lors de la MAJ du vecteur de travail (ligne ou colonne suivant la direction X ou Y)



Condition de dépendance :
Pour une ligne donnée, le thread 0 ne peux écrire son résultat que lorsque le thread 1 à fini sa lecture. Au maximum, on ne pourra avoir qu'une ligne de décalage entre les différents threads !

Approche hybride MPI/OpenMP

- Approche pérenne, basée sur des standards reconnus (MPI et OpenMP), c'est un investissement à long terme.
- Les avantages comparés à l'approche pure MPI :
  - Meilleure extensibilité
    - réduction du nombre de messages MPI et du nombre de processus impliqués dans des communications collectives
    - augmentation de la granularité et meilleur équilibrage de charge.
  - Gain en performance à un nombre de coeur d'exécution fixe
    - Meilleure adéquation à l'architecture des calculateurs modernes (noeuds à mémoire partagée interconnectés, machines NUMA...), alors que MPI seul est une approche flat.
    - Optimisation de l'utilisation du réseau d'interconnexion.
  - Optimisation de la consommation de mémoire totale d'un facteur 2 à 5 suivant le type d'application (grâce à l'approche mémoire partagée OpenMP, gain au niveau des données répliquées dans les processus MPI et de la mémoire utilisée par la bibliothèque MPI elle-même).
  - Peut lever certaines limitations algorithmiques (découpage maximum dans une direction par exemple).
  - Approche bien adapté aux architectures qui nécessitent de lancer plusieurs threads par coeur (hyperthreading) pour utiliser efficacement les unités de calcul.

Résultats gain mémoire

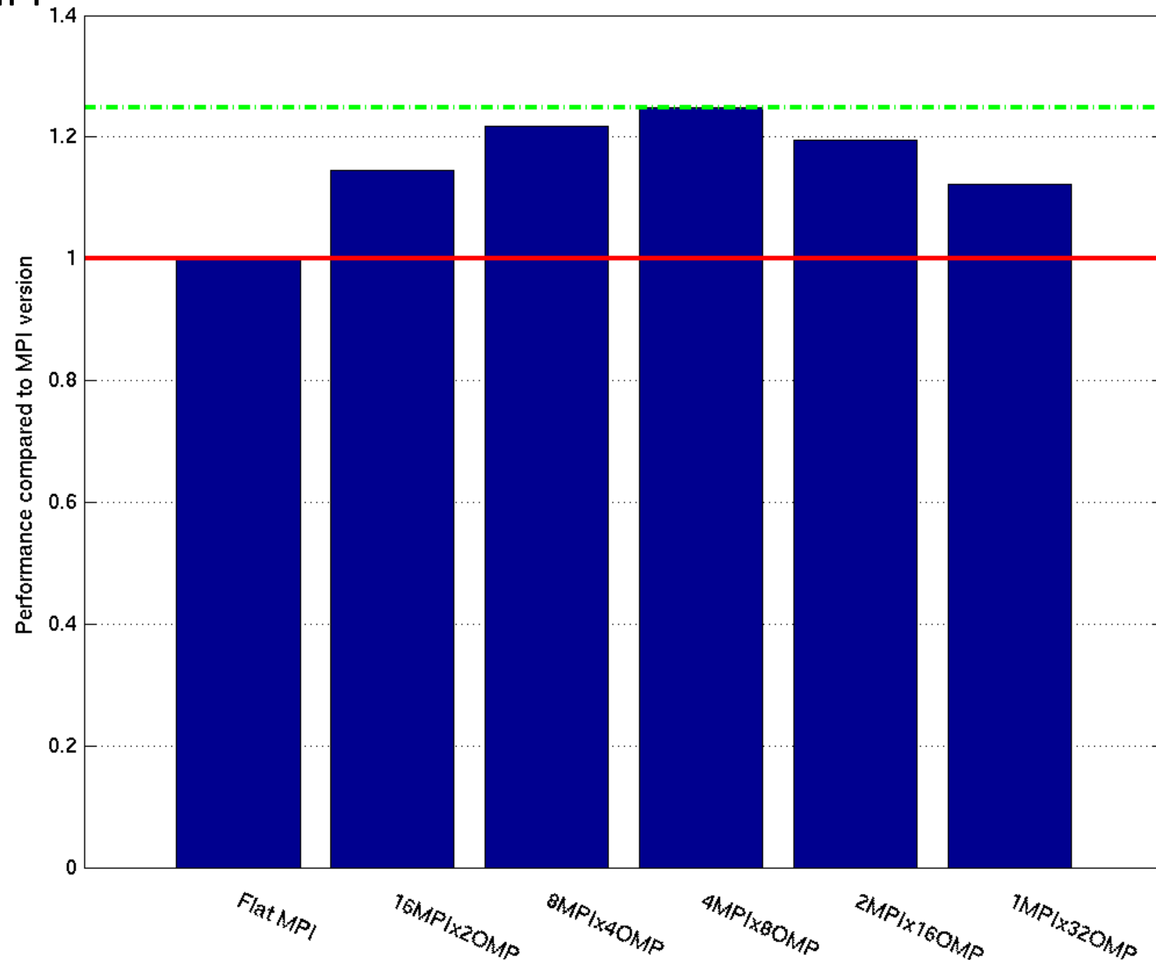
- Source : « Mixed Mode Programming on HECToR », Anastasios Stathopoulos,
- Machine cible : HECToR CRAY XT6 (1856 Compute Nodes, chacun composé de deux processeurs AMD 2.1 GHz à 12 coeurs se partageant 32 Go de mémoire)
- La mémoire par node est exprimée en Mo.

Code	Version pure MPI		Version hybride		Gain mémoire
	Nbre MPI	Mém./Node	MPI x Threads	Mém./Node	
CPMD	1152	2400	48 x 24	500	4.8
BQCD	3072	3500	128 x 24	1500	2.3
SP-MZ	4608	2800	192 x 24	1200	2.3
IRS	2600	2600	108 x 24	900	2.9
Jacobi	3850	3850	96 x 24	2100	1.8

Résultats performance

Machine cible : 2 noeuds IBM P6, 32 coeurs par noeud SMP
Domaine de taille : nx=100000 et ny=1000
À nombre de coeurs d'exécution constant (64 coeurs), on fait varier le nombre de threads OpenMP par processus MPI

MPI x OMP par noeud	Temps en seconde	
	Mono	64 coeurs
32 x 1	361.4	7.00
16 x 2	361.4	6.11
8 x 4	361.4	5.75
4 x 8	361.4	5.61
2 x 16	361.4	5.86
1 x 32	361.4	6.24



Résultats extensibilité

Machine cible : 10 racks IBM BG/P, 40960 coeurs, 4 coeurs par noeud SMP
Test de type *Strong Scaling* sur un domaine de taille nx=ny=40000
Mode *Dual* : version hybride avec 2 threads par processus MPI
Mode SMP : version hybride avec 4 threads par processus MPI