Le parallélisme dans LMDZ

Formation LMDZ 8-9 novembre 2011

Josefine Ghattas, Yann Meurdesoif

Voir aussi :
http://lmdz.lmd.jussieu.fr/developpeurs/notes-techniques/ressources/parallelisme-LMDZ.pdf

Parallélisme, pourquoi ?
- pour diminuer le temps de calcul d'une simulation. Chaque processus travail sur une partie du domaine global

Pour qui ?
- LMDZ peut tourner en séquentielle ou en parallèle. Sur les centre de calculs de l'IDRIS et du CCRT il faut tourner en parallèle.

Il existe deux grands standards normalisés : MPI et OpenMP

==> MPI : Le parallélisme en mémoire distribuée SPMD

- Le code à exécuter est répliqué sur l'ensemble des CPUs au sein d'un processus.

- Chaque processus s'exécute indépendamment et n'a pas accès à la mémoire des autres processus.

- Les échanges de données se font à travers une librairie d'échange de message qui utilise le réseau d'interconnexion entre les nœuds du calculateur.
  Les performances reposent sur la qualité du réseau d'interconnexion.

- Modèle de programmation adapté à tous les calculateurs du marché.

==> OpenMP : Le parallélisme à mémoire partagée SMP

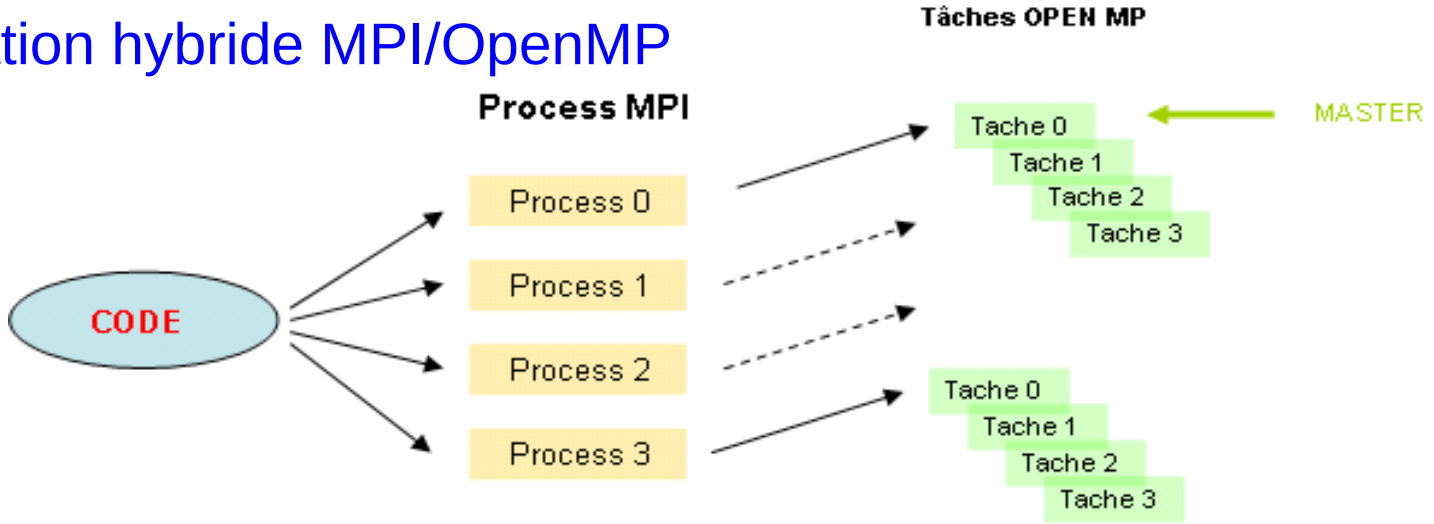
- Le parallélisme repose sur le principe du multithreading. Plusieurs threads s'exécutent concurremment au sein d'un processus.

- Chaque thread a un accès partagé à la mémoire globale du processus.

- Les boucles sont parallélisées à l'aide de directives.

- Modèle de programmation limité aux machines SMP, uniquement à l'intérieur d'un Nœud.

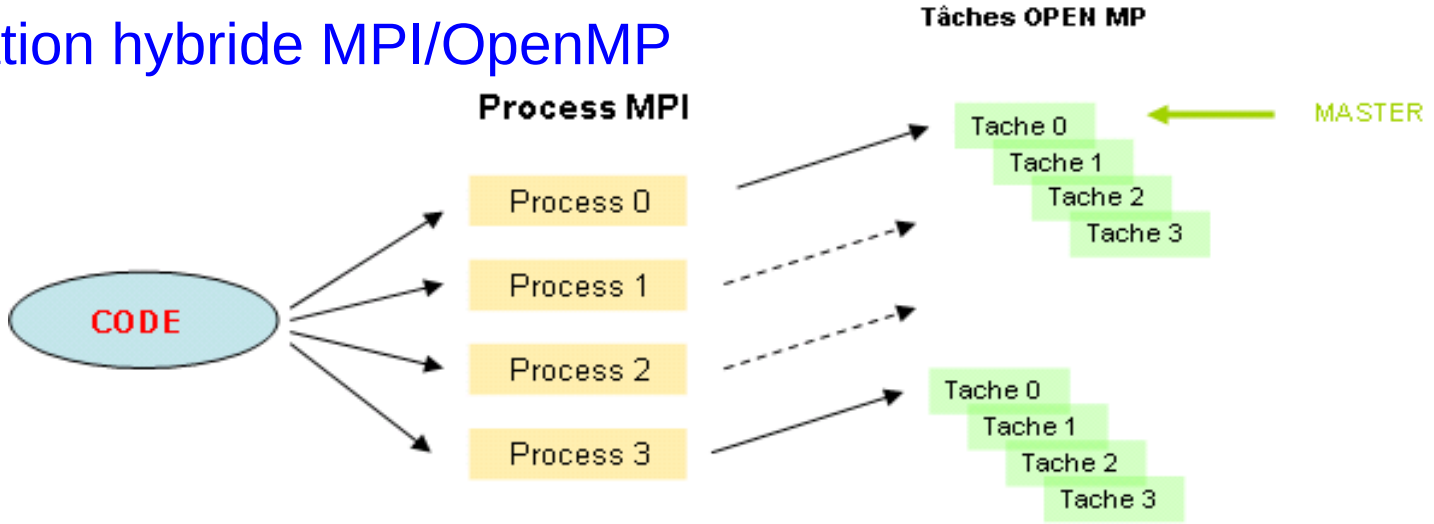
Programmation hybride MPI/OpenMP



- Chaque processus MPI lance des threads OpenMP dans son espace mémoire

- Dans LMDZ, parallélismes en MPI et OpenMP ne sont pas les mêmes.

Programmation hybride MPI/OpenMP



- Chaque processus MPI lance des threads OpenMP dans son espace mémoire

- Dans LMDZ, parallélismes en MPI et OpenMP ne sont pas les mêmes.

Stratégie de parallélisation différente entre la dynamique et la physique

- La partie dynamique de LMDZ

  Pas de temps très courts, beaucoup d'interaction entre les mailles voisines. Exige de nombreux échanges et synchronisations. Partie délicate de la parallélisation.

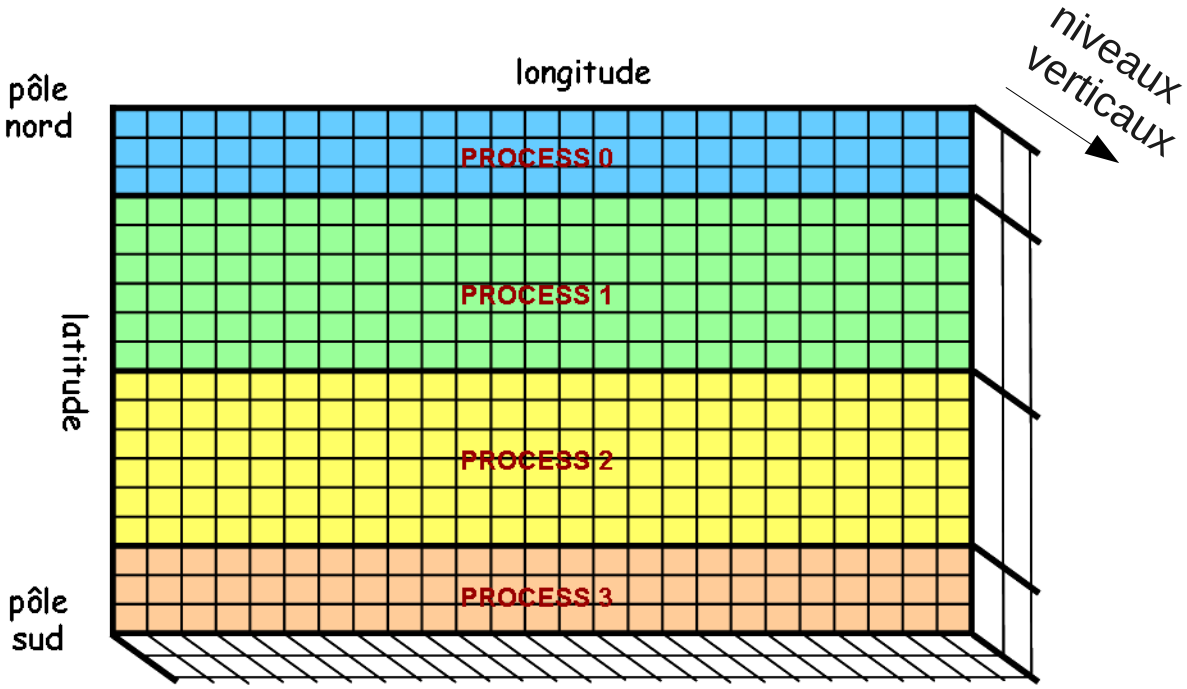
- La partie physique

  Pas de temps plus long, aucune interaction entre les colonnes d'atmosphères.

Parallélisme dans la partie dynamique

=> MPI

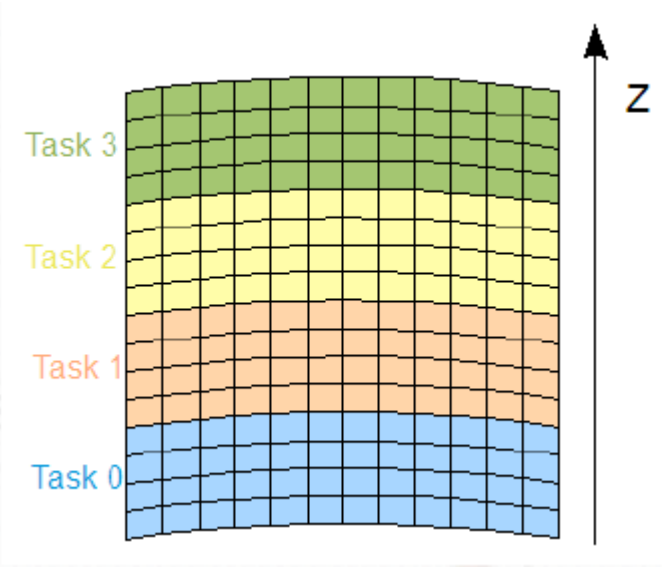
- Découpage par bande de latitude
- Minimum 3 bandes de latitude par processus MPI
- Utilisation de la fonction *adjust* pour optimiser la distribution des latitudes par processus, équilibrage de charge entre les processus.



Parallélisme dans la partie dynamique

=> OpenMP

- Découpage sur la verticale en plus



Parallélisme dans la partie physique

- La partie physique est chargée du calcul des phénomènes atmosphériques interagissant au sein d'une même colonne d'atmosphère : rayonnement, convection, interaction des couches limites, etc. …

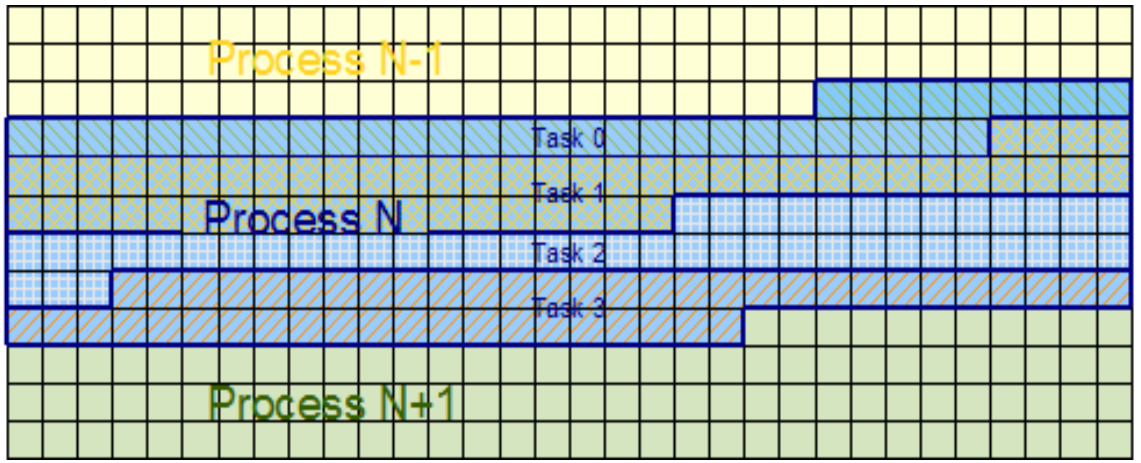
- Les différentes colonnes d'atmosphère n'interagissent pas entre-elles.

- La stratégie de parallélisation consiste donc à distribuer le calcul des colonnes d'atmosphère sur les différents processeurs.

- Grille de la physique : klon_glo points géographique sur klev niveaux verticaux.
  1er point => pôle nord, dernier point (klon_glo) => pôle sud.

Parallélisme dans la partie physique

- Les points du domaine global sont distribués aux différents processus MPI.
- Les points d'un domaine MPI sont distribués à chacun des threads OpenMP tournant à l'intérieur du processus :
  - Le domaine global : klon_glo mailles de surface
  - Le domaine MPI : klon_mpi mailles de surface : Σ klon_mpi = klon_glo
  - Le domaine OpenMP : klon_omp mailles de surface : Σ klon_omp = klon_mpi
- La taille du domaine local klon est un alias de klon_omp. klon est diffèrent sur chaque processus.



Définitions des paramètres du domaine

Grille globale : module mod_grid_phy_lmdz

- klon_glo : nombre de mailles sur l'horizontale du domaine globale (grille 1D)
- nbp_lon : nombre de points en longitude (grille 2D) = iim
- nbp_lat : nombre de points en latitude (grille 2D) = jjm+1
- nbp_lev : nombre de niveaux verticaux = klev ou llm

Grille MPI : module mod_phys_lmdz_mpi_data

- klon_mpi : nombre de points sur le domaine mpi local.
- klon_mpi_begin : indice de départ du domaine sur la grille 1D globale.
- klon_mpi_end : indice de fin du domaine sur la grille 1D globale.
- ii_begin : indice en longitude du début du domaine (grille 2D globale).
- ii_end : indice en longitude de la fin du domaine (grille 2D globale).
- jj_begin : indice en latitude de début de domaine (grille 2D globale).
- jj_end : indice en latitude de fin de domaine (grille 2D globale).
- jj_nb : nombre de bande de latitude = jj_end-jj_begin+1
- Is_north_pole : .true. si le processus possède le pôle nord.
- is_south_pole : .true. si le processus possède le pôle sud.
- is_mpi_root : .true. si processus MPI maître.
- mpi_rank : rang du processus MPI.
- mpi_size : nombre de processus MPI.

Grille OpenMP : sous-grille du domaine MPI

- klon_omp : nombre de points sur le domaine OpenMP.
- klon_omp_begin: indice de début de domaine OpenMP par rapport au domaine MPI.
- klon_omp_end : indice de fin de domaine OpenMP.
- is_omp_root : .true. tâche maîtresse OpenMP.
- omp_size : nb tâche OpenMp à l'intérieur du processus.
- omp_rank : rang de la tâche OpenMP.

Comment coder dans la partie physique ?

• Rien ne change si :
    il n'y a pas d'interaction entre les colonnes d'atmosphère
    Il n'y a pas de calcul de moyenne global ou zonal
    Il n'y a pas de lecture ou d'écriture de fichiers

• Seul impératif pour l'OpenMP : toutes les variables en SAVE ou les « common » doivent se trouver dans une clause !$OMP THREADPRIVATE, par exemple :

```
REAL, SAVE :: save_var
!$OMP THREADPRIVATE(save_var)
```

• Les tableaux sont alloués en taille local klon, sachant que klon est différent sur les différents processus. Par exemple :

```
ALLOCATE (myvar(klon))
```

NB! myvar(1) ou myvar(klon) ne représente ni le pôle sud, ni le pôle nord, sauf exception.

• Utiliser les variables booléens, is_north_pole et is_south_pole si on a besoin d'un traitement spécifique sur les pôles

Comment coder dans la partie physique ?

- Attention si :
  - interaction entre les colonnes d'atmosphère
  - calcul de moyenne globale ou zonale
  - lecture ou écriture de fichiers

=>> Pour ces cas, il faut faire des transferts entre les différents processus MPI et les tâches OpenMP.

Le transfert des données entre tâches/processus

Nécessité de transférer des données entre processus et/ou entre tâches pour les Entrées/Sorties ou les interfaces vers les autres codes.

Toutes les routines de transfert sont encapsulées et gèrent de façon transparente les transferts entre les processus MPI et entre les tâches OpenMP.

Inclusion du module : mod_phys_lmdz_transfert_para

Les interfaces de transfert acceptent indifféremment les principaux types de base :

- REAL
- INTEGER
- LOGICAL
- CHARACTER : uniquement le broadcast

Les interfaces acceptent indifféremment les champs jusqu'à 4 dimensions.

Le transfert des données (suite)

Broadcast : le processeur maître duplique ses données sur les autres processus/tâches.

Indépendant des dimensions de la variable

CALL bcast(var)

Scatter : la tâche maîtresse possède un champ sur la grille globale (klon_glo) qu'elle distribue sur la grille locale (klon).

La 1ère dimension du champs global doit être klon_glo, et celle du champ local klon

CALL scatter(field_glo,field_loc)

Gather : un champ défini sur la grille locale (klon) est rassemblé sur la grille globale de la tâche maîtresse (klon_glo).

La 1ère dimension du champ global doit être klon_glo, et celle du champ local klon

CALL gather(field_loc,field_glo)

Scatter2D : même chose que Scatter sauf que le champ global est défini sur la grille 2D : nb_lon x nbp_lat.

La 1ère et 2ème dimension du champs global doit être (nbp_lon,nbp_lat), et celle du champ local klon

CALL scatter2D(field2D_glo,field1D_loc)

Gather2D : rassemble les données sur la grille 2D de la tâche maîtresse.

CALL gather2D(Field1D_loc,Field2D_glo)

Cas particulier

La lecture des fichiers de paramètres de la physique : physiq.def et config.def

- Effectué dans conf_phys.F90
- La tâche maître lit la valeur puis la duplique sur les autres tâches.

La lecture des champs du fichier de démarrage : startphy.nc

- Effectué dans phyetat0
- La tâche maître sur le processus root(master) lit le champ sur la grille globale puis le redistribue sur la grille locale à l'aide d'un scatter
- Encapsulé dans la routine get_field du module iostart

L'écriture des champs dans le fichier de démarrage : restartphy.nc

- Effectué dans phyredem
- Encapsulé dans la routine put_field du module iostart. put_field effectuera d'abord un gather pour rassembler les champs locaux dans un champs global. Ensuite la tâche maîtresse sur le processus master écrit le champs dans le fichier restartphy.nc

Exemple d'ouverture d'un fichier, extrait du fichier phylmd/read_map2D.F90

```
USE dimphy
USE netcdf
USE mod_grid_phy_lmdz
USE mod_phys_lmdz_para
...

REAL, DIMENSION(nbp_lon,nbp_lat) :: var_glo2D
REAL, DIMENSION(klon_glo)        :: var_glo1D
REAL, DIMENSION(klon)            :: varout

! Read variable from file. Done by master process MPI and master thread OpenMP
  IF (is_mpi_root .AND. is_omp_root) THEN
     NF90_OPEN(filename, NF90_NOWRITE, nid)
     NF90_INQ_VARID(nid, varname, nvarid)

     start=(/1,1,timestep/)
     count=(/nbp_lon,nbp_lat,1/)
     NF90_GET_VAR(nid, nvarid, var_glo2D,start,count)
     NF90_CLOSE(nid)

     ! Transform the global field from 2D to 1D
     CALL grid2Dto1D_glo(var_glo2D,var_glo1D)
  ENDIF

! Scatter gloabl 1D variable to all processes
  CALL scatter(var_glo1D, varout)
```

L'écriture des fichiers histoires d'IOIPSL et reconstruction

- Chaque processus MPI écrit la partie de son domaine dans un fichier distinct. On obtient autant de fichiers histmth_00XXX.nc que de processus MPI.

- Le domaine du fichier IOIPSL est définit au moment de l'appel à histbeg, encapsulé dans histbeg_phy

- Les données sont rassemblées sur la tâche de rang 0 de chaque processus -> Chaque processus MPI appelle la routine histwrite d'IOIPSL, encapsulé dans histwrite_phy

L'écriture des fichiers histoires d'IOIPSL et reconstruction

On doit reconstruire les données dans un fichier globale. Pour ceci, on utilisera le programme rebuild. Par exemple :

```
rebuild -o histmth.nc histmth_00*.nc
```

- Rebuild est un programme distribué avec IOIPSL
- Rebuild se compile comme IOIPSL
- Sur les centres de calculs IDRIS et CCRT, rebuild est disponible déjà compilé dans un répertoire avec des outils communs :

IDRIS

Ulam	/home/rech/psl/rpsl035/bin/rebuild
Vargas	/homegpfs/rech/psl/rpsl035/bin/rebuild
Brodie	/home/rech/psl/rpsl035/SX/bin/rebuild

CCRT

Cesium	/home/cont003/p86ipsl/CESIUM/bin/rebuild
Titane	/home/cont003/p86ipsl/X64/bin/rebuild
Mercure	/home/cont003/p86ipsl/SX8/bin/rebuild
Platine	/home/cont003/p86ipsl/IA64/bin/rebuild

Exemple d'un job pur MPI sur Vargas

>cat Job.ksh

```
#!/usr/bin/ksh
# ######################
# ##   VARGAS IDRIS   ##
# ######################
# @ job_name = test
# @ job_type = parallel
# @ output = Script_output
# @ error =  Script_output_error
# Nombre de processus demandes
# @ total_tasks = 32
# Temps CPU max. par processus MPI hh:mm:ss
# @ wall_clock_limit = 0:30:00
# Memoire max. utilisee par processus
# @ data_limit = 3.2gb
# Memoire stack demandee
# @ stack_limit = 0.3gb,0.3gb
# Pas d OpenMP
# @ resources = ConsumableCpus(1)
# Fin de l entete
# @ queue

./lmdz.x   > out_lmdz.x 2>&1
```

>llsubmit Job.ksh

libIGCM

- l'entête du job, certain paramètres à modifier : mémoire + temps limite

- reconstructions lancées automatiquement à posteriori

- 16 novembre : formation modipsl + libIGCM

En pratique / résumé

► Dans la physique, les mailles ne doivent pas communiquer entre elles. Les variables en SAVE sont aussi déclarer en THREADPRIVATE.

► On apprend plus si on veut programmer dans la partie dynamique

► On compile en mode parallèle mpi ou mpi_omp
makelmdz_fcm -parallel mpi .....
makelmdz_fcm -parallel mpi_omp

► On lance le job en batch, sur plusieurs processus selon la taille de la grille.
Nombre maximum de processus MPI = nombre de points en latitude / 3

► Pour optimiser le découpage sur les différents processus MPI, premier mois on met adjust=y dans run.def. On obtient un fichier bands_resol_Xprc.dat qu'on ré-utilisera pour la reste de la simulation.

► On reconstruit les fichiers histoires : rebuild -o histmth.nc histmth_00*.nc