ORCHIDEE Parallélisation Mpi OpenMP

La parallélisation du modèle couplé LMDZOR est disponible avec la configuration LMDZOR_v5.2 moyennant quelques modifications dans LMDZ (rev 1729)

1. Mettre à jour LMDZ avant rev 1905

```
cd modeles/LMDZ/libf/phylmd
rm mod_synchro_omp.F90
rm mod_phys_lmdz_omp_transfert.F90
svn update -f 1905 mod_synchro_omp.F90
svn update -r 1905 mod_phys_lmdz_omp_transfert.F90
```

2. compilation de LMDZOR et ORCHIDEE Offline

Pour compiler il faut modifier modipsl/util/AA_make.gdef :

```
#-Q- curie  F_C = mpif90 -c -cpp -openmp
(…)
#-Q- curie   F_O = -DCPP_PARA -O3   -DCPP_OMP -openmp-threadprivate compat $(F_D) $(F_P) -I$(MODDIR) -module $(MODDIR)  -fp-model precise
```

-openmp-threadprivate compat permet l'utilisation des pointeurs dans le routage (routing.f90)

ATTENTION : il est impératif que LMDZ, IOIPSL et ORCHIDEE soient compilés avec les mêmes options. Pour LMDZ vous devez donc vérifier le fichier modeles/LMDZ/arch/arch-***.fcm correspondant à votre machine. La ligne *PROD_FFLAGS* doit être la suivante.

```
%PROD_FFLAGS        -O3 -fp-model precise -openmp-threadprivate compat
```

Il faut ensuite créer les différents makefile avec la commande modipsl/util/ins_make. Et indiquer que l'on travaille en mpi_omp à lmdz :

```
    (cd ../../modeles/LMDZ; ./makelmdz_fcm -d $(RESOL_LMDZ) -cosp true -v true -parallel mpi_omp -arch $(FCM_ARCH) gcm ; cp bin/gcm_$(RESOL_LMDZ)_phylmd_para_orch.e ../../bin/gcm.e ; )
```

3. Lancement d'une première simulation

Dans config.card il faut modifier la ligne correspondant à l'atmosphère dans la rubrique « Executable »
ATM= (gcm.e, lmdz.x, 24MPI, 2OMP)

et ensuite après création du job il faut modifier l'entête de celui-ci.
Pour Curie :

```
#MSUB -n 24 # reservation des processeurs pour le job
#MSUB -c 2                  → nombre de threads OMP
#MSUB -N 3                  → nombre de nœuds de calcul = nbMPI x nbOMP /16 = 24 x 2 / 16
(…)
BATCH_NUM_PROC_TOT=48  → nombre de cœurs : nbMPI x nbOMP = 24 x 2
```

4. Changements dans le code de ORCHIDEE

4.1. Interdits

Il ne faut surtout pas utiliser en début de ligne l'ensemble de caractères «!$ » qui est réservé à l'openmp. Ces deux caractères étaient beaucoup utilisés dans les versions précédentes de ORCHIDEE pour démarrer une ligne de commentaires.

4.2. Code – principes de base

- Toutes les variables déclarées dans des modules sont des variables de type « SAVE »
- Toutes les variables codées en « SAVE » doivent être déclarées en « THREADPRIVATE » . Cette règle ne s'applique pas dans le répertoire « src_parallel ».

```
  INTEGER(i_std), ALLOCATABLE, SAVE, DIMENSION (:) :: indexveg
!$OMP THREADPRIVATE(indexveg)
  INTEGER(i_std), ALLOCATABLE, SAVE, DIMENSION (:) :: indexlai
!$OMP THREADPRIVATE(indexlai)
```

- Sauf si l'on travaille sur le processeur maître mpi/omp tous les appels aux routines ioipsl sont suffixées par un « _p », elles sont définies dans le module src_parallel/ioipsl_para.F90

```
 CALL histwrite_p(hist_id, 'beta', kjit, vbeta, kjpindex, index)
 CALL ioconf_setatt_p('UNITS', '-')
CALL getin_p('FORCE_CO2_VEG',fatmco2)
```

Ces routines sont codées dans src_parallel/ioipsl_para.f90

- Le processeur maître mpi/omp est repéré par la variable « is_root_prc ». Elle est déclarée dans le module src_parallel/mod_orchidee_para.F90
- Le thread maître est repéré par la variable « is_omp_root ». Elle est déclarée dans le module src_parallel/mod_orchidee_omp_data.F90
- Le processeur maître mpi est repéré par la variable « is_mpi_root ». Elle est déclarée dans le module src_parallel/mod_orchidee_mpi_data.F90
- Les routines de transfert de données au sein des domaines de parallélisation ont été réécrite de telle manière qu'un call à la routine gather appelle d'abord un gather_omp pour réunir la variable sur le thread principal, puis un gather mpi pour réunir la variable sur le processeur principal. Idem pour les scatter qui appellent d'abord un scatter mpi pour redistribuer une variable sur tous les processeurs puis un scatter_omp pour les redistribuer sur les différents

thread. Remarque : pour les parties du code encore en MPI (intersurf_1d, 2d, et dim2_driver) on appelle directement les gather et scatter mpi (ex : gather2d_mpi).

Note : si vous avez besoin de rajouter des barrières OMP il faut utiliser la routine : **barrier2_omp** qui double l'appel à OMP_BARRIER et permet de contrer des bugs rencontrés dans certaines configurations.

```
call barrier2_omp()
```

5. Routines gérant la parallélisation dans Orchidee

Les modules de parallélisations sont systématiquement doublée :

- mod_orchidee_mpi_data.F90 / mod_orchidee_omp_data.F90 → initialisation de la parallélisation pour le mpi / omp
- mod_orchidee_mpi_transfert.F90 / mod_orchidee_omp_transfert.F90 → routines de transferts de données inter-process / inter_threads pour mpi / omp

Et ils sont drivés par des modules « génériques » :

- mod_orchidee_para.F90
- mod_orchidee_transfert_para.F90

Systématiquement nous appellerons la routine générique qui en fonction de la parallélisation retenue lors de la compilation appellera la routine nécessaire.

5.1. Initialisation de la parallélisation pour le couplage avec LMDZ

La partie initialisation des domaines de parallélisation a été déportée dans une routine init_intersurf appelée depuis la routine surf_land_orchidee de LMDZ/libf/phylmd/surf_land_orchidee_mod.F90.

Dans init_intersurf la parallélisation est initialisée à travers l'appel à la routine *init_orchidee_data_para* qui elle même appellera *init_orchidee_omp_data* et *init_orchidee_mpi_data.*

Dans ce cas là les informations nécessaires à l'initialisation de la parallélisation viennent directement de LMDZ (le communicator mpi et les données sur les threads omp).

Remarque : la routine init_intersurf n'est appelée que par les threads ayant des points de terre sur la grille globale.

5.2. Initialisation de la parallélisation pour Orchidee offline

Contrairement au couplage avec LMDZ dans lequel la parallélisation est définie par celle de l'atmosphère, il faut définir la parallélisation depuis zéro. Pour cela les routines utilisées sont :

init_orchidee_para qui appelle *init_orchidee_omp* et *init_orchidee_mpi.*

La partie omp n'est pas encore complète. Pour la partie mpi les appels classiques aux routines mpi sont réalisés pour la définition du communicator.

```
CALL MPI_INIT(ierr)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD,mpi_size,ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD,mpi_rank,ierr)
```

```
is_ok_mpi=.TRUE.
COMM=MPI_COMM_WORLD
```