Intégration de l'expérience COSMOMC dans EGEE

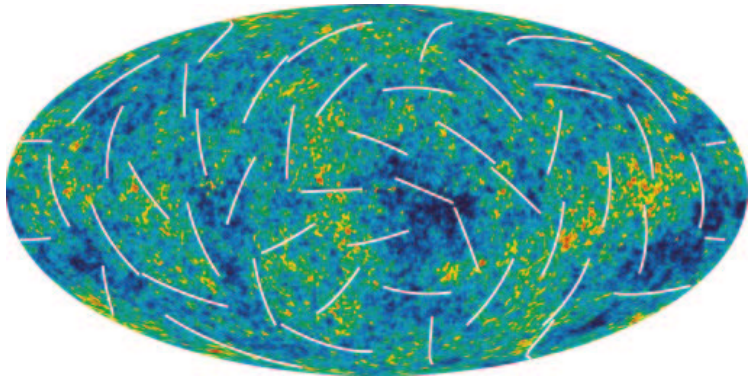
Julien LESGOURGUES (LAPTH)

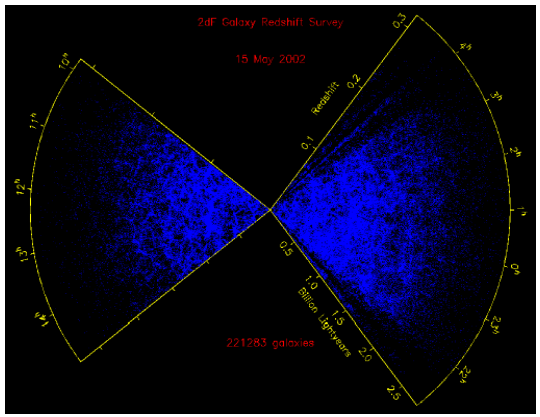
Cécile BARBIER (LAPP)

27 Septembre 2007

Le principe de CosmoMC

- **Le problème:**
  - on dispose de nombreuses observations de l'Univers à différentes époques:

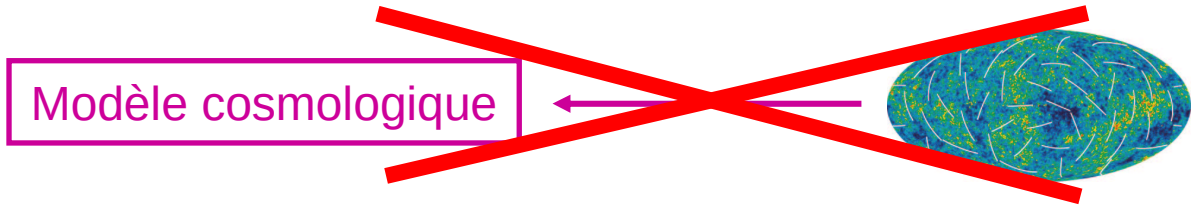




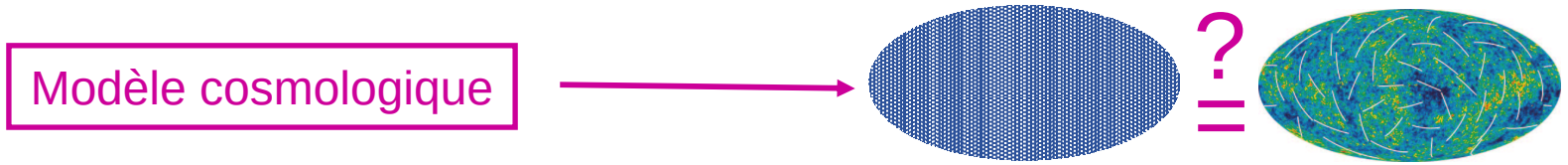
  - on veut en déduire ses caractérisques globales (composition, etc.) et son histoire globale

Le principe de CosmoMC

- **Le principe:**
  - on ne peut pas partir des observations et remonter le temps:



  - mais on peut:
    - faire l'hypothèse d'un modèle (="scénario cosmologique")
    - spécifier les paramètres du modèle (="paramètres cosmologiques") : entre 6 et 15 paramètres suivant les modèles)
    - faire tourner un code pour simuler l'évolution globale de cet Univers (prend environ 2-3 minutes)
    - voir si le résultat ressemble ou non aux vraies données
    - en déduire si le modèle de départ est bon ou mauvais

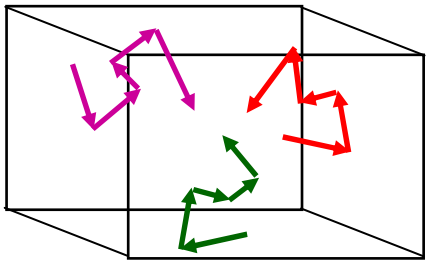


Le principe de CosmoMC

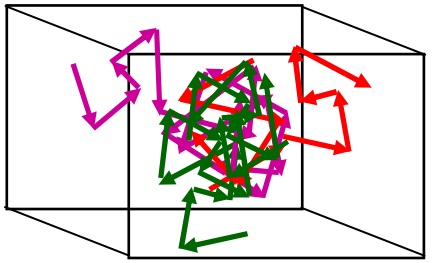
- **La méthode:**
  - le code:
    - explore l'espace des paramètres
    - pour chaque point dans cet espace, simule cet Univers et le compare aux données expérimentales
    - en déduit la distribution de probabilité de chaque paramètre
  - c'est un processus complexe en raison du grand nombre de paramètres (5 à 15)
  - tout dépend de l'algorithme d'exploration:
    - un algorithme naif aurait besoin de ~10^{15} itérations x 3 mn…(temps total > à l'âge de l'Univers!)
    - les meilleurs algorithmes ont besoin de 10^4 à 10^5 itérations (12h à une semaine suivant les modèles)

Le principe de CosmoMC

- **CosmoMC utilise un des meilleurs algorithmes connus (Metropolis-Hastings):**
  - Chaque processeur décrit une marche aléatoire dans l'espace des paramètres (= chaîne de Markhov), avec une "attirance" pour les régions en bon accord avec les données.



  - Les processeurs communiquent entre eux avec MPI. Ils mettent en commun les informations acquises par chacun sur la localisation des bonnes régions dans l'espace des paramètres, ce qui permet d'accélérer le processus.



  - Lorsque toutes les chaînes prédisent la même probabilité pour les paramètres cosmologiques, le processus a convergé et s'arrête. On en déduit les paramètres cosmologiques en accord avec les observations, avec valeur centrale et barre d'erreur.

Le principe de CosmoMC

- **Un run typique :**
  - 8 à 32 chaînes de Markhov (= 8 à 32 CPUs),
  - 10^3 à 10^5 itérations par chaîne (le temps de convergence dépend de beaucoup de choses: très difficile à prévoir !!! En général 0.5 à 10 jours)

- **pour être dans le coup et publier, il faut faire des nouveaux runs pour chaque combinaison de :**
  - nouveaux modèles (les théoriciens proposent souvent des nouveaux modèles intéressants à tester)
  - nouveaux ensembles de données (depuis ~2000 les astrophysiciens fournissent chaque mois de nouvelles données importantes)

  … pour cela il faut adapter et modifier le code public de base …

- **de plus, on peut faire des runs sur des données simulées pour aider les astrophysiciens à optimiser les caractéristiques des futures expériences**

Historique des runs

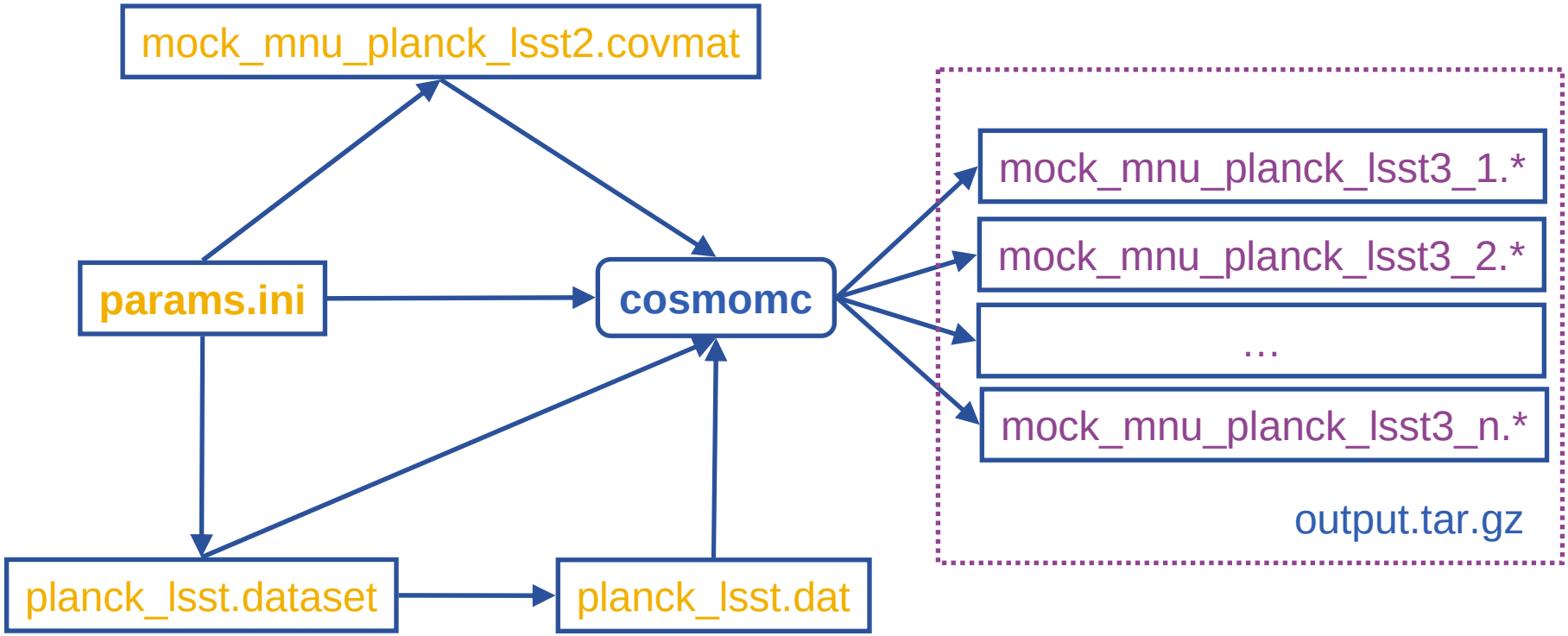
- **Runs en batch au CC-IN2P3 Lyon, à Barcelone (Mare Nostrum) et à Cambridge**
- **Compilation et tests au LAPP**
  - En interactif sur machine dédiée
- **Soumission depuis le LAPP**
  - sur ferme de calcul du LAPP en batch
  - sur ferme de calcul du LAPP via la grille avec la VO locale vo.lapp.in2p3.fr
  - sur la grille avec la VO esr

Création de l'exécutable

- **Code**
  - Langage : Fortran 90
  - Algèbre linéaire : librairie Lapack
  - Programmation parallèle : librairies OpenMP et MPI
- **Compilation**
  - Compilateur Intel sous licence :
    - UI identique aux WN
    - Exécutable semi-statique (option -i-static : pas besoin des librairies MPI Intel)

Fonctionnement du code

Exécution sur n CPUs : edg-job-submit –vo esr MPIcosmo.jdl



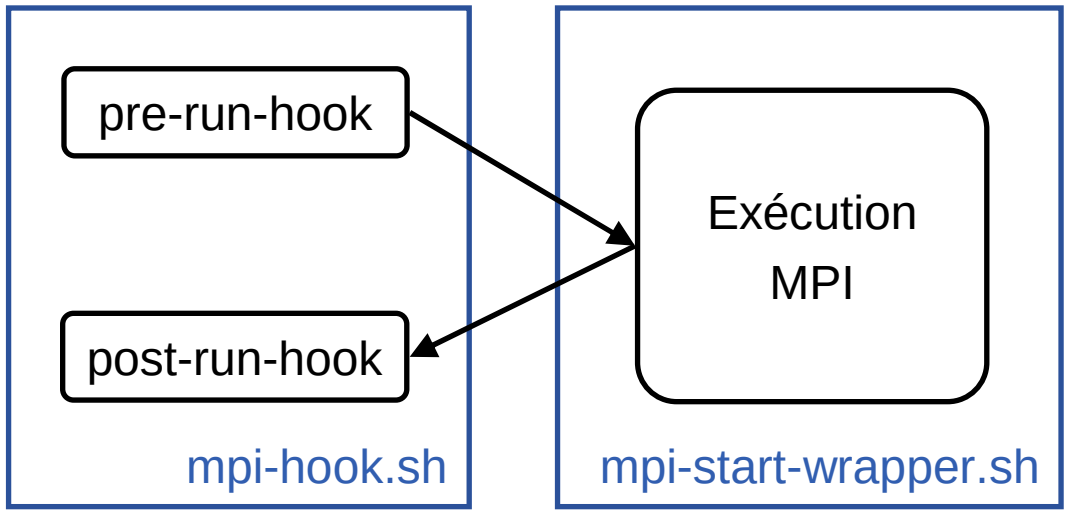
Fichier MPIcosmo.jdl

```
JobType        = "MPICH";
NodeNumber  = 8;
Executable     = "mpi-start-wrapper.sh";
Arguments     = "cosmomc params.ini OPENMPI";
StdOutput      = "std.out";
StdError       = "std.err";
InputSandbox = {          "mpi-start-wrapper.sh",
                          "mpi-hooks.sh",
                          "cosmomc",
                          "params.ini",
                          "../data_share/planck_lsst.dataset",
                          "../data_share/planck_lsst.dat",
                          "mock_mnu_planck_lsst2.covmat"};
OutputSandbox = {         "std.out",
                          "std.err",
                          "output.tar.gz"};
Requirements =
    Member("MPI-START", other.GlueHostApplicationSoftwareRunTimeEnvironment)
 &&  Member("OPENMPI", other.GlueHostApplicationSoftwareRunTimeEnvironment)
 ;
```

Soumission avec MPI

- **Documentation :**
  - http://egee-docs.web.cern.ch/egee-docs/uig/development/uc-mpi-jobs_2.html
  - http://www.grid.ie/mpi/wiki/FrontPage
  - http://www.grid.ie/mpi/wiki/JobSubmission
  - http://www.grid.ie/mpi/wiki/SiteConfig

- **Principe :**



Bilan

Depuis 3 mois :

- **Simulations quotidiennes**
- **En batch et par la grille**
- **4 types de problème :**
  - 2 nouveaux modèles
  - 2 nouveaux jeux de données
- **Tests de 8 à 32 chaînes**

→ **2 publications effectuées**
2 publications par mois d'ici fin décembre 2007

QUESTIONS ?

Wrapper MPI-START

```
# Pull in the arguments.
MY_EXECUTABLE=`pwd`/$1
MY_ARGS=$2
MPI_FLAVOR=$3

# Convert flavor to lowercase for passing
    to mpi-start.
MPI_FLAVOR_LOWER=`echo $MPI_FLAVOR
  | tr '[:upper:]' '[:lower:]'`

# Pull out the correct paths
    for the requested flavor.
eval MPI_PATH=
  `printenv MPI_${MPI_FLAVOR}_PATH`

# Ensure the prefix is correctly set.
  Don't rely on the defaults.
eval I2G_${MPI_FLAVOR}_PREFIX=$MPI_PATH
export I2G_${MPI_FLAVOR}_PREFIX
```

```
# Touch the executable.  It exist must for
    the shared file system check.
# If it does not, then mpi-start may try to distribute
    the executable when it shouldn't.
touch $MY_EXECUTABLE

# Setup for mpi-start.
export I2G_MPI_APPLICATION=$MY_EXECUTABLE
export I2G_MPI_APPLICATION_ARGS=$MY_ARGS
export I2G_MPI_TYPE=$MPI_FLAVOR_LOWER
export I2G_MPI_PRE_RUN_HOOK=mpi-hooks.sh
export I2G_MPI_POST_RUN_HOOK=mpi-hooks.sh

# If these are set then you will get more debugging
    information.
export I2G_MPI_START_VERBOSE=1
export I2G_MPI_START_DEBUG=1

# Invoke mpi-start.
$I2G_MPI_START
```

Pré-traitements

```
# This function will be called before the MPI
      executable is started.
# You can, for example, compile the
      executable itself.
#
pre_run_hook () {

  echo "Executing pre hook."

  # Check directory content
  echo "----------------------"
  echo "Check directory content"
  echo "----------------------"
  ls -lF

  # Make cosmomc executable
  echo "----------------------"
  echo "Make cosmomc executable"
  echo "----------------------"
  /bin/chmod +x cosmomc
  ls -lF cosmomc

# Change number of iterations
  echo "-------------------------"
  echo "Change number of iterations"
  echo "-------------------------"
  echo "Before :"
  grep "^samples =" params.ini
  sed -e "s|^samples = 200000|samples =
      16|" -i params.ini
  echo "After :"
  grep "^samples =" params.ini

  echo "Finished the pre hook."

  return 0
}
```

Post-traitements

```
#
# This function will be called before the MPI executable is finished.
# A typical case for this is to upload the results to a storage element.
#
post_run_hook () {

  echo "Executing post hook."

  # Check directory content
  echo "--------------------"
  echo "Contenu du repertoire"
  echo "--------------------"
  ls -lF

  # tar the results
  echo "-------------------------"
  echo "Recuperation des resultats"
  echo "-------------------------"
  /bin/tar cvfz output.tar.gz *.log *.txt

  echo "Finished the post hook."

  return 0
}
```