

Memo pratique d'utilisation des ressources de l'IDRIS

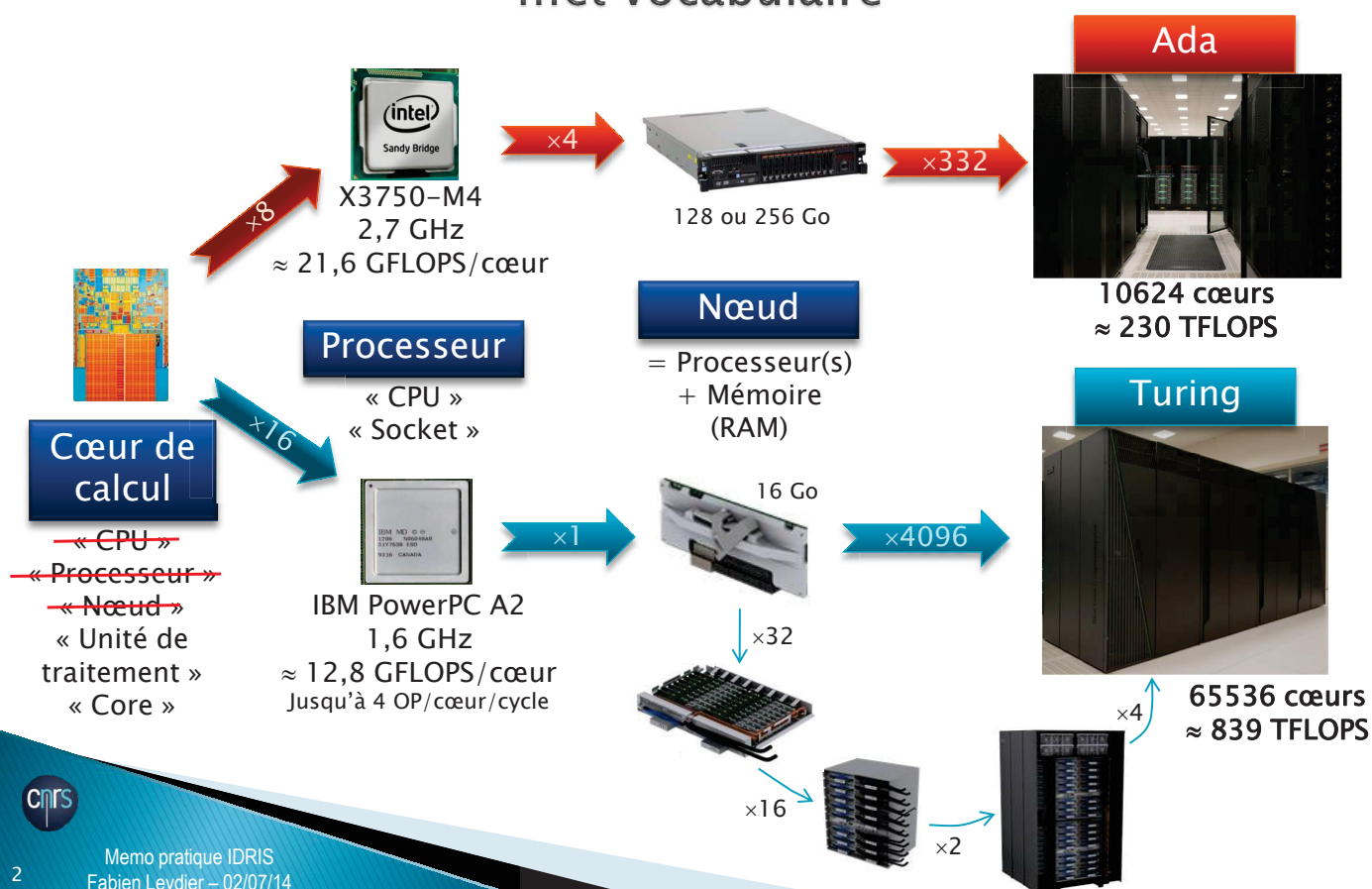
26 juin 2014



Fabien Leydier



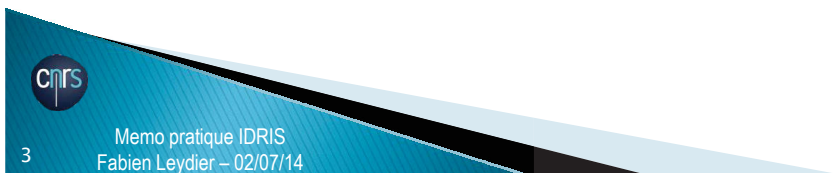
Architecture des machines de calcul... ...et vocabulaire



Comment calculer à l'IDRIS

- Les espaces disques
- Le système de classe
- Le script de soumission
 - Script mono- et multi-étapes
- Commandes de bilan/comptabilité

► Documentation plus détaillée disponible sur le site de l'IDRIS (www.idris.fr)



Les espaces disques

► 2 types de machines

- Serveur de fichiers GPFS
 - Espace disque des machines de calcul
 - Découpé en 3 sections pour chaque login du projet
 - \$HOME : petit espace, sauvegardé tous les jours
 - \$WORKDIR : espace de travail, pas de sauvegarde
 - \$TMPDIR : espace temporaire, effacé après chaque calcul
- Serveur d'archives (Ergon)
 - Stockage sur cartouches magnétiques
 - Archivage à long terme

► Demande d'espace

- Lors de la constitution du dossier DARI
- <https://extranet.idris.fr/>
 - Demande par machine
 - Volume + inodes pour chaque machine de calcul et pour Ergon



Les espaces disques

► Serveur de fichier GPFS

- \$HOME
 - Modèles de scripts, exécutables, etc.
- \$WORKDIR : fichiers d'entrée et de sortie des calculs
 - On peut y lancer des calculs
 - Visibilité directe pendant l'exécution
 - 1 dossier pour 1 calcul permet de mieux s'y retrouver
 - Soumis à quota
 - Faire le ménage en fin de calcul
 - Performances identiques au \$TMPDIR
- \$TMPDIR : espace temporaire par calcul
 - Utile si les fichiers générés sont très (très) volumineux
 - Supprimé en fin de calcul
 - Pas de ménage à faire en fin de calcul
 - Attention à prévoir la recopie sur \$WORKDIR !
 - Conservé entre les étapes d'un *job* multi-étapes
 - Utile pour être sûr de recopier ses fichiers en fin de calcul
 - Limite de temps, plantage, etc.
 - Pas de visibilité directe si $N_{\text{cœurs}} \leq 32$ sans *job* multi-étapes
 - Sinon, chemin à récupérer pendant l'exécution (**echo \$TMPDIR**)



Les espaces disques

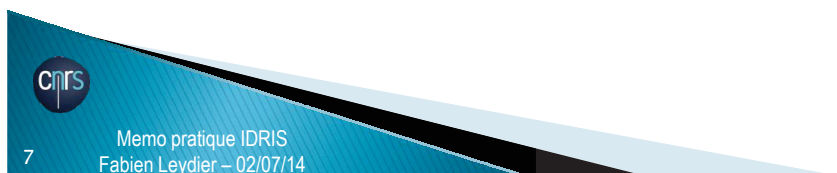
► Serveur d'archives (Ergon)

- Stockage sur cartouches magnétiques
 - Conserver des résultats à long terme
 - Réutilisés localement pour des calculs ultérieurs
 - Fichiers trop volumineux pour le \$WORKDIR
 - Sauvegarde sélective de résultats du \$WORKDIR
- Accès aux fichiers
 - Commandes spécifiques :
 - **mput** : écrire sur Ergon
 - **mget** : copie de Ergon vers le répertoire courant
 - Durée de vie des fichiers
 - 1 an par défaut
 - Allongement via la commande **mret** une fois par an
 - Email envoyé avant expiration des fichiers



Les espaces disques

- ▶ **\$WORKDIR** partagé entre Ada et Turing
- ▶ **Commandes pratiques**
 - Quotas
 - Sur chaque machine :
 - `quota_u` : quota pour le \$HOME
 - `quota_u -w` : quota pour le \$WORKDIR
(rappel : pas de quota sur \$TMPDIR)
 - Rappel : espace utilisé par un répertoire
 - `du -sh nom_du_répertoire`



Le système de classes

- ▶ **Répartition des *jobs* par LoadLeveler**
 - Type de *job* (séquentiel, OpenMP, MPI/hybride)
 - Nombre de cœurs ou nœuds de calcul
 - Mémoire
 - Temps d'exécution
- ▶ **Sur Ada**
 - Temps maximum par *job* : $100\text{ h} \leq 32\text{ cœurs}$, $20\text{ h} \leq 2048\text{ cœurs}$
 - *Jobs* OpenMP et grosse mémoire sur des nœuds dédiés (28 nœuds)
- ▶ **Sur Turing**
 - Temps maximum par *job* : 20 h (jusqu'à toute la machine, 4096 nœuds)
 - Réservation minimale aujourd'hui : 64 nœuds (= 1024 cœurs)
- ▶ **Commande : `news class`**



Le système de classes

► Commandes

- `llsubmit script` : lancer son script de soumission
- `llcancel numéro_du_job` : annuler l'exécution
- `llq` : pour voir toute la file d'attente
- `llq -u login` : pour voir sa propre file d'attente
 - Pour personnaliser l'affichage : `man llq (llq -f %...)`
- `llq -l numéro_du_job` : TOUTES les informations connues sur le *job* (temps, dossiers, etc.)

Le système de classes

► exemple

Nœud maître
d'exécution

```
$ llq
Id                Owner      Submitted   ST PRI Class      Running On
-----
ada338.222911.2   rgmt002    4/22 15:06 R  100 mt8t4      ada305-id
ada338.228902.0   rspe001    4/26 11:08 R  100 mt8t4      ada316-id
ada338.228957.0   rspe001    4/26 11:48 R  100 mt8t4      ada320-id
ada338.229237.0   rspe001    4/26 14:55 R  100 mt8t4      ada306-id
ada338.229277.0   rspe001    4/26 15:19 R  100 mt8t4      ada308-id
ada338.229375.0   rspe001    4/26 16:14 R  100 mt8t4      ada312-id
ada338.229376.0   rspe001    4/26 16:15 R  100 mt8t4      ada307-id
ada338.228083.0   rwvq004    4/25 18:17 R  100 t4L        ada309-id
ada338.228084.0   rwvq004    4/25 18:17 R  100 t4L        ada311-id
ada338.228085.0   rwvq004    4/25 18:17 R  100 t4L        ada314-id
ada338.228086.0   rwvq004    4/25 18:17 R  100 t4L        ada315-id
ada338.228087.0   rwvq004    4/25 18:17 R  100 t4L        ada310-id
ada338.228088.0   rwvq004    4/25 18:17 R  100 t4L        ada317-id
ada338.228089.0   rwvq004    4/25 18:17 R  100 t4L        ada318-id
ada338.228090.0   rwvq004    4/25 18:17 R  100 t4L        ada305-id
ada338.228091.0   rwvq004    4/25 18:17 R  100 t4L        ada313-id
ada338.228092.0   rwvq004    4/25 18:17 R  100 t4L        ada316-id
ada338.232472.1   regi904    4/29 16:08 R  100 archive    ada338
ada338.231397.0   rjjv002    4/28 19:32 R  100 c8t4L       ada320-id
ada338.231310.0   rana323    4/28 19:32 R  100 c16t4       ada306-id
ada338.231309.0   rana323    4/28 19:32 R  100 mt8t4       ada308-id
ada338.231308.0   rana323    4/28 19:32 R  100 c128t3      ada002-id
ada338.231307.0   rana323    4/28 19:32 R  100 c256t3      ada013-id
ada338.231306.0   rana323    4/28 19:32 R  100 c1024t3     ada039-id
ada338.231305.0   rana323    4/28 19:32 R  100 c512t3      ada144-id
ada338.231304.0   rana323    4/28 19:32 R  100 mt16t4      ada312-id
ada338.231303.0   rana323    4/28 19:32 R  100 c32t3       ada191-id
```

Classe OpenMP : **mt**
Classe MPI/hybride : **c**
Nombre de cœurs max. de la classe : **t0,1,2,3,4**
Temps max. de la classe : **t0,1,2,3,4**
Nœuds mémoire large : **L**

Le script de soumission

```

#@ job_name =
#@ job_type =
#@ output =
#@ error =
#@ total_tasks =
#@ parallel_threads =
#@ wall_clock_limit =
#@ queue

```

module load chimie

set -x

poe chimie.exe input.in > out

Directives LoadLeveler

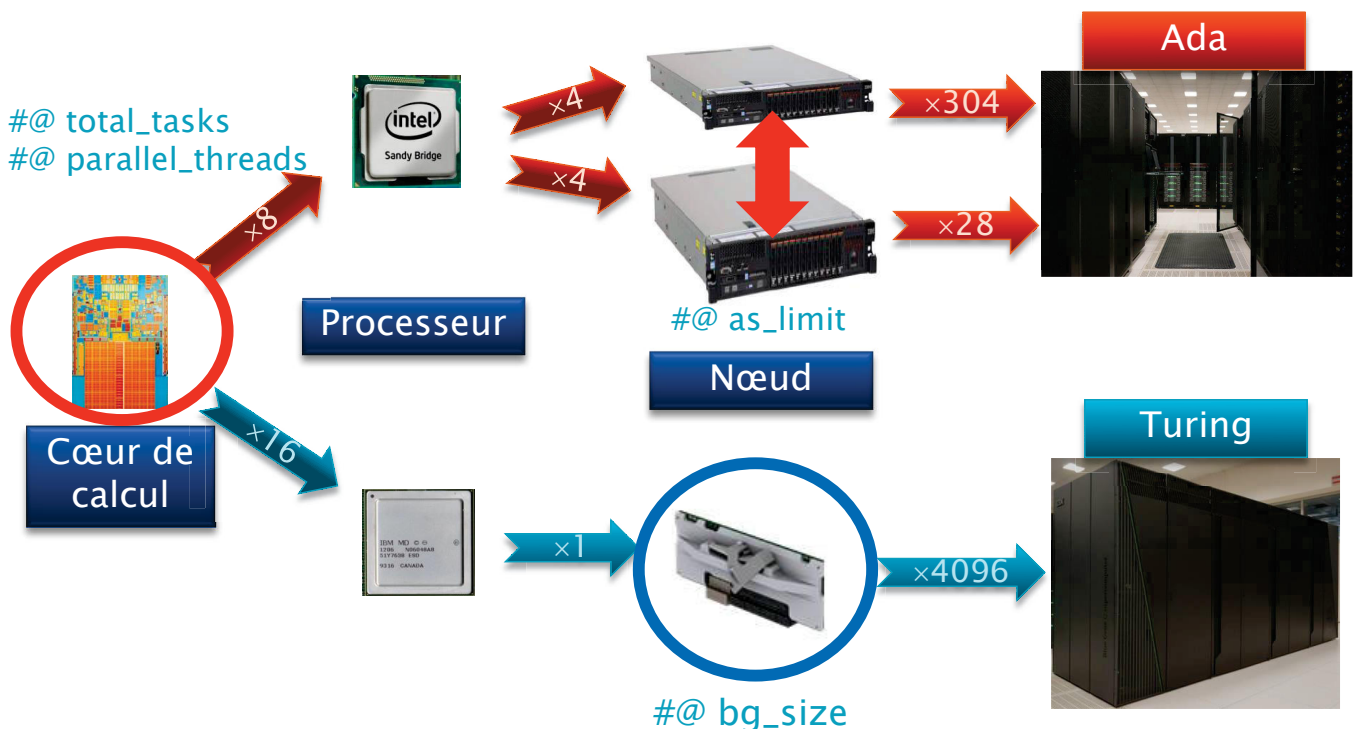
- Commencent par `#@`
 - ~~`#@, # @, # #@, # # @, @, #`~~
- Se terminent par « `#@ queue` »
- Obligatoires / Fortement conseillées :
 - `#@ job_name`
 - `#@ job_type`
 - `#@ total_tasks` et/ou `parallel_threads`
 - `#@ bg_size` (pour Turing)
 - `#@ wall_clock_limit`
 - `#@ output`
 - `#@ error`
- Optionnelles :
 - `#@ notify_user` : envoi de mails
 - `#@ notification` : type de mail
 - `#@ class` : pour les classes spéciales
 - `#@ as_limit` : demande mémoire par processus
 - **ATTENTION !** Ada ≠ Vargas : `data_limit` et `stack_limit` ne vous donneront pas votre demande, et risquent de la limiter
 - Documentation : <http://www.idris.fr/ada/ada-doc-ibm-intel.html>



Memo pratique IDRIS
Fabien Leydier – 02/07/14

11

Ce que je réserve pour mon calcul



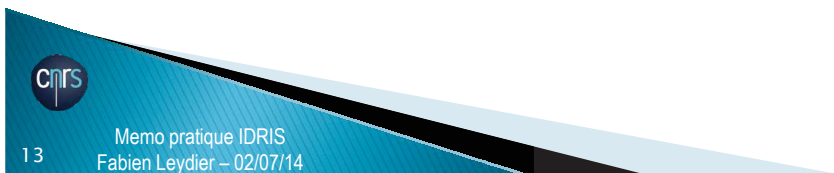
Memo pratique IDRIS
Fabien Leydier – 02/07/14

12

Le script de soumission

► Classes spéciales

- Classe archive
 - Transferts de fichiers Ergon (mfput, mfget), « manipulation » de fichiers (cp, cat, >, ...)
 - `#@ class = archive`
 - Non facturée, calculs interdits
- Classe compilation
 - Exécution séquentielle, 20 h max
 - `#@ class = compil`
- Classe pré/post traitement
 - Traitement des données
 - `# @ requirements = (Feature == "prepost")`
 - Non facturée, calculs interdits, durée très limitée



Le script de soumission : OpenMP

► Ada

- `#@ job_type = serial`
- Réserve des cœurs
 - `#@ parallel_threads = Ncœurs`
 - Décompte du temps (« facturation »)
 - $\leq 32 \text{ cœurs} : t \times N_{\text{cœurs}}$
- Réserve de la mémoire
 - Par défaut (pas de directive)
 - $3,5 \text{ Go} \times N_{\text{cœurs}}$ pour le programme
 - `#@ as_limit = Mem`
 - Au maximum : $7,0 \times N_{\text{cœurs}}$ pour le programme (ex : `# @ as_limit = 28.0GB` pour 4 threads)
 - Utilisation des nœuds « large mémoire »



Le script de soumission : OpenMP

► Turing

- Pas impossible, mais...
 - C'est utilisable a priori
 - D'après le découpage, 1 nœud = 16 cœurs + 16 Go de mémoire
 - Le programme s'exécute en intra-nœud : jusqu'à 64 *threads*
 - Réserve minimale
 - `#@ bg_size ≥ 64` ... nœuds
 - Le programme ne va s'exécuter que sur 1 nœud : 63 nœuds « perdus » !
 - Performances
 - Processeurs
 - Turing : 1 nœud \approx 204,1 GFLOPS
 - Ada : 1 nœud \approx 691,2 GFLOPS
 - Mémoire
 - Turing : 16 Go/nœud \rightarrow 256 Mo—1 Go/cœur
 - Ada : 128—256 Go/nœud \rightarrow 4—8 Go/cœur
- En OpenMP, Turing n'est pas adapté par rapport à Ada

Le script de soumission : MPI

► Ada

- `#@ job_type = parallel`
- Réserve des cœurs
 - `#@ total_tasks = Ncœurs`
 - Décompte du temps (« facturation »)
 - ≤ 32 cœurs : $t \times N_{\text{cœurs}}$
 - > 32 cœurs : $t \times N_{\text{nœuds}} \times 32$ (ex : 33 cœurs demandés = 64 cœurs décomptés !)
- Réserve de la mémoire
 - Par défaut (pas de directive)
 - 3,5 Go/cœur = 3,5 Go/processus MPI
 - `#@ as_limit = Mem`
 - Utilisation des nœuds « larges mémoire » : max 32 cœurs
 - 7,0 Go/cœur = 7,0 Go/processus MPI


Le script de soumission : MPI

► Turing

- `#@ job_type = BLUEGENE`
- Réserve des nœuds
 - `#@ bg_size = Nnœuds`
 - $N_{\text{nœuds}} \geq 64$ (\Rightarrow 1024 cœurs = de 64 à 4096 processus MPI !)
 - Réserve par puissance de 2 (64, 128, 256, ..., 4096)
 - Décompte du temps (« facturation »)
 - $t \times N_{\text{nœuds}} \times 16$
- Répartition MPI
 - `--np NMPI` : nombre total de processus MPI (jusqu'à 4 par cœur)
- Réserve de la mémoire
 - Selon l'agencement des processus (`--ranks-per-node=`)
 - Minimum : 64 processus/nœud \rightarrow 256 Mo/processus MPI
 - Maximum : 1 processus/nœud \rightarrow 16 Go/processus MPI

Le script de soumission : Hybride

► Ada

- `#@ job_type = parallel`  *Elément de base : processus MPI*
- Réserve des cœurs
 - `#@ total_tasks = Nprocessus MPI`
 - `#@ parallel_thread = Nthreads/processus`
 - $N_{\text{cœurs}} = N_{\text{processus MPI}} \times N_{\text{threads/processus}}$
 - Décompte du temps (« facturation »)
 - ≤ 32 cœurs : $t \times N_{\text{cœurs}}$
 - > 32 cœurs : $t \times N_{\text{nœuds}} \times 32$ (ex : 33 cœurs demandés = 64 cœurs décomptés !)
- Réserve de la mémoire
 - Par défaut (pas de directive)
 - 3,5 Go/cœur = $3,5 \text{ Go} \times N_{\text{threads/processus}} / \text{processus MPI}$
 - `#@ as_limit = Mem`
 - Au maximum : $7,0 \times N_{\text{cœurs}} / \text{processus MPI}$ (ex : `#@ as_limit = 28,0GB` pour 4 threads/processus MPI)
 - Limité à 32 cœurs ($N_{\text{cœurs}}$ au total)

Le script de soumission : Hybride

► Turing

- `#@ job_type = BLUEGENE`
- Réserve des nœuds
 - `#@ bg_size = Nnœuds`
 - $N_{\text{nœuds}} \geq 64$ (\Rightarrow 1024 cœurs = de 64 à 4096 processus MPI !)
 - Réserve par puissance de 2 (64, 128, 256, ..., 4096)
 - Décompte du temps (« facturation »)
 - $t \times N_{\text{nœuds}} \times 16$
- Répartition MPI
 - `--np NMPI` : nombre total de processus MPI
- Réserve de la mémoire
 - Selon l'agencement des processus (`--ranks-per-node=`)
 - Minimum : 64 processus/nœud \rightarrow 256 Mo/processus MPI
 - Maximum : 1 processus/nœud \rightarrow 16 Go/processus MPI
- Répartition OpenMP
 - `--envs "OMP_NUM_THREADS=NOMP"` : nombre de *threads* par processus MPI
- Bilan de répartition
 - $Bg_size \times \text{ranks-per-node} = np$
 - $\text{Ranks-per-node} \times \text{OMP_NUM_THREADS} \leq 64$
 - Pas de mémoire supplémentaire apportée par des *threads* OpenMP, contrairement à Ada



Memo pratique IDRIS
Fabien Leydier – 02/07/14

19

Le script de soumission

► Lignes de commandes

```
#@ job_name =  
#@ job_type =  
#@ output =  
#@ error =  
#@ total_tasks =  
#@ parallel_threads =  
#@ wall_clock_limit =  
#@ queue
```

```
module load chimie  
  
set -x  
  
poe chimie.exe input.in > out
```

- Programme Modules
 - Charge l'environnement du logiciel (*paths*, correctifs, variables d'environnement)
 - Commandes :
 - `module load logiciel[/version]`
 - `module avail [logiciel]`
 - `module show logiciel[/version]`
 - `module unload logiciel[/version]`
 - `module switch logiciel logiciel/version`
- `set -x`
 - Donne le retour des commandes passées
 - Très utile en cas de problème
 - Peut être verbeux (module, etc.)
 - A placer après `module` par exemple



Memo pratique IDRIS
Fabien Leydier – 02/07/14

20

Le script de soumission

► Lignes de commandes

```
## job_name =  
## job_type =  
## output =  
## error =  
## total_tasks =  
## parallel_threads =  
## wall_clock_limit =  
## queue
```

```
module load chimie
```

```
set -x
```

```
poe chimie.exe input.in > out
```

◦ Lancement de l'exécutable

- OpenMP : lancement direct
 - Ex : **g09 input**
- MPI et hybride : lancement avec **poe**
 - Ex : **poe MPPCrystal**
- Pour certains logiciels : « **< input** »
 - Ex : **poe pw.x < input**
- Sur Turing : lancement avec **runjob**

◦ Redirection de la sortie standard (écran)

- Par défaut : tout est copié dans le fichier indiqué par **## output**
- Indiquer « **> fichier** » en fin de ligne d'exécutable pour découpler sortie du programme et sortie du **job**
 - Conseil : **## output** et **## error** vers un même fichier, et « **> fichier** » pour une sortie du calcul indépendante



Le script de soumission

► Exemple Gaussian sur Ada avec \$TMPDIR

```
## job_name = moncalculGaussian  
## job_type = serial  
## output = $(job_name).$(jobid)  
## error = $(job_name).$(jobid)  
## parallel_threads = 32  
## wall_clock_limit = 100:00:00  
## queue  
  
module load gaussian/g09_C01  
  
set -x  
  
cd $TMPDIR  
cp $LOADL_STEP_INITDIR/input.in .  
cp $LOADL_STEP_INITDIR/input.chk .  
  
g09 input.in > output  
  
rm *.chk  
  
mkdir $LOADL_STEP_INITDIR/resultat  
cp * $LOADL_STEP_INITDIR/resultat
```

► Exemple VASP sur Ada avec \$WORKDIR

```
## job_name = moncalculVASP  
## job_type = parallel  
## output = $(job_name).$(jobid)  
## error = $(job_name).$(jobid)  
## total_tasks = 128  
## wall_clock_limit = 20:00:00  
## queue  
  
module load vasp  
  
set -x  
  
poe vasp
```



Le script de soumission

► Exemple CPMD sur Turing avec \$WORKDIR

```
#!/bin/bash
#@ job_name = moncalculCPMD
#@ job_type = BLUEGENE
#@ output = $(job_name).$(jobid)
#@ error = $(job_name).$(jobid)
#@ bg_size = 512
#@ wall_clock_limit = 20:00:00
#@ queue

module load cpmd

set -x

runjob --ranks-per-node 4 --envs "OMP_NUM_THREADS=16" --np 2048 : $CPMD_EXEDIR/cpmd.x ./input > out
```



23

Memo pratique IDRIS
Fabien Leydier – 02/07/14

Script multi-étapes

```
#!/bin/bash
#==== Directives globales ====
#@ job_name = multi-steps-calcul
#@ output = $(job_name).$(step_name).$(jobid)
#@ error = $(output)

#==== Directives pour étape 1 ====
#@ step_name = calcul
#@ job_type = serial
#@ parallel_threads = 8
#@ wall_clock_limit = 20:00:00
#@ queue

#==== Directives pour étape 2 ====
#@ step_name = copie
#@ dependency = (calcul >= 0)
#@ class = archive
#@ queue

case ${LOADL_STEP_NAME} in

#== Etape 1 =====
calcul )

set -ex
module load gaussian
cd $TMPDIR
cp ${LOADL_STEP_INITDIR}/* .
g09 input > resultat
;;

#== Etape 2 =====
copie )

set -ex
cd $TMPDIR
mkdir ${LOADL_STEP_INITDIR}/Output
cp * ${LOADL_STEP_INITDIR}/Output
;;

esac
```

- Permet de lancer l'équivalent de plusieurs scripts à la suite
 - Utilisation du TMPDIR (rémanent)
 - Sauver les résultats obtenus après la limite de temps de l'étape calcul
 - Dépendance entre les étapes
- S'écrit comme des scripts « éclatés »
- Lignes en # : commentaires



24

Memo pratique IDRIS
Fabien Leydier – 02/07/14

Script multi-étapes

```
#===== Directives globales =====
# @ job_name = multi-steps-calcul
# @ output = $(job_name).$(step_name).$(jobid)
# @ error = $(output)

#===== Directives pour étape 1 =====
# @ step_name = calcul
# @ job_type = serial
# @ parallel_threads = 8
# @ wall_clock_limit = 20:00:00
# @ queue

#===== Directives pour étape 2 =====
# @ step_name = copie
# @ dependency = (calcul >= 0)
# @ class = archive
# @ queue

case ${LOADL_STEP_NAME} in

#== Etape 1 =====
calcul )

set -ex
module load gaussian
cd $TMPDIR
cp ${LOADL_STEP_INITDIR}/*.
g09 input > resultat
;;

#== Etape 2 =====
copie )

set -ex
cd $TMPDIR
mkdir ${LOADL_STEP_INITDIR}/Output
cp * ${LOADL_STEP_INITDIR}/Output
;;

esac
```

► Découpé en deux parties

- Directives LoadLeveler
- Commandes
- Encadrées par des commandes obligatoires :
case et esac



Script multi-étapes

```
#===== Directives globales =====
# @ job_name = multi-steps-calcul
# @ output = $(job_name).$(step_name).$(jobid)
# @ error = $(output)

#===== Directives pour étape 1 =====
# @ step_name = calcul
# @ job_type = serial
# @ parallel_threads = 8
# @ wall_clock_limit = 20:00:00
# @ queue

#===== Directives pour étape 2 =====
# @ step_name = copie
# @ dependency = (calcul >= 0)
# @ class = archive
# @ queue

case ${LOADL_STEP_NAME} in

#== Etape 1 =====
calcul )

set -ex
module load gaussian
cd $TMPDIR
cp ${LOADL_STEP_INITDIR}/*.
g09 input > resultat
;;

#== Etape 2 =====
copie )

set -ex
cd $TMPDIR
mkdir ${LOADL_STEP_INITDIR}/Output
cp * ${LOADL_STEP_INITDIR}/Output
;;

esac
```

► 1^{er} paragraphe

- Directives communes à toutes les étapes
 - Indiquer le minimum de directives car certaines ne se propagent pas forcément sur toutes les étapes
- Chaque étape possède une sortie distincte



Script multi-étapes

```
#===== Directives globales =====
#@ job_name = multi-steps-calcul
#@ output = $(job_name).$(step_name).$(jobid)
#@ error = $(output)

#===== Directives pour étape 1 =====
#@ step_name = calcul
#@ job_type = serial
#@ parallel_threads = 8
#@ wall_clock_limit = 20:00:00
#@ queue

#===== Directives pour étape 2 =====
#@ step_name = copie
#@ dependency = (calcul >= 0)
#@ class = archive
#@ queue

case ${LOADL_STEP_NAME} in

#== Etape 1 =====
calcul )

set -ex
module load gaussian
cd $TMPDIR
cp ${LOADL_STEP_INITDIR}/* .
g09 input > resultat
;;

#== Etape 2 =====
copie )

set -ex
cd $TMPDIR
mkdir ${LOADL_STEP_INITDIR}/Output
cp * ${LOADL_STEP_INITDIR}/Output
;;

esac
```

► 1^{re} étape :

Lancement du calcul

- **#@ step_name**
 - Doit correspondre dans les directives et les commandes
 - Eviter les caractères spéciaux (-#@ : non acceptés)
- **set -ex**
 - Donne l'écho des commandes
 - En cas d'erreur sur l'une des commandes, l'ensemble de l'étape est considérée comme erronée
 - A placer en tête de l'étape
- Les directives se terminent par « **#@ queue** »
- Les commandes se terminent par « **;;** »



Script multi-étapes

```
#===== Directives globales =====
#@ job_name = multi-steps-calcul
#@ output = $(job_name).$(step_name).$(jobid)
#@ error = $(output)

#===== Directives pour étape 1 =====
#@ step_name = calcul
#@ job_type = serial
#@ parallel_threads = 8
#@ wall_clock_limit = 20:00:00
#@ queue

#===== Directives pour étape 2 =====
#@ step_name = copie
#@ dependency = (calcul >= 0)
#@ class = archive
#@ queue

case ${LOADL_STEP_NAME} in

#== Etape 1 =====
calcul )

set -ex
module load gaussian
cd $TMPDIR
cp ${LOADL_STEP_INITDIR}/* .
g09 input > resultat
;;

#== Etape 2 =====
copie )

set -ex
cd $TMPDIR
mkdir ${LOADL_STEP_INITDIR}/Output
cp * ${LOADL_STEP_INITDIR}/Output
;;

esac
```

► 2^e étape :

Copie des résultats

- **#@ dependency**
 - Gère la condition de lancement de l'étape 2
 - Utilise le code de retour dans la condition
 - == 0 : si l'étape 1 s'est bien déroulée
 - > 0 : si l'étape 1 a eu un problème
 - >= 0 : peu importe l'état de l'étape 1
- L'ordre d'exécution est à gérer soi-même



Script multi-étapes : exemple TMPDIR

```
# @ job_name = multi-steps-vasp
# @ output = $(job_name).$(step_name).$(jobid)
# @ error = $(output)
```

```
# @ step_name = copy
# @ class = archive
# @ queue
```

```
# @ step_name = vasp
# @ dependency = (copy == 0)
# @ job_type = parallel
# @ total_tasks = 128
# @ wall_clock_limit = 10:00:00
# @ queue
```

```
# @ step_name = fin
# @ dependency = (copy == 0) && (vasp >= 0)
# @ class = archive
# @ queue
```

dépendances



```
case ${LOADL_STEP_NAME} in
```

```
copy )
set -ex
cd $TMPDIR
cp ${LOADL_STEP_INITDIR}/INCAR .
cp ${LOADL_STEP_INITDIR}/POSCAR .
cp ${LOADL_STEP_INITDIR}/POTCAR .
cp ${LOADL_STEP_INITDIR}/KPOINTS .
cp ${LOADL_STEP_INITDIR}/WAVECAR .
;;
```

```
vasp )
set -ex
module load vasp
cd $TMPDIR
echo "calcul lancé" > etat
poe vasp
;;
```

```
fin )
set -ex
cd $TMPDIR
rm WAVECAR
rm CHG*
cp * ${LOADL_STEP_INITDIR}
cd ${LOADL_STEP_INITDIR}
echo "job terminé" > etat
;;

esac
```

Script multi-étapes : exemple WORKDIR

```
# @ job_name = multi-steps-vasp
# @ output = $(job_name).$(step_name).$(jobid)
# @ error = $(output)
```

```
# @ step_name = copy
# @ class = archive
# @ queue
```

```
# @ step_name = vasp
# @ dependency = (copy == 0)
# @ job_type = parallel
# @ total_tasks = 128
# @ wall_clock_limit = 10:00:00
# @ queue
```

```
# @ step_name = fin
# @ dependency = (copy == 0) && (vasp >= 0)
# @ class = archive
# @ queue
```

dépendances



```
case ${LOADL_STEP_NAME} in
```

```
copy )
set -ex
mfget WAVECAR.calcul_01-06-13 WAVECAR
;;
```

```
vasp )
set -ex
module load vasp
poe vasp
;;
```

```
fin )
set -ex
mfput WAVECAR WAVECAR.calcul-28-06-13
;;

esac
```

Script multi-étapes : remarques

- ▶ Ne pas lancer plus d'une étape de calcul par *job*
 - Monopolisation des ressources en court-circuitant la file d'attente
 - Utiliser plus de cœurs pour diminuer le temps de restitution
 - Utiliser des *jobs* en cascade :

Script1

```
# @ job_name = multi-steps-vasp1
# @ output = $(job_name).$(step_name).$(jobid)
# @ error = $(output)

# @ step_name = vasp
# @ job_type = parallel
# @ total_tasks = 128
# @ wall_clock_limit = 20:00:00
# @ queue

# @ step_name = copy
# @ dependency = (vasp >= 0)
# @ job_type = archive
# @ queue

# @ step_name = submit
# @ dependency = (vasp > 0)&&(copy == 0)
# @ job_type = serial
# @ queue

case ${LOADL_STEP_NAME} in
    vasp )
        set -ex
        module load vasp
        cd $TMPDIR
        cp ${LOADL_STEP_INITDIR}/INCAR .
        cp ${LOADL_STEP_INITDIR}/POSCAR .
        cp ${LOADL_STEP_INITDIR}/POTCAR .
        cp ${LOADL_STEP_INITDIR}/KPOINTS .
        poe vasp
        ;;
    copy )
        set -ex
        cd $TMPDIR
        cp * ${LOADL_STEP_INITDIR}
        ;;
    submit )
        set -ex
        !submit Script2
        esac
```

Script2

```
# @ job_name = multi-steps-vasp2
# @ output = $(job_name).$(step_name).$(jobid)
# @ error = $(output)

# @ step_name = vasp
# @ job_type = parallel
# @ total_tasks = 128
# @ wall_clock_limit = 20:00:00
# @ queue

# @ step_name = copy
# @ dependency = (vasp >= 0)
# @ job_type = archive
# @ queue

case ${LOADL_STEP_NAME} in
    vasp )
        set -ex
        module load vasp
        cd $TMPDIR
        cp ${LOADL_STEP_INITDIR}/INCAR .
        cp ${LOADL_STEP_INITDIR}/POSCAR .
        cp ${LOADL_STEP_INITDIR}/POTCAR .
        cp ${LOADL_STEP_INITDIR}/KPOINTS .
        cp ${LOADL_STEP_INITDIR}/CONTCAR .
        cp ${LOADL_STEP_INITDIR}/WAVECAR .
        cp ${LOADL_STEP_INITDIR}/CHG* .
        poe vasp
        ;;
    copy )
        set -ex
        cd ${LOADL_STEP_INITDIR}
        mv OUTCAR OUTCAR.1
        mv OSZICAR OSZICAR.1
        mv XDATCAR XDATCAR.1
        cd $TMPDIR
        cp * ${LOADL_STEP_INITDIR}
        ;;
    esac
```

Commandes de bilan/comptabilité

- ▶ **Commande idrjar**
 - Edite un bilan de ses *jobs* par période
 - Par défaut, édite le bilan sur le mois en cours
 - **idrjar -d numéro_de_mois** : choix du mois
 - **idrjar -e date-date** : choix d'une période
 - Données du bilan :
 - noms et numéros des *jobs*
 - nombre de cœurs
 - temps *elapsed* et CPU (s)
 - efficacité : rapport (temps CPU) / (temps elapsed × N_{cœurs})
 - Donne une information sur l'efficacité parallèle de chaque *job*
- ▶ **Récapitulatif des heures consommées**
 - Commande **cpt**
 - par login, pour tout le groupe
 - % de l'attribution restant
 - Extranet de l'IDRIS
 - <https://extranet.idris.fr/>