GUIDE DE CALCUL SCIENTIFIQUE AU CENTRE D'INFORMATIQUE NATIONAL DE L'ENSEIGNEMENT SUPERIEUR (CINES)

Laure Coquart, en collaboration avec l'équipe assistance scientifique et l'équipe système

Octobre 2005

TABLE DES MATIERES

1. Généralités

1.1 Moyens informatiques de calcul

En termes de moyens de calcul, le CINES dispose de deux serveurs de **calcul parallèle**, l'un IBM P1600, l'autre SGI O3800.
Les caractéristiques de la machine IBM sont :

- 9 nœuds de 32 processeurs, avec 32Go ou 64Go de mémoire par nœud
- un espace total de travail de 4 To
- 5.2 Gflops crête par processeur (1.85 Tflops)
- la connexion se fait sur zeus.cines.fr qui est un des nœuds, mais la machine de calcul globale s'appelle hera

Les caractéristiques de la machine SGI sont :

- 512 processeurs dotés de 256 Go de mémoire centrale sur minerve
- 256 processeurs dotés de 128 Go de mémoire centrale sur athena
- un espace de travail local de 1.5 To
- 1 Gflop crête par processeur (768 Gflops au total)
- la connexion se fait sur leda.cines.fr et les calculs sur athena et minerve

Il y a également un serveur de fichiers qui est constitué de deux machines SGI O2100. Les données gérées par le serveur sont communes au /home des deux machines IBM et SGI par montage NFS. Les caractéristiques du serveur sont les suivantes :

- 8 processeurs par machines
- 22 To d'espace disque au total

Il existe également un serveur de visualisation qui est une machine SGI O2000 ONYX2 caractérisée par :

- 8 processeurs
- une mémoire globale de 4Go
- 4 pipes graphiques

1.2 Demande de login

Lorsqu'on ne possède pas de compte au CINES, il faut remplir un dossier (entièrement électronique) de demande de ressources sur les serveurs de calcul, qui devra comporter aussi bien des informations scientifiques que techniques sur le projet. On le trouve sur http://dari.cines.fr. Les dossiers de demandes d'heures sont évalués chaque automne par des groupes d'experts classés en Comités Thématiques (CT) donnés ci-dessous.

Les demandes d'heures en cours d'année doivent être adressées avec une lettre au directeur du CINES en précisant les raisons de cette demande hors campagne d'évaluation, et le dossier sera transmis au président du comité thématique concerné. Ce dossier est à renouveler chaque année à l'automne pour obtenir des heures de calcul. Une convention de prestation de services doit être également signée entre l'organisme auquel appartient le nouvel utilisateur et le CINES et se trouve à l'adresse internet http://www.cines.fr/textes/Conv4.doc (ou encore dans Utilisateurs puis Convention directement sur le site).

Lorsqu'on possède déjà un numéro de compte au CINES, il faut remplir un formulaire de demande d'ouverture de login. On peut trouver et imprimer ce formulaire sur internet à l'adresse http://www.cines.fr/textes/login.html (ou dans Utilisateurs puis Demande de login directement sur le site). Il est également nécessaire de lire puis de signer la charte de bon usage des ressources informatiques du CINES. On la trouve sur internet à l'adresse http://www.cines.fr/textes/charte2.html (ou dans Utilisateurs puis Charte). Lors de l'ouverture d'un login, un répertoire /home/login est créé. Chaque utilisateur dispose d'un espace personnel avec beaucoup d'espace disque. Néanmoins, il est conseillé de travailler dans le /tmpp pour lancer ses travaux lorsqu'un grand nombre de fichiers temporaires sont générés au cours des calculs, et ne sauvegarder sur le /home que les fichiers nécessaires. L'espace /tmpp est un espace de travail rapide, avec un espace disque important, directement sur les machines. Pour avoir accès à cet espace, il faut en faire la demande explicitement. **Attention**, **contrairement au /home, les données de cet espace de travail temporaire ne sont pas sauvegardées**. De même il est interdit d'utiliser l'espace /tmp.

Pour écrire :
M. Le Directeur du CINES
950 rue de Saint Priest
34097 Montpellier Cedex 5

Liste des Comités Thématiques :
CT1 : Environnement
CT2 : Mécanique des Fluides
CT3 : Milieux réactifs
CT4 : Astrophysique, géophysique, terre solide
CT5 : Electromagnétisme et plasmas chauds
CT6 : Mathématiques, mathématiques appliquées, systèmes modèles
CT7 : Systèmes moléculaires organisés et biologie
CT8 : Chimie quantique et modélisation moléculaire
CT9 : Physique, chimie et propriétés des matériaux

Tout au long de l'année, on peut visualiser sa consommation sur http://reser.cines.fr/ (ou dans Utilisateur puis Consultation des ressources consommées).

1.3 Assistance utilisateurs

D'une manière générale, toute demande de renseignements peut être adressée par messagerie électronique à svp@cines.fr.

L'ensemble de l'équipe assistance utilisateurs se tient à votre disposition pour vous aider dans l'exécution et la parallélisation de vos applications sur les machines de calcul :

Jean- Louis Ambrosino	Jean-louis.Ambrosino@cines.fr
Nicole Audiffren	Nicole.Audiffren@cines.fr
Michèle Batlle	Michele.Batlle@cines.fr
Philippe Falandry	Philippe.Falandry@cines.fr
Alain Mango	Alain.Mango@cines.fr
Georges Urbach	Georges.Urbach@cines.fr

2. Machine de calcul parallèle IBM

On trouve sur la machine IBM les langages Fortran, C et C++. **Cette machine est dédiée au calcul parallèle**. La parallélisation des programmes est réalisée soit en appelant des routines de passage de messages avec la bibliothèque MPI, soit avec des directives OpenMP. Néanmoins, il est toujours possible de soumettre des travaux monoprocesseur.

2.1 Compilation

Les options de compilations pour un programme tournant sur un seul processeur et pour un programme tournant sur plusieurs processeurs sont identiques.
La mise en œuvre de la compilation sur IBM est illustrée à partir de la compilation d'un fichier source écrit en Fortran **pi.f.** Les commandes essentielles de compilation et d'édition de liens sont présentées. Il est vivement recommandé d'utiliser un makefile qui permet de rassembler et d'exécuter l'ensemble des actions de compilation en utilisant la commande make. Des exemples de fichier makefile sont donnés en annexe.

2.1.1 Compilation monoprocesseur

Le compilateur en Fortran est xlf_r, xlc_r pour le C et xlC_r pour le C++. On trouve aussi des compilateurs purement Fortran 90 (xlf90) et purement Fortran 95 (xlf95). On peut obtenir des informations supplémentaires sur ces compilateurs en utilisant la commande **man** xlf.

On rappelle que l'on travaille avec :

pi.f : fichier source
pi.o : code objet
pi : exécutable

Deux étapes sont importantes dans la création d'un exécutable : la compilation des fichiers avec –c puis l'édition de liens avec –o et –llib. L'option –o nomme l'exécutable, tandis que -llib permet d'identifier une bibliothèque de sous-programmes utilisée par pi.f.

```
#-----------------------------------------------------------------------
xlf_r –c pi.f
#compile le fichier pi.f et crée le fichier objet pi.o
#-----------------------------------------------------------------------
#-----------------------------------------------------------------------
xlf_r –O3 -q64  –c  pi.f
#compile le fichier avec les options de compilation –O3 –q64
#-----------------------------------------------------------------------

#-----------------------------------------------------------------------
xlf_r –O3 -q64 –o pi pi.f
#crée l'exécutable et le renomme en pi grâce à l'option d'édition de
#liens –o (compilation de pi.f par défaut et édition de liens)
#-----------------------------------------------------------------------

#-----------------------------------------------------------------------
xlf_r –O3 -q64 –o pi -L/usr/oem/nag -lnag –lessl  pi.f
#crée l'exécutable, le renomme en pi grâce à l'option d'édition de
#liens –o, -lnag et –lessl permettent d'indiquer les bibliothèques
#utilisées en indiquant le répertoire /usr/oem/nag où trouver la
#librairie NAG (Numerical Algorithms Group)
#(compilation de pi.f par défaut et édition de liens)
#-----------------------------------------------------------------------
```

On exécute ce programme série en entrant directement **pi** ou **./pi**

Si le programme principal fait appel à des sous-programmes ssprog1.f et ssprog2.f, il faut les compiler séparément puis les assembler avec pi à l'édition de liens :

```
#-----------------------------------------------------------------------
xlf_r –O3 -q64 –c ssprog1.f
```

```
xlf_r –O3 -q64 –c ssprog2.f

xlf_r –O3 -q64 –o pi ssprog1.o ssprog2.o \
              –lessl /usr/local/lib/libfftw.a   pi.f
#crée l'exécutable, le renomme en pi grâce à l'option d'édition de
#liens –o, /usr/local/lib/libfftw.a et –lessl permettent d'indiquer les
#bibliothèques (ici FFTW et ESSL) utilisées
#(compilation de pi.f par défaut et édition de liens)
#-------------------------------------------------------------------
```

On peut remarquer que l'édition de liens d'une bibliothèque dépend de la bibliothèque utilisée. Elle sera détaillée en (**2.2.1**) en fonction de la bibliothèque.

On peut aussi faire (voir les fichiers makefile en annexe) :

```
#-------------------------------------------------------------------
xlf_r –O3 -q64 –c pi.f
xlf_r –O3 -q64 –c ssprog1.f
xlf_r –O3 -q64 –c ssprog2.f
xlf_r –O3 -q64 –o pi pi.o ssprog1.o ssprog2.o –lessl
#-------------------------------------------------------------------
```

Il est fortement conseillé d'utiliser l'option de compilation –C (qui équivaut à -qcheck) lors de la mise au point du programme et des premiers calculs. Cette option permet de déboguer le programme, notamment en vérifiant le dépassement éventuel des bornes des indices des tableaux. Par contre elle empêche certaines optimisations de l'option –O3 et il est donc recommandé de l'enlever une fois la mise au point du programme réalisée.

Les options importantes de compilation sont :
-C –qsigtrap -qsource : seulement lors du débogage du code
-c : compilation seule
-q64 : codage des données sur 64 bits (mais ne les passe pas en double précision, c'est seulement au niveau de l'adressage)
-O3 : plus haut niveau d'optimisation du programme. Il est conseillé de comparer les résultats obtenus avec –O3 et –O. S'ils diffèrent il faut absolument travailler avec l'option –O
-qflttrap : détection des erreurs de type division par zéro, de dépassement, en virgule flottante
-qsave ; -qnosave : par défaut l'option du compilateur xlf est save. Cette option définit si les variables locales sont sauvegardées ou non. Il ne faut pas les sauvegarder si on fait de l'OpenMP

On peut trouver toutes les options par défaut du compilateur xlf dans /etc/xlf.cfg.

Il est important de noter que le compilateur utilise l'option –q32 par défaut.

2.1.2 Compilation multiprocesseurs

Le compilateur Fortran en MPI est mpxlf_r, mpcc_r pour le C et mpCC_r pour le C++.
Le compilateur Fortran en OpenMP est xlf_r, xlc_r pour le C et xlC_r pour le C++.

On rappelle que les options de compilations pour un programme tournant sur un seul processeur et pour un programme tournant sur plusieurs processeurs sont identiques et que les plus importantes ont été données en (**2.1.1**). Seules la compilation et l'édition de liens vont changer. L'option de compilation –qnosave est nécessaire pour les programmes en OpenMP.

Pour un programme parallèle MPI :

```
#-----------------------------------------------------------------------
mpxlf_r –O3 -q64 –c pi.f
#compile le fichier pi.f et crée le fichier objet pi.o

mpxlf_r –O3 -q64 –o pi pi.o –lessl
#crée l'exécutable, le renomme pi grâce à l'option d'édition de
#liens –o, –lessl permet d'indiquer la bibliothèque utilisée
#-----------------------------------------------------------------------
```

Pour un programme parallèle OpenMP :
Le compilateur utilisé dans l'exemple xlf_r –qsmp=omp est nécessaire en OpenMP

```
#-----------------------------------------------------------------------
xlf_r  -qsmp=omp –O3 -q64 –qnosave –c pi.f
#compile le fichier pi.f et crée le fichier objet pi.o
#-----------------------------------------------------------------------
```

Ou encore

```
#-----------------------------------------------------------------------
xlf_r  -qsmp=omp –O3 -q64 –qnosave –o pi pi.o –lessl
#crée l'exécutable, le renomme pi grâce à l'option d'édition de
#liens –o, –lessl permet d'indiquer la bibliothèque utilisée
#-----------------------------------------------------------------------
```

2.2 Bibliothèques et progiciels

2.2.1 Bibliothèques scientifiques accessibles sur IBM

Les chemins d'accès et appels aux différentes librairies donnés ci-dessous sont **propres à la machine IBM du CINES.**

ESSL : Engineering and Scientific Subroutine Library : bibliothèque IBM optimisée. Elle inclut la BLAS (Basic Linear Algebra Subprograms) niveaux 1, 2 et 3. Elle inclut également certaines routines de LAPACK, des FFT et un générateur de nombres aléatoires. Même si les routines dans ESSL et dans LAPACK portent des noms similaires, il faut vérifier les arguments qui peuvent être légèrement différents.
Pour l'utiliser il faut coder **`-lessl`** lors de l'édition de liens (**voir 2.1.1** et **l'annexe**).

LAPACK : Linear Algebra PACKage.
Pour l'utiliser il faut coder **`/usr/local/pub/LAPACK/lapack.a`** lors de l'édition de liens (**voir 2.1.1** et **l'annexe**).

NAG : Numerical Algorithms Group.
Pour l'utiliser il faut coder **`-L/usr/oem/nag -lnag`** lors de l'édition de liens (**voir 2.1.1** et **l'annexe**).

FFTW : The Fastest Fourier Transform in the West : permet de calculer des Transformées de Fourrier Discrètes en une ou plusieurs dimensions pour des données réelles ou complexes en simple ou en double précision. Elle n'existe que sur 64 bits.
Pour l'utiliser il faut coder **`/usr/local/lib/libfftw.a`** lors de l'édition de liens (**voir 2.1.1** et **l'annexe**).
Les routines commençant par **s** sont réelles, simple précision. Celles commençant par **c** sont complexes, simple précision. Celles commençant par **d** sont réelles, double précision et celles commençant par **z** sont complexes, double précision.

Les bibliothèques ci-dessus s'utilisent avec un code série ou un code parallèle.

On trouve aussi des bibliothèques parallèles qui peuvent être utilisées avec des codes monoprocesseur pour optimiser les calculs :
ESSLSMP : calculs parallèles sur des machines à mémoire partagée.
Pour l'utiliser il faut coder **`-lesslsmp`** lors de l'édition de liens (**voir 2.1.1** et **l'annexe**).

FFTW : contient également une version parallèle adaptée à l'architecture mémoire distribuée MPI, en version réelle en simple ou en double précision. Elle n'existe que sur 64 bits.
Pour l'utiliser il faut coder **`/usr/local/lib/libfftw_mpi.a`** lors de l'édition de liens (**voir 2.1.1** et **l'annexe**).

2.2.2 Bibliothèques de sous programmes

La création d'une bibliothèque de routines Fortran ou d'autres langages est possible avec la commande *ar* du système UNIX. Par défaut, cette commande ne manipule que des objets codés sur 32 bits. Il est conseillé d'utiliser l'option –X64 à la création de la bibliothèque pour coder les objets sur 64 bits, comme c'est le cas dans les exemples ci-dessous.
Ce sont les fichiers (**.o**) qui contiennent les sous-programmes membres de la bibliothèque, créés avec l'option –c pour ne pas invoquer l'édition de liens.

```
#-----------------------------------------------------------------------
xlf –c -q64  ssprog1.f  ssprog2.f
ar –vq  -X64  libmybibli.a  ssprog1.o ssprog2.o
#-----------------------------------------------------------------------
```

où **mybibli** est le nom de la bibliothèque. Le fait de nommer cette librairie **libmybibli.a** et non simplement **mybibli.a** est très important car lors de l'édition de liens, **le -lmybibli rajoute lib devant mybibli**.

```
#-----------------------------------------------------------------------
ar –vt libmybibli.a                        #pour lister le contenu de la librairie
ar -vr libmybibli.a ssprog1.o ssprog3.o    #ssprog1 est remplacé, ssprog3 ajouté
ar -vru libmybibli.a ssprog2.o             #pour mettre à jour un élément
#-----------------------------------------------------------------------
```

Si le répertoire où se trouve la librairie s'appelle **/u/mylib**, l'utilisation de la bibliothèque à l'édition de liens s'écrit :

```
#-----------------------------------------------------------------------
xlf_r –O3 -q64 –c pi.f
xlf_r –O3 -q64 –o pi pi.o -L/u/mylib -lmybibli
#-----------------------------------------------------------------------
```

La commande *ranlib* appliquée à une archive permet de générer un index du contenu de la librairie. Une archive avec un tel index accélère l'édition des liens avec la librairie et permet à des routines de la librairie de s'appeler l'une l'autre sans s'occuper de leurs emplacements respectifs dans l'archive.

```
#-------------------------------------------------------------------
ranlib  -X64  libmybibli.a
#-------------------------------------------------------------------
```

2.2.3 Progiciels

Il existe un certain nombre de progiciels mis à disposition des utilisateurs, aussi bien en Chimie, qu'en Mécanique des Fluides ou en Calcul des Structures. On en trouve la liste à l'adresse internet http://www.cines.fr/textes/calc_parallele/bibliotheques.html (ou dans Matériels et Logiciels puis Calcul scientifique parallèle directement sur le site). Certains tournent sur IBM et d'autres sur SGI.

2.3 Soumission de travaux en interactif

Le temps de calcul est limité à 30 minutes en interactif.

Soumission d'un programme MPI :
Une fois que le programme est compilé, il faut créer un fichier qui s'appelle *host.list* avec autant de lignes ‘’zeus.cines.fr ‘’ que de processus désirés. Il faut ensuite créer le script définissant l'environnement MPI et lançant le programme :

```
#-------------------------------------------------------------------
poe pi –cpu_use multiple –shared_memory yes –procs 8 –labelio yes
#-------------------------------------------------------------------
```

On lance le programme (ici sur 8 processeurs) en soumettant le script.

Soumission d'un programme OpenMP :
Pour soumettre un programme écrit en OpenMP il faut positionner correctement la variable OMP_NUM_THREADS :

```
#-------------------------------------------------------------------
export OMP_NUM_THREADS=nombre de threads désirés
pi ou  ./pi
#-------------------------------------------------------------------
```

2.4 Classes d'exécution des travaux scientifiques

Le temps de calcul interactif est donc limité à 30 minutes. Il ne sert que pour des travaux tests et pour la mise au point des programmes. Pour des travaux nécessitant des temps supérieurs à cette limite et nécessitant beaucoup de mémoire, il faut utiliser la soumission **batch**. Le gestionnaire de travaux en batch sur IBM est le système **LoadLeveler**. Les classes sont attribuées automatiquement par la machine en fonction du temps maximum d'exécution prévu et du nombre de processeurs demandé pour l'exécution du travail.

La classe SE (travaux de plus de 12 heures) ne lance les travaux que dans une fenêtre allant du vendredi 16 heures au samedi 23 heures.

	1-16	17-64	>64
<30mn			
30mn-12H			
12H-48H			

Légende du tableau avec le nom des classes correspondant :

A	**B**	**SE**

Les scripts de soumission correspondants sont décrits ci-dessous, pour un programme série, pour un programme MPI et pour un programme OpenMP.

2.5 Script de soumission en batch monoprocesseur

Le script de soumission, **script.cmd**, est le suivant :

```
#!/bin/ksh
#
#-----------------------------------------------------------------------
# script.cmd pour soumission d'un job serie
#
#  Configuration :
#
#  step_name        =     commentaire qui apparait pendant le suivi du
#                         job lors des calculs
```

```
#  initialdir       =     localisation de l'exécutable (par défaut là
#                         où on a envoyé le script)
#  ouput            =     fichier de sortie, nom au choix
#  error            =     fichier d'erreur, nom au choix
#  wall_clock_limit =     HH:MM:SS ou MM:SS ou SS (temps max prévu pour
#                         l'exécution). 30 minutes par défaut
#  job_type         =     serial
#  total_tasks      =     nombre de processeurs minimum nécessaires au
#                         programme : ici forcément 1
#  resources        =     CPU demandés pour un processus, donc 1
#  notify_user      =     e-mail de l'utilisateur pour notification de
#                         fin de job, ou d'erreurs
#  notification     =     type d'informations contenues dans le mail
#  environment      =     pour copie de son environnement
#
#------------------------------------------------------------------------
# @ step_name        = code_serie
# @ initialdir       = /home/login
# @ input            = /dev/null
# @ output           = sortie_machine
# @ error            = erreur_machine
# @ wall_clock_limit = 29:00
# @ job_type         = serial
# @ total_tasks      = 1
# @ resources        = ConsumableCpus(1)
# @ notify_user      = email
# @ notification     = complete
# @ environment      = COPY_ALL;          \
#                      LANG=En_US
# @ restart          = no
# @ queue
     ./pi
#------------------------------------------------------------------
```

2.6 Script de soumission en batch avec MPI

```
#!/bin/ksh
#
#------------------------------------------------------------------------
# script.cmd pour soumission de jobs parallèles MPI
#
#  Configuration :
#
#  step_name        =     commentaire qui apparait pendant le suivi du
#                         job lors des calculs
#  initialdir       =     localisation de l'exécutable (par défaut là
#                         où on a envoyé le script)
#  ouput            =     nom au choix.  $(cluster) permet de lancer
#                         plusieurs scripts à la suite sans les
#                         renommer
#  error            =     nom au choix.  $(cluster) permet de lancer
```

```
#                          plusieurs scripts à la suite sans les
#                          renommer
#  wall_clock_limit =      HH:MM:SS ou MM:SS ou SS (temps max prévu pour
#                          l'exécution). 30 minutes par défaut
#  job_type         =      parallel
#  total_tasks      =      nombre de processeurs minimum nécessaires au
#                          programme (> 2 en parallèle), c'est aussi le
#                          nombre total de processus MPI
#  resources        =      chaque processus MPI utilise 1 CPU
#  notify_user      =      e-mail de l'utilisateur pour notification de
#                          fin de job, ou d'erreurs
#  environment      =      pour copie de son environnement et définition
#                          de l'environnement MPI
#
#------------------------------------------------------------------------
# @ step_name        = code_parallele_mpi
# @ initialdir       = /home/login
# @ input            = /dev/null
# @ output           = sortie_machine.$(cluster)
# @ error            = erreur_machine.$(cluster)
# @ wall_clock_limit = 1:29:00
# @ job_type         = parallel
# @ total_tasks      = 8
# @ resources        = ConsumableCpus(1)
# @ notify_user      = email
# @ notification     = complete
# @ environment      = COPY_ALL;          \
#                      LANG=En_US;
#                      MP_SHARED_MEMORY=YES; \
#                      MP_EUILIB=us;      \
#                      MP_HOSTFILE=NULL;  \
#                      MP_LABELIO=yes; \
#                      MP_RMPOOL=0;       \
#                      MP_PROCS=$(total_tasks)
# @ restart          = no
# @ queue
     poe  pi
#------------------------------------------------------------------
```

2.7 Script de soumission en batch avec OpenMP

```
#!/bin/ksh
#
#------------------------------------------------------------------------
# script.cmd pour soumission de jobs parallèles OpenMP
#
#  Configuration :
#
#  step_name        =     commentaire qui apparait pendant le suivi du
#                         job lors des calculs
#  initialdir       =     localisation de l'exécutable (par défaut là
```

```
#                          où on a envoyé le script)
#  ouput            =      nom au choix   $(cluster) permet de lancer
#                          plusieurs scripts à la suite sans les
#                          renommer
#  error            =      nom au choix  $(cluster) permet de lancer
#                          plusieurs scripts à la suite sans les
#                          renommer
#  wall_clock_limit =      HH:MM:SS ou MM:SS ou SS (temps maximum prévu
#                          pour l'exécution). 30 minutes par défaut
#  job_type         =      parallel
#  total_tasks      =      doit toujours être égal à 1 en OpenMP
#  resources        =      nombre de threads tournant en parallèle,
#                          c'est aussi le nombre de processeurs
#  notify_user      =      e-mail de l'utilisateur pour notification de
#                          fin de job, ou d'erreurs
#  environment      =      pour copie de son environnement
#  nombre de threads =     nombre de threads sur lesquel sera
#                          parallélisé le code (au maximum 32)
#
#------------------------------------------------------------------------
# @ step_name        = code_parallele_omp
# @ initialdir       = /home/login
# @ input            = /dev/null
# @ output           = sortie_machine.$(cluster)
# @ error            = erreur_machine.$(cluster)
# @ wall_clock_limit = 1:29:00
# @ job_type         = parallel
# @ total_tasks      = 1
# @ resources        = ConsumableCpus(nombre de threads)
# @ notify_user      = email
# @ notification     = complete
# @ environment      = COPY_ALL;          \
#                      LANG=En_US;        \
#                      OMP_NUM_THREADS  = nombre de threads
# @ restart          = no
# @ queue
    export XLSMPOPTS=spins=0 : yields=0
     ./pi
#------------------------------------------------------------------------
```

2.8 Script hybride de soumission en batch MPI-OpenMP

```
#!/bin/ksh
#
#------------------------------------------------------------------------
# script.cmd pour soumission de jobs parallèles mixtes MPI-OpenMP
#
```

```
#  Configuration :
#
#  step_name        =     commentaire qui apparait pendant le suivi du
#                         job lors des calculs
#  initialdir       =     localisation de l'exécutable (par défaut là
#                         où on a envoyé le script)
#  ouput            =     nom au choix   $(cluster) permet de lancer
#                         plusieurs scripts à la suite sans les
#                         renommer
#  error            =     nom au choix  $(cluster) permet de lancer
#                         plusieurs scripts à la suite sans les
#                         renommer
#  wall_clock_limit =     HH:MM:SS ou MM:SS ou SS (temps maximum prévu
#                         pour l'exécution). 30 minutes par défaut
#  job_type         =     parallel
#  node             =     nombre de nœuds attribués au job
#                         (indispensable en hybride)
#  tasks_per_node   =     nombre de processus MPI par nœud
#                         (indispensable en hybride)
#  resources        =     nombre de threads par processus MPI
#                         (indispensable en hybride)
#  notify_user      =     e-mail de l'utilisateur pour notification de
#                         fin de job, ou d'erreurs
#  environment      =     pour copie de son environnement et définition
#                         de l'environnement MPI-OpenMP
#  nombre de threads =    nombre de threads par processus MPI pour
#                         OpenMP
#
#------------------------------------------------------------------------
# @ step_name        = code_parallele_OpenMPI
# @ initialdir       = /home/login
# @ input            = /dev/null
# @ output           = sortie_machine.$(cluster)
# @ error            = erreur_machine.$(cluster)
# @ wall_clock_limit = 1:29:00
# @ job_type         = parallel
# @ node             = 3
# @ tasks_per_node   = 4
# @ resources        = ConsumableCpus(nombre de threads)
# @ notify_user      = email
# @ notification     = complete
# @ environment      = COPY_ALL;          \
#                      LANG=En_US;
#                      MP_SHARED_MEMORY=YES; \
#                      MP_EUILIB=us;      \
#                      MP_HOSTFILE=NULL;  \
#                      MP_LABELIO=yes; \
#                      MP_RMPOOL=0;       \
#                      MP_PROCS=$(node)*$(tasks_per_node); \
                       OMP_NUM_THREADS=nombre de threads par processus
                                        MPI
# @ restart          = no
# @ queue
    export XLSMPOPTS=spins=0 : yields=0
```

```
    poe pi
#------------------------------------------------------------------
```

2.9 Commandes LoadLeveler

Les commandes de base sont les suivantes :
llsubmit script.cmd : soumission du job
llq –u userid : pour l'affichage de l'état des travaux
llcancel host.cluster.proc : pour détruire un travail
llq –l host.cluster.proc : donne un rapport complet sur le job référencé

Voici un exemple de sortie de la commande llq (r21 et r33 sont deux des nœuds de la machine):

Id	**Owner**	**Submitted**	**ST**	**PRI**	**Class**	**Running On**
r25.17695.0	utilis1	7/24 13:44	R	50	A	r21
r21.17743.0	utilis2	7/25 14:49	R	50	SE	r33
r25.17766.0	utilis4	7/26 12:11	I	50	SE	
r21.17794.0	utilis5	7/26 12:38	I	50	SE	
r25.17707.0	utilis6	7/25 11:40	C	50	A	

Status des jobs
I : job en attente de démarrage
R : job en cours d'exécution
C : job terminé normalement
CA : job supprimé
RM : temps CPU alloué dépassé

3. Machine de calcul parallèle SGI

3.1 Compilation

On trouve sur la machine SGI les langages Fortran, C et C++. **Cette machine est également dédiée au calcul parallèle**. La parallélisation des programmes est réalisée soit en appelant des routines de passage de messages avec la bibliothèque MPI, soit avec des directives OpenMP. Néanmoins, comme sur la machine IBM, il est toujours possible de soumettre des travaux monoprocesseur.

Les compilateurs sont f77 pour le fortran 77, f90 pour le fortran 90, cc pour le C et CC pour le C++. La construction d'un exécutable sur SGI reste la même que sur IBM (**voir 2.1.1** et **l'annexe**). Seuls les compilateurs, les options de compilation et certaines librairies vont changer.

Pour un programme parallèle MPI :
La bibliothèque –lmpi utilisée dans l'exemple est nécessaire en MPI

```
#-----------------------------------------------------------------------
f90 –O3 -64 –c pi.f
#compile le fichier pi.f et crée le fichier objet pi.o

f90 –O3 -64 –o pi pi.o –lmpi -lscs
#crée l'exécutable, le renomme pi grâce à l'option d'édition de
#liens –o, –lmpi nécessaire pour faire du MPI, -lscs pour utiliser la
#bibliothèque scs par exemple
#-----------------------------------------------------------------------
```

Pour un programme parallèle OpenMP :
Le compilateur utilisé dans l'exemple f90 –mp est nécessaire en OpenMP

```
#-----------------------------------------------------------------------
f90 -mp –O3 -64 –c pi.f
#compile le fichier pi.f et crée le fichier objet pi.o, -mp nécessaire
#en OpenMP

f90 –mp –O3 -64 –o pi pi.o -lscs
#crée l'exécutable, le renomme pi grâce à l'option d'édition de
#liens –o, -lscs pour utiliser la bibliothèque scs par exemple
```

Les options importantes de compilation sont :
-C -g : seulement lors du débogage du code
-c : compilation seule
-O3 : plus haut niveau d'optimisation du programme. Il est conseillé de comparer les résultats obtenus avec –O3 et –O. S'ils diffèrent il faut absolument travailler avec l'option –O
-64 : codage des données sur 64 bits (mais ne les passe pas en double précision, c'est seulement au niveau de l'adressage)

3.2 Bibliothèques et progiciels

3.2.1 Bibliothèques scientifiques accessibles sur SGI

Les chemins d'accès et appels aux différentes librairies donnés ci-dessous sont **propres à la machine SGI du CINES.**

SCSL : Silicon Graphics / Cray Scientific Library : bibliothèque optimisée pour la SGI. Elle inclut la BLAS (Basic Linear Algebra Subprograms) niveaux 1, 2 et 3. Elle inclut également LAPACK, des FFT et un générateur de nombres aléatoires.
Pour l'utiliser il faut coder **`-lscs`** lors de l'édition de liens (**voir 2.1.1** et **l'annexe**).
Les routines commençant par **s** sont réelles, simple précision. Celles commençant par **c** sont complexes, simple précision. Celles commençant par **d** sont réelles, double précision et celles commençant par **z** sont complexes, double précision.

NAG : Numerical Algorithms Group.
Pour l'utiliser il faut coder **`-lnag`** lors de l'édition de liens (**voir 2.1.1** et **l'annexe**) puis
`. /usr/oem/flsg619da/nag.kshrc` avant d'exécuter le programme.

FFTW : The Fastest Fourier Transform in the West : permet de calculer des Transformées de Fourrier Discrètes en une ou plusieurs dimensions pour des données réelles ou complexes en simple ou double précision.
Pour l'utiliser en 64 bits il faut coder (**voir 2.1.1** et **l'annexe**) :
`/usr/local/lib/fftw2.1.5/lib64/libfftw.a` lors de l'édition de liens
Les routines commençant par **s** sont réelles, simple précision. Celles commençant par **c** sont complexes, simple précision. Celles commençant par **d** sont réelles, double précision et celles commençant par **z** sont complexes, double précision.

Les bibliothèques ci-dessus s'utilisent avec un code série ou un code parallèle.

On trouve aussi des bibliothèques parallèles qui peuvent être utilisées avec des codes monoprocesseur pour optimiser les calculs :
SCS_MP : calculs parallèles sur des machines à mémoire partagée

Pour l'utiliser il faut coder **`-lscs_mp`** lors de l'édition de liens (**voir 2.1.1** et **l'annexe**).

FFTW : contient également une version parallèle adaptée à l'architecture mémoire distribuée MPI, en version réelle ou complexe en simple ou double précision.
Pour l'utiliser en 64 bits il faut coder (**voir 2.1.1** et **l'annexe**) :
/usr/local/lib/fftw2.1.5/lib64/libfftw_mpi.a lors de l'édition de liens.

3.2.2 Bibliothèques de sous programmes

La création et l'utilisation de librairies personnelles créées à partir de fichiers Fortran ont été décrites pour la machine IBM (paragraphe **2.2.2**). Elles restent les mêmes puisqu'elles ne dépendent que de commandes UNIX.

3.2.3 Progiciels

Il existe un certain nombre de progiciels mis à disposition des utilisateurs, aussi bien en Chimie, qu'en Mécanique des Fluides ou en Calcul des Structures. On en trouve la liste à l'adresse internet http://www.cines.fr/textes/calc_parallele/bibliotheques.html.
Certains tournent sur IBM et d'autres sur SGI.

3.3 Soumission des travaux en interactif

Le temps de calcul est limité à 30 minutes en interactif.

Soumission d'un programme MPI :
Une fois que le programme est compilé on le lance par :

```
#-----------------------------------------------------------------------
mpirun  -np procs  pi
#-----------------------------------------------------------------------
```

Soumission d'un programme OpenMP :
Pour soumettre un programme écrit en OpenMP il faut positionner correctement la variable OMP_NUM_THREADS :

```
#-----------------------------------------------------------------------
export OMP_NUM_THREADS=nombre de threads désirés
pi
#-----------------------------------------------------------------------
```

3.4 Classes d'exécution des travaux scientifiques

Comme sur la machine IBM, le temps de calcul interactif est limité à 30 minutes. Il ne sert que pour des travaux tests et pour la mise au point des programmes. Pour des travaux nécessitant des temps supérieurs à cette limite il faut utiliser la soumission **batch**. Le système utilisé pour soumettre les jobs est le système **LSF** (Load Sharing Facilities). Les classes sont attribuées automatiquement par la machine en fonction du temps maximum d'exécution prévu et du nombre de processeurs demandé pour l'exécution du travail.

Les classes session_s et session_l (travaux de plus de 24 heures) ne lancent les travaux que dans une fenêtre allant du vendredi 16 heures au samedi 23 heures.

Les scripts de soumission correspondants sont décrits ci-dessous pour un programme série, pour un programme MPI et pour un programme OpenMP.

	1-16	16-32	32-64	64-128	128-256	256-500
<30mn						
30mn-24H						
24H-100H						

Légende du tableau avec le nom des classes correspondant :

court	**long_s**	**long_m**	**long_l**	**long_xl**	**session_s**	**session_l**

3.5 Script de soumission en batch monoprocesseur

Il n'y a pas de classes réservées aux travaux monoprocesseur. Le script de soumission, **script.lsf**, est le suivant :

```
#!/bin/sh
#
#------------------------------------------------------------------------
# script.lsf pour soumission d'un job serie
#
#  Configuration :
#
#  -J               =     pour nommer le job
#  -i               =     fichier d'entrée (/dev/null si pas de
#                         fichier d'entrée)
#  -N               =     pour notification
#  -u               =     email
#  -o               =     fichier de sortie, nom au choix
#  -e               =     fichier d'erreur, nom au choix
#  -n               =     nombre de processeurs demandés
#                         (forcément 1 ici)
#  -W               =     HH:MM (temps max prévu pour l'exécution)
#                         par défaut 30 minutes
#  -M               =     mémoire utilisée (maxi 500000 en koctet)
#  %J               =     permet de spécifier le numéro du travail
#
#------------------------------------------------------------------------
#BSUB –J       code_serie
#BSUB –i       /dev/null
#BSUB –N –u    email
#BSUB –o       sortie_machine.%J
#BSUB –e       erreur_machine.%J
#BSUB –n       1
#BSUB –W       00:30
#BSUB –M       500000
     ./pi
#-------------------------------------------------------------------
```

3.6 Script de soumission en batch avec MPI

```
#!/bin/sh
#
#------------------------------------------------------------------------
# script.lsf pour soumission de jobs parallèles MPI
#
#  Configuration :
#
#  -J               =     pour nommer le job
#  -i               =     fichier d'entrée (/dev/null si pas de
#                         fichier d'entrée)
#  -N               =     pour notification
#  -u               =     email
#  -o               =     fichier de sortie, nom au choix
#  -e               =     fichier d'erreur, nom au choix
```

```
#  -n               =     nombre de processeurs demandés (égal au
#                         nombre de np dans mpirun)
#  -W               =     HH:MM (temps max prévu pour l'exécution)
#                         par défaut 30 minutes
#  -M               =     mémoire utilisée en koctet: taille du code
#                         (maxi 500000) x nombre de processeurs
#  %J               =     permet de spécifier le numéro du travail
#
#------------------------------------------------------------------------
#BSUB –J       code_parallele_mpi
#BSUB –i       /dev/null
#BSUB –N –u    email
#BSUB –o       sortie_machine.%J
#BSUB –e       erreur_machine.%J
#BSUB –n       8
#BSUB –W       01:00
#BSUB –M       4000000
     mpirun –np 8 pi
#------------------------------------------------------------------------
```

3.7 Script de soumission en batch avec OpenMP

```
#!/bin/sh
#
#------------------------------------------------------------------------
# script.lsf pour soumission de jobs parallèles OpenMP
#
#  Configuration :
#
#  -J               =     pour nommer le job
#  -i               =     fichier d'entrée (/dev/null si pas de
#                         fichier d'entrée)
#  -N               =     pour notification
#  -u               =     email
#  -o               =     fichier de sortie, nom au choix
#  -e               =     fichier d'erreur, nom au choix
#  -n               =     nombre de processeurs demandés (égal au
#                         nombre de threads de OMP_NUM_THREADS)
#  -W               =     HH:MM (temps max prévu pour l'exécution)
#                         par défaut 30 minutes
#  -M               =     mémoire utilisée : taille du code (maxi
#                         500000) x nombre de processeurs
#  %J               =     permet de spécifier le numéro du travail
#
#------------------------------------------------------------------------
#BSUB –J       code_parallele_omp
#BSUB –i       /dev/null
#BSUB –N –u    email
#BSUB –o       sortie_machine.%J
#BSUB –e       erreur_machine.%J
#BSUB –n       8
```

```
#BSUB –W       01:00
#BSUB –M       4000000
     export OMP_NUM_THREADS=8
     pi
#-------------------------------------------------------------------
```

3.8 Commandes LSF

Les commandes de base sont les suivantes :
bsub < script.lsf : soumission du job
bjobs : donne le status des jobs de l'utilisateur
bjobs -a : affichage du statut des travaux en cours et terminés
bjobs -p : affichage des raisons du ''pending'' (attente du démarrage)
bjobs -l : affichage des informations détaillées pour chaque job
bkill numéro : pour tuer un job
bpeek : les fichiers d'erreur et de sortie ne sont consultables qu'à la fin du job. Cette commande permet de les visualiser au cours du calcul

Voici un exemple de sortie de la commande bjobs :

JOBID	USER	STAT	QUEUE	FROM_HOST	EXEC_HOST	JOB_NAME	SUBMIT_TIME
491825	utilis1	RUN	long_s	athena	4*minerve	*e/utilis1	Jul 27 16:39
491811	utilis1	PEND	court	athena		*parallele	Jul 27 17:39

Status des jobs
PEND : job en attente de démarrage
RUN : job en cours d'exécution
DONE : job terminé normalement
EXIT : job terminé avec problèmes

4. Sauvegarde des données

Le CINES dispose toujours d'une version sauvegardée des anciens fichiers. Les fichiers créés ou modifiés pendant la journée sont sauvegardés chaque nuit. Ensuite pour des fichiers modifiés tous les jours, on ne dispose au maximum que d'une version par jour sur les sept derniers jours. Après la destruction d'un fichier, la sauvegarde de celui-ci est

conservée 7 jours. En cas de destruction accidentelle d'un fichier vous pouvez contacter la cellule support aux utilisateurs svp@cines.fr pour une éventuelle restauration. **Attention**, si le fichier créé dans la journée est détruit avant le soir, il n'existera pas de version sauvegardée de ce fichier (les sauvegardes étant lancées la nuit).
Une fois les ressources consommées et/ou en cas de non renouvellement d'un projet scientifique, les données sont sauvegardées par le CINES et mis à disposition de l'utilisateur pour une période d'un an.

5. Visualisation Scientifique

Le CINES met à disposition des scientifiques de toutes disciplines des ressources informatiques nécessaires au calcul, à la visualisation et à l'animation d'images. Les données communes du répertoire /home/login des machines IBM et SGI sont aussi communes à cette machine grâce au même montage NFS.

Vous pouvez bénéficier :

- De l'utilisation à distance des moyens graphiques(matériels et logiciels) du CINES, grâce au logiciel de partage des ressources Vizserver
- De la création d'outils graphiques spécifiques à votre problématique (OpenDX)
- D'une aide à la réalisation d'animations complexes complètes MPEG
- D'un accueil scientifique pour réaliser des visualisations
- D'une formation spécifique au logiciel OpenDX

Toute demande de renseignements peut être adressée par mail à :

Philippe Falandry Philippe.Falandry@cines.fr

A. Annexe : Makefile

L'utilisation d'un fichier makefile pour compiler ses programmes est très utile. Il suffit de lancer la commande make dans le répertoire où se trouve le fichier makefile pour générer l'exécutable lorsque des fichiers sources ont été modifiés. De nombreuses macros sont définies de façon implicite, la commande (make –p) permet de les lister. **Toutes les commandes `$(F90)…, $(FLINKER)…, rm –f …` doivent être précédées d'une tabulation. Les symboles # correspondent à des commentaires.**

Pour un seul programme source on a par exemple le makefile suivant :

```
#----------------------------------------------------------------------
F90          =         xlf_r
#F90 : compilateur ; xlf_r mieux que xlf avec q64
FLINKER      =         xlf _r
# FLINKER : édition de liens
FFLAGS       =         -O3  -q64
# FFLAGS : options de compilation
LDFLAGS      =          -q64
# LDFLAGS : options à l'édition de liens
FLIBS        =         -lessl
#FLIBS : édition de liens avec la bibliothèque essl
all          =          pi
#all : nom de l'exécutable pi

pi : pi.o
#pi : il faut lui donner les objets dont il dépend
      $(FLINKER) $(FFLAGS) $(LDFLAGS) –o pi pi.o $(FLIBS)
      #cette ligne de commande permet ensuite de le créer

.f.o:
#.f.o : les cibles .o dépendent des fichiers de même nom .f
      $(F90) $(FFLAGS) –c *.f
      #cette ligne de commande permet de compiler le fichier .f

clean :
      rm  –f  *.o pi
#make clean nettoie le directory des .o et de l'exécutable
#----------------------------------------------------------------------
```

Pour un programme comportant plusieurs fichiers sources on a par exemple le makefile suivant :

```
#----------------------------------------------------------------------
F90          =          xlf_r
#F90 : compilateur ; xlf_r mieux que xlf avec q64
FLINKER      =          xlf_r
#FLINKER : édition de liens
```

```
FFLAGS       =          -O3 -q64
#FFLAGS : options de compilation
LDFLAGS      =           -q64
#LDFLAGS : options à l'édition de liens
FLIB_PATH = -L/usr/oem/nag
#indique le répertoire où trouver la librairie nag
LIBS_LIST = -lnag –lessl
#liste des bibliothèques utilisées pour l'édition de liens
FLIBS = $(FLIB_PATH) $(FLIBS_LIST)
#FLIBS : édition de liens avec les bibliothèques essl et nag
OBJS         =          ssprog1.o ssprog2.o
#OBJS : définition d'une macro avec les sous programmes appelés
#dans le programme principal et compilés séparément
all          =          pi
#all : nom de l'exécutable

pi : pi.o  $(OBJS)
#pi : il faut lui donner les objets dont il dépend
      $(FLINKER) $(FFLAGS) $(LDFLAGS) –o pi pi.o $(OBJS) $(FLIBS)
      #cette ligne de commande permet ensuite de le créer

.f.o :
#.f.o : les cibles .o dépendent des fichiers de même nom .f
      $(F90) $(FFLAGS) –c *.f
      #cette ligne de commande permet de compiler le fichier .f

pi.o : pi.f ssprog1.f ssprog2.f
#pi.o : donne les dépendances explicitement

ssprog1.o : ssprog1.f ssprog2.f
#ssprog1.o : donne les dépendances explicitement

clean :
      rm  –f  *.o pi
#make clean nettoie le directory des .o et de l'exécutable
#---------------------------------------------------------------------
```