Rappel :

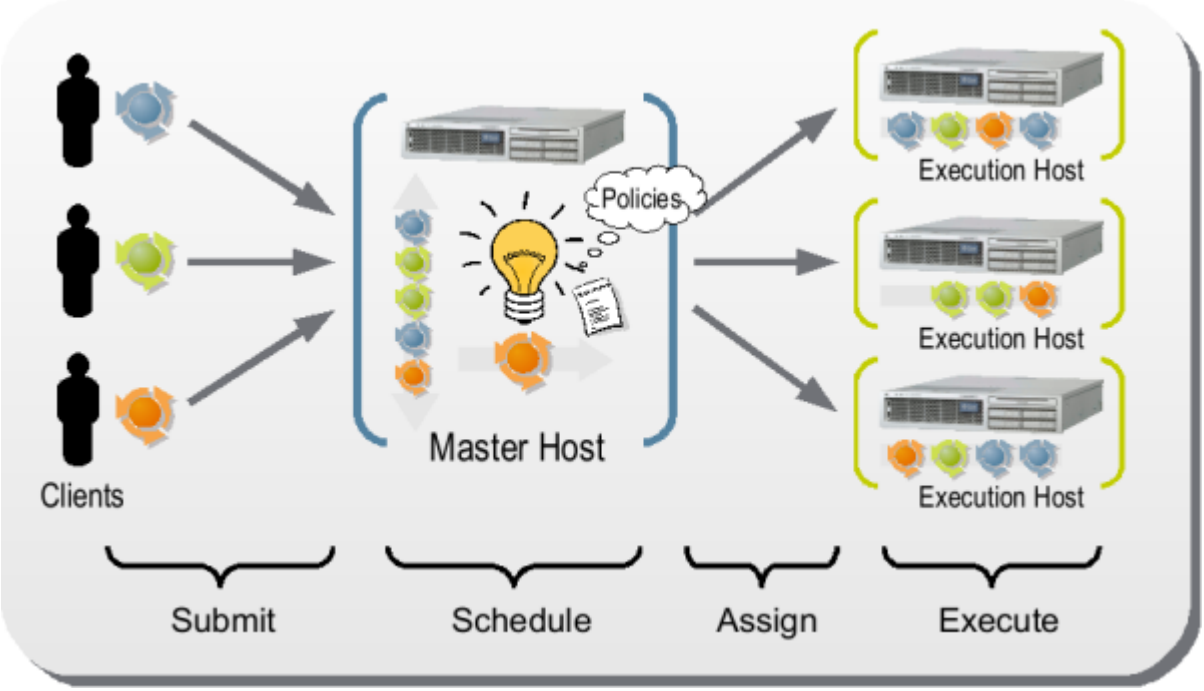
Les noeuds de soumission de jobs sont :

- mesoshared.univ-fcomte.fr
- mesologin1.univ-fcomte.fr
- mesologin2.univ-fcomte.fr

Grid Engine

SGE est un système batch avec répartition automatique de charge et gestion de files d'attente. SGE permet :

- Le partage des ressources
- La répartition des jobs soumis dans des files d'attente (queues)
- L'ordonnancement des jobs en fonction des requêtes utilisateur et de la charge des noeuds
- Le lancement des jobs éligibles sur un ou plusieurs processeurs
- La mise en attente des jobs restants



En pratique :

1. L'utilisateur soumet un job avec des paramètres spécifiques
2. Le système de batch (GE) lancera alors le job dès que les ressources requises par l'utilisateur seront disponibles

Définitions

Queue : une queue est une file d'attente dans laquelle s'accumulent les jobs en attente de traitement par l'ordonnanceur.

Job : une tâche ou job est un petit programme (shell) contenant la définition de l'environnement dans lequel il souhaite lancer son calcul, ainsi que les commandes nécessaires au lancement du traitement.

Organisation des files d'attente au Mésocentre

Suivre ce lien Organisation des files d'attente

Soumission de jobs interactifs

Problème :

- Ouvrir un shell sur un noeud particulier. Utile pour la compilation et le post-traitement par exemple.
- Lancer une programme directement sur un nœud de calcul en mode interactif

Solution :

1. `qlogin` pour se connecter sur un noeud de calcul en mode interactif
2. `qrsh` pour lancer directement un programme sur un noeud en mode interactif

Exemple 1 :

```
[user@mesologin1~]$ qlogin

[user@node1-13 ~]
```

il est possible de quitter ce mode interactif à l'aide de la commande `exit` ou du raccourci clavier `CTRL-D`

Exemple 2 : se connecter à un noeud particulier (**utilise la queue par défaut : all.q**)

```
[user@mesologin1 ~]$ qlogin -l hostname=node1-20
```

Exemple 2.1 : se connecter à un noeud particulier d'une queue donnée, par exemple on se connecte sur les machines de la queue **tesla.q** ou **bigmem**

```
[user@mesologin1 ~] qlogin  -q tesla.q
```

Exemple 3 : demander un noeud avec 16 coeurs pendant 8h

```
[user@mesologin1 ~] qlogin  -q all.q -pe openmp 16 -l h_rt=8:00:00
```

Exemple 4 : lancer un programme directement sur le noeud GPU (CTL +C) pour l'arrêter.

```
$ qrsh -q tesla.q  PGI_ACC_TIME=1 ./acc_mm.pgi.exe
```

Soumission passive de jobs (mode batch)

Pour lancer des jobs de manière asynchrone, c'est à dire sans avoir besoin d'attendre la fin de l'exécution du calcul, il est nécessaire d'utiliser la commande `qsub`.

Une aide exhaustive est disponible à l'aide de la commande suivante :

```
$ qsub -help
$ man qsub
```

Généralement :

```
$ qsub [options] [scriptfile]
```

Où **options** correspond aux ressources demandées, par exemple:

- file d'attente
- mémoire
- nombre de nœuds
- temps d'exécution
- ...

et **scriptfile** représente le script de lancement de l'application

Toute ces options peuvent être également intégrées dans le script, de manière à être appliquée à toute utilisation du script de soumission.

Options principales de SGE

Option	Description
`#$ -q [queue]`	Queue (file d'attente) à utiliser
`#$ -N [job]`	Nom du job
`#$ -V`	Exporte toutes les variables d'environnement
`#$ -cwd`	Utiliser le répertoire courant comme répertoire de travail
`#$ -o [outfile]`	Nom du fichier où stocker la sortie standard
`#$ -e [errfile]`	Nom du fichier où stocker les erreurs
`#$ -pe [par]`	Environnement parallèle à utiliser

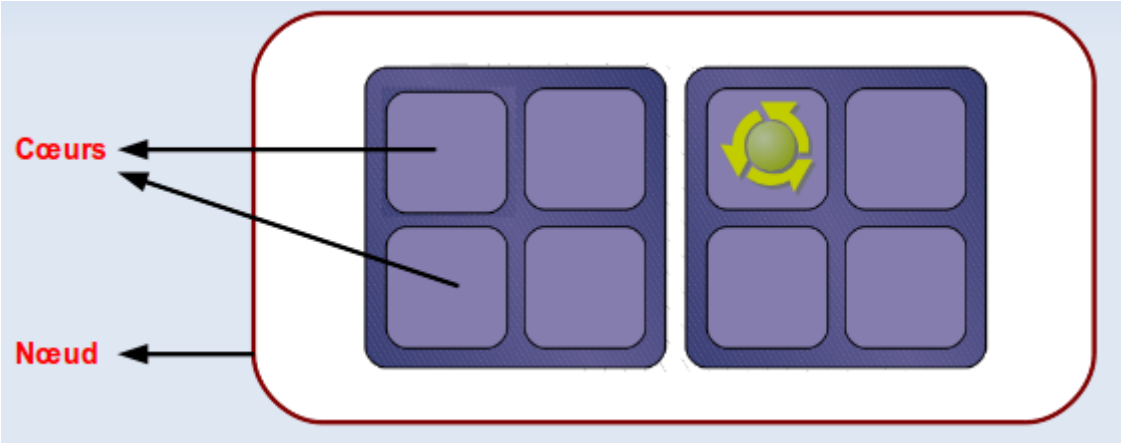
Variables d'environnement

Dans le script SGE, on peut utiliser les variables d'environnement suivantes :

Variable	Description
`$JOB_ID`	Identifiant unique propre au job
`$JOB_NAME`	Nom attribué au job (par défaut, correspond au nom du script)
`$TMPDIR`	Répertoire temporaire de SGE, supprimé à la fin de l'exécution du job
`$SGE_TASK_ID`	Identificant de la tâche courante en cas d'utilisation de tableaux de tâches (CF. Job parallèle)
`$NSLOTS`	Nombre de cœurs demandés par le script (Voir Job parallèle)

Soumission d'un job séquentiel

Un job séquentiel s'exécute sur un et un seul cœur d'un nœud particulier



Exemple de script :

script.sge

```
#!/bin/bash
#$ -V
#$ -N test_sge
#$ -cwd
#$ -o $JOB_NAME.$JOB_ID.out
#$ -e $JOB_NAME.$JOB_ID.err



pwd
sleep 30
```

un autre exemple :

script.sge

```
#!/bin/bash
#$ -V
#$ -N test_sge
#$ -cwd
#$ -o $JOB_NAME.$JOB_ID.out
#$ -e $JOB_NAME.$JOB_ID.err


time ./monProgrammeCompile
```

Soumission du job :

```
$ qsub script.sge
```

Visualisation de l'état :

```
$ qstat
job-ID  prior   name       user         state submit/start at     queue
slots ja-task-ID
-----------------------------------------------------------------------------
-------------------------------------
  90312 0.74999 test_sge   Username     r     07/01/2010 14:38:21
all.q@node1-31                1
```

Soumission d'un job parallèle

Un job parallèle s'exécute sur plusieurs coeurs

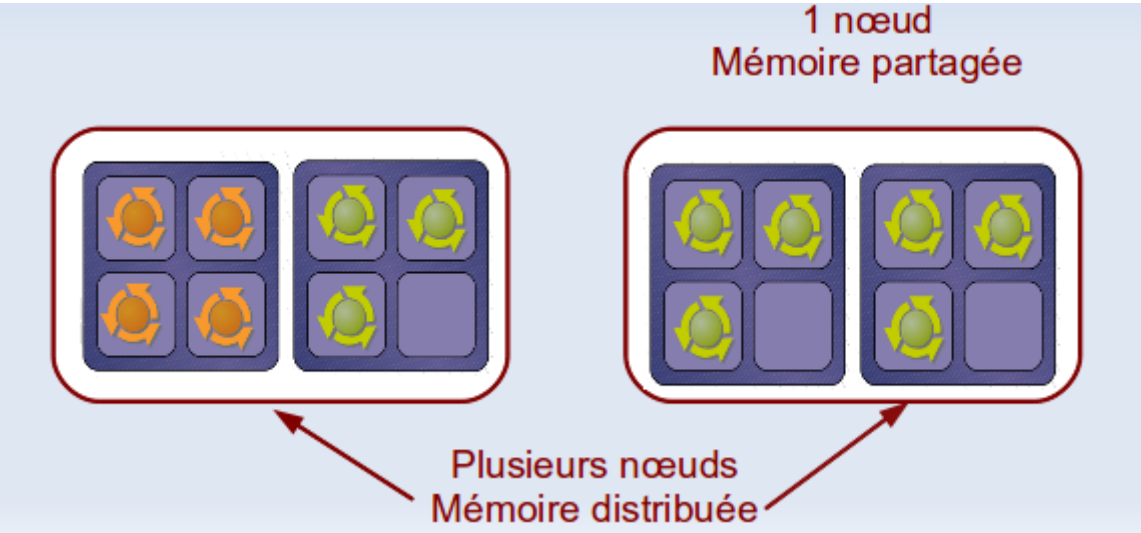
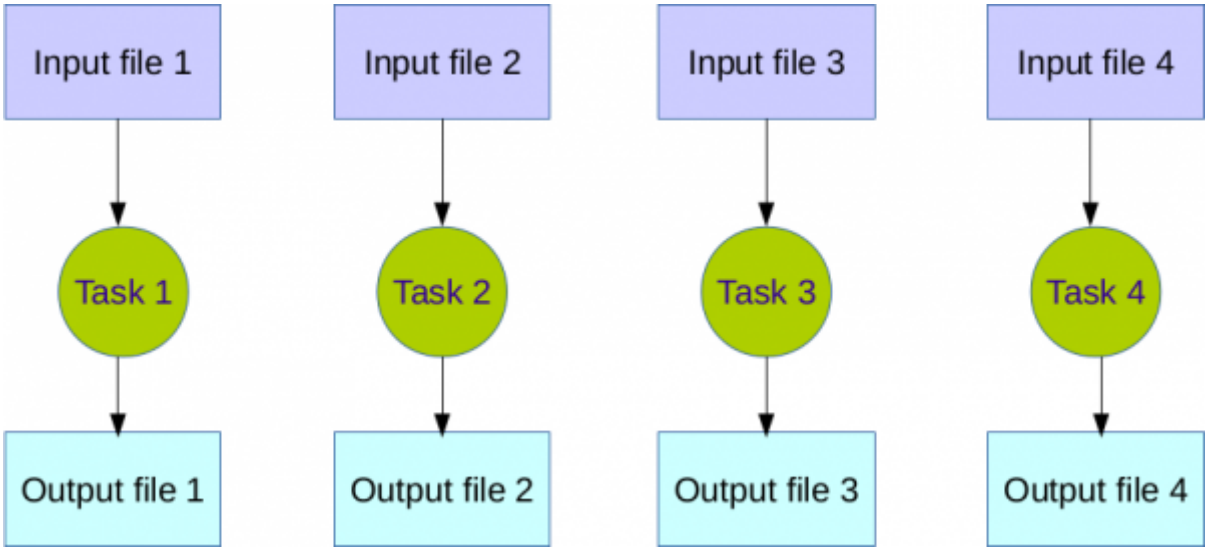


Tableau de tâches

Problème : lancer plusieurs instances du même programme en parallèle sur des données différentes

Exemple applicatif : lancer l'application `appli` 100 fois en parallèle avec des données différentes : `input1`,… `input100`. L'application produira des données `output1`, …, `output100`

Solution : utilisation de tableau tâches (Technique SGE)



Syntaxe :

```
#$ -t 1-100
```

On peut fixer le nombre de tâches actives à un instant donné avec l'option

```
#$ -tc
```

Exemple 100 tâches à exécuter avec un pool de 50 :

```
#$ -t 1-100 -tc 50
```

Script pour l'exemple applicatif :

script_array.sge

```
#!/bin/bash
#$ -q all.q
#$ -N test_sge
#$ -t 1-100
#$ -o $JOB_NAME.$JOB_ID.out
#$ -o $JOB_NAME.$JOB_ID.err
#$ -l h_vmem=4G

./appli input$SGE_TASK_ID output$SGE_TASK_ID
```

Application parallèle openMP (mémoire partagée)

Contexte : l'application s'exécute sur un nœud en utilisant plusieurs cœurs

Solution : utilisation l'environnement parallèle openmp

```
#$ -pe openmp <nbSlots>
export OMP_NUM_THREADS=$NSLOTS
```

Exemple

script_openmp.sge

```
#!/bin/bash
#$ -N test_sge
#$ -pe openmp 8  ## on demande 8 coeurs
#$ -o $JOB_NAME.$JOB_ID.out

export OMP_NUM_THREADS=$NSLOTS

## lancement de l'application

./appli input
```

Application parallèle MPI (mémoire distribuée)

- **Contexte** : on demande plusieurs coeurs sur plusieurs noeuds
- Utilisation de l'environnement parallèle `mpi`
- Utilisation de la file d'attente `parallel.q`

Exemple d'un script SGE MPI

script_mpi.sge

```
#!/bin/bash
#$ -q parallel.q
#$ -pe mpi 16
#$ -N test_sge
#$ -o $JOB_NAME.$JOB_ID.out
#$ -e $JOB_NAME.$JOB_ID.err

#### on charge le module open mpi

module load mpi/openmpi/icc/1.7.5

## lancement de l'application
mpirun -np $NSLOTS ./appli_mpi
```

Suivi des jobs

Notification par Mail

SGE peut notifier l'utilisateur lorsque le job est démarré, arrêté ou supprimé. Pour cela il faut rajouter certaines options dans vos scripts :

1. Spécifier l'adresse Mail de réception des messages SGE `#$ -M addresse@mail`
2. Spécifier le type de notification :
   - recevoir un mail en fin d'exécution du job `#$ -m e`
   - recevoir un mail en début d'exécution du job `#$ -m b`
   - recevoir un mail lorsque le job est interrompu `#$- m a`

Il est possible de combiner les types de notifications : `#$ -m bea`

Exemple

```
#!/bin/bash

#$ -N test_sge
#$ -o $JOB_NAME.$JOB_ID.out

#$ -M kamel.mazouzi@univ-fcomte.fr
#$ -m bea  ##notification : debut, arret, fin

sleep 60
```

Quelques commandes utiles

Pour lister les jobs

```
$ qstat
```

Afficher la liste de tous les jobs

```
$ qstat -u "*"
```

Affichier la liste des noeuds exécutant mes jobs

```
$ qstat -g t
```

Afficher l'état des instances des queues

```
$ qstat -f
```

Afficher l'utilisation des queues

```
$ qstat -g c

CLUSTER QUEUE                   CQLOAD   USED    RES  AVAIL  TOTAL aoACDS
cdsuE
--------------------------------------------------------------------------------
----
3dvisu.q                          0.01                8      8
all.q                             0.00              508    508
parallel.q                        0.00              704    704
tesla.q                           0.00                8      8
xphi.q                            0.01               12     12
```

Afficher les jobs en **attente** pour l'utilisateur toto

```
$ qstat -u toto -s p
```

Afficher les jobs en **exécution** pour l'utilisateur toto

```
$ qstat -u toto -s r
```

Afficher l'état d'un job en erreur (la ligne error peut indiquer la raison de non soumission si le job est en mode **Eqw**)

```
$ qstat -j 66666
```

Savoir pourquoi un job n'a pas ete schedule

```
$ qstat -j job_id
```

Afficher les jobs terminés (historique)

```
$ qstat -s z
```

Afficher les ressources disponibles

```
$ qhost
```

Afficher les ressources disponibles en détail (mémoire, swap, jobs, ...)

```
$ qhost -q
```

Suppression de jobs

Supprimer un job

```
$ qdel 666 ## supprime le job 666
```

Supprimer tous les jobs de toto

```
$ qdel -u toto
```

Forcer la suppression du job numéro 66666

```
$ qdel -f 66666
```

Gestion des jobs

Pour basculer les jobs d'un utilisateur de la file courante vers une nouvelle file (newqueue)

Exemple : basculer les jobs de l'utilisateur `dupond` de la file `all.q` vers la file `bigmem.q`

```
$ qalter -u dupond -q bigmem.q
```

Pour suspendre un job :

```
$ qhold jobid
```

Pour reprendre le traitement d'un job :

```
$ qrls jobid
```

États possible d'un job

Après la soumission d'un job, ce dernier peut avoir plusieurs états :

d	deleting
t , r	transfering, running
s , S , T	suspending , threshold
R	restarted
w	waiting
h	hold
E	error
q	queuing

Gestion de la mémoire

L'utilisation de l'option **h_vmem** est vivement recommandée pour spécifier la mémoire nécessaire à l'exécution des jobs. Cette pratique permet à votre application de disposer des ressources appropriées (cœurs/mémoire) et d'éviter des crashs liés à l'utilisation de ressources excessives sur le cluster (saturation d'un ou plusieurs nœuds de calcul).

Il est par conséquent important de préciser la quantité mémoire à prévoir dans vos scripts :

```
#$-l h_vmem=MEM_MAX
```

MEM_MAX	Quantité de mémoire maximale autorisée par cœur/processus

ATTENTION : Votre job sera tué par SGE s'il dépasse cette quantité mémoire même d'un seul octet
Si aucune valeur n'est précisée pour h_vmem dans vos scripts, **une valeur de 1G lui sera attribuée par défaut**.

Nous vous invitons a bien spécifier la valeur h_vmem en évitant **la surestimation** pour ne pas bloquer les autres utilisateurs.
Pour savoir la mémoire réellement utilisée par votre application, utiliser les commandes suivantes : `qmem, qmemview` ou sur le portail https://mesoportail.univ-fcomte.fr rubrique `mesjobs`

Exemple d'utilisation

```
qsub -l h_vmem=2G mon_script.sge
```

```
qsub -l h_vmem=1800M mon_script.sge
```

Mon job a besoin de 1800 Mo et il sera tué par SGE s'il dépasse cette valeur.

```
qsub -l h_vmem=400M mon_script.sge
```

Vous pouvez spécifier une valeur inférieur à la valeur par défaut (2G)

Applications Séquentielles

L'application séquentielle s'exécute sur un 1 seul cœur, une valeur h_vmem lui sera attribuée :

```
#$ -l h_vmem=300M
```

ou

```
qsub -l h_vmem=300M script.sge
```

Applications MPI

La quantité mémoire demandée est par **processus** ou par **cœur**. Si une application parallèle s'exécutant sur : `NOMBRE_DE_COEURS` coeurs et a besoin d'une mémoire : `MEMOIRE_TOTALE`, il faut diviser cette quantité sur l'ensemble de cœeurs :

```
#$ -l h_vmem= MEMOIRE_TOTALE/NOMBRE_DE_COEURS
```

Exemple

L'application MPI à besoin en total de 24000 M

```
qsub -pe impi_tight=16 -h v_mem=1500M script.sge
```

Applications OpenMP

Dans le cas ou tous les cœurs sont utilisés, il vaut mieux prendre la totalité de la mémoire sur un noeud.

Exemple

Sur une machine de 12G mémoire :

```
qsub -pe openmp 8 -l h_vmem=1.5G script.sge
```

Mode interactif

Par exemple :

```
$ qlogin -q tesla.q -l h_vmem=2G
```

Comment connaître les valeurs disponibles ''h_vmem'' sur les noeuds de calcul ?

```
qhost -F h_vmem
```

```
qmemview
```

Résumé

- La quantité mémoire à spécifier est par **coeur physique**.
- La valeur par défaut est de **1.5G/coeur**.
- Vous pouvez ajuster cette valeur selon vos besoins : vous pouvez demander **plus** ou **moins**.
- Évitez la surestimation, utilisez les commandes : qmem, qmemAll, qmemview pour visualiser la mémoire réellement utilisée.
- En cas de doute, contactez les administrateurs.

Visualisation de la mémoire utilisée par différents jobs, la commande : qmemAll

JOB ID	SLOTS	VMEM REQUESTED	TOTAL VMEM USED	RECOMMENDED VMEM VALUE	RATIO USED
157263	110	1100M	63.159G	705.331M	53.434%
157264	60	1000M	19.821G	405.504M	33.792%
157353	32	2.7G	53.173G	1.993G	61.518%

JOB ID	SLOTS	VMEM REQUESTED	TOTAL VMEM USED	RECOMMENDED VMEM VALUE	RATIO USED
157385	24	1.5G	23.591G	1206.681M	65.466%
157407	64	2.0G	70.971G	1.329G	55.400%
157409	60	1000M	21.235G	433.766M	36.147%
157462	68	1000M	23.149G	417.792M	34.816%
158141	8	1.5G	545.871M	81.879M	4.442%

Analyse

- La première chose a voir ici est la colonne **RECOMMENDED VMEM VALUE**. En fait, l'outil vous suggère des valeurs approximatives pour vos applications.
- On remarque que la plupart des applications **n'utilisent pas toute la mémoire demandée !**
- Le job **158141** utilise la mémoire par défaut attribuée par SGE (1.5G) . Alors que l'utilisateur pouvait donner une valeur inférieur à 1G. Part exemple `100M`

Choisir le type du processeur à utiliser

Certains utilisateurs étudient les performances des applications et le temps 'exécution est important dans ce cas. Attention, le cluster n'est pas homogène, il contient 3 types de processeurs (westmere, sandybridge, haswell).

processeur	date d'achat	vitesse (GHz)	puissance (GFlops)/node	Nombre de cœurs	nombre de nœuds	remarque
Westmere	2010	2.66GHz	255	12	x	utiliser dans la file d'attente `all.q`
Sandybridge	2013	2.2GHz	281	16	x	utiliser dans la file d'attente `all.q` et parallel.q
Haswell	2015	2.6GHz	665	16	20	utiliser dans la file d'attente parallel.q

Pour demander à utiliser un type de processeur particulier ;

```
#$ -l proctype=XXXX
```

Avec XXXX=`westmere`, `sandybridge` ou `haswell`.

Ainsi, pour s'exécuter sur le processeur `sandybridge`, l'option SGE est :

```
#$ -l proctype=sandybridge
```

Le processeur Haswell est disponible uniquement pour les applications MPI de la file `parallel.q`

Comment connaître le type de processeur sur un noeud ? La commande `proctype` permet d'afficher le type d'architecture du processeur :

```
$ [user@node1-32 ~]$ proctype
    -march= westmere
```