

Utilisation de Moyens de Calculs *avec Slurm*

<https://www.math.u-bordeaux.fr/~lfacq/MoyensDeCalcul.pdf>

Durée 3h

2 pauses de 5min

H+1h

H+2h

Questions à la volée

Introduction

- Premier réflexe pour faire des calculs :
utiliser son poste de travail / ordinateur portable
- Mais rapidement bloquant quand les besoins augmentent...

- **Quelles sont les autres solutions disponibles ?**

Moyens disponibles par complexité/puissance croissante

1. *Poste de travail / Portable*
2. *Serveurs de laboratoire*
ex : *IMB - Serveurs (servisu, bureau, compute-1To)*
fonctionnent à l'identique des postes individuels - en dépannage ponctuel
3. => **PlaFRIM3** – Plateforme Fédérative de Recherche en Informatique et Mathématique
Périmètre : les 3 laboratoires : IMB+INRIA+LaBRI
4. => **MCIA** – Mésocentre de Calcul Intensif Aquitain – machine **Curta (Bx)**
Périmètre : toutes les structures universitaires de Nouvelle Aquitaine
+DOREMI : Cali3 (Limoges/H100) + Pyren (Pau) + Jarvis (NUMA/Poitiers)
5. *National (GENCI, Grille Européenne) – sur projet/Dossier*
6. *Européen... – sur projet/Dossier*

Ouvrir un compte

Mot clefs : « imb calcul développement »

<https://www.math.u-bordeaux.fr/imb/cellule/rubrique47.html>

- MCIA

- https://redmine.mcia.fr/projects/cluster-curta/wiki/Comptes_Utilisateurs
- Se connecter : `ssh curta.mcia.fr`

- PlaFRIM3

- <https://www.plafrim.fr/fr/connexion/inscription/>
- Se connecter : `ssh ssh.plafrim.fr` (⚠ nécessite un `.ssh/config` particulier)

MCIA & PlaFRIM3 : 2 plateformes au fonctionnement similaire

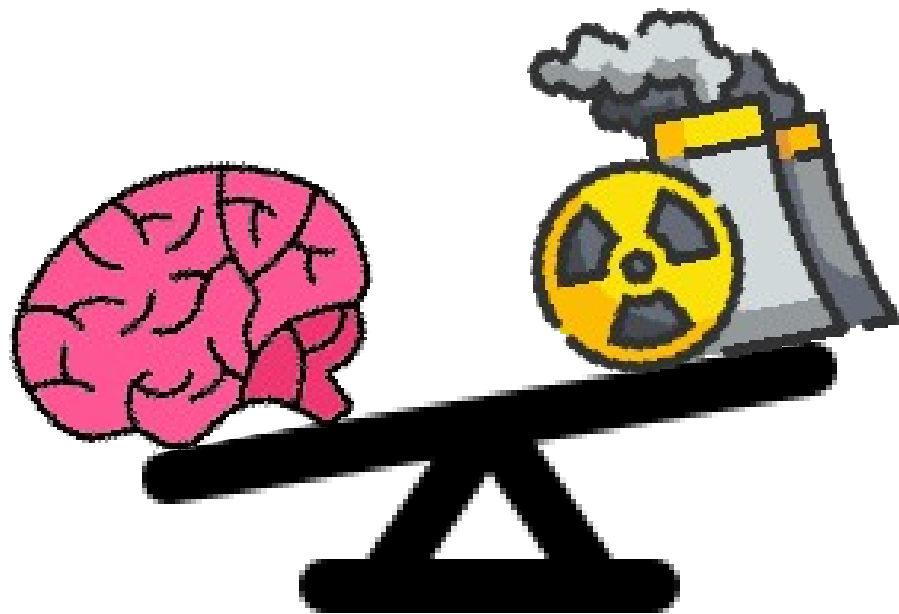
- Ce sont des **Clusters de calcul**
 - = un grand nombre de machines identiques reliées par un réseau rapide
- Plusieurs centaines d'utilisateurs => file d'attente / « chacun son tour »
- => notion de **job** : on doit décrire ses besoins / ses calculs
 - besoin en ressources (combien de machines/quel(s) type(s), cpu, quantité mémoire, ...)
 - description des calculs (script à lancer)
- => notion de **files d'attente** : on y soumet son job
 - puis on attend qu'il s'exécute quand suffisamment de ressources sont libres
- Même gestionnaire de job « **slurm** » sur les deux plateformes MCIA et PlaFRIM3 mais avec un **paramétrage différent**



avertissement !




- l'utilisation des clusters de calculs n'est pas simple... (surtout si on ne pratique pas souvent)
- ... et leur mode d'utilisation peut évoluer avec le temps
- venez demander de l'aide au bureau IMB/MCD 270 ou à votre ingénieur calcul de proximité
- venez faire « contrôler » vos scripts
 - pour partir sur de bonnes bases dès le début en faisant les choix optimaux
 - avant de lancer de grandes campagnes de calcul

⚠ Sobriété ! ⚠



Cogiter vs Calculer ...

Doc & Assistance

- De proximité :
 - Web IMB – les spécificités pour les personnels IMB
<https://www.math.u-bordeaux.fr/imb/cellule/moyens-de-calcul-et-de-stockage>
 - IMB - bureau MCD 270
 - <https://mattemost.math.cnrs.fr> ...
- Officielle MCIA :
 - Doc : <https://redmine.mcia.fr/projects/cluster-curta/wiki>
 - Help :  <https://redmine.mcia.fr/projects/cluster-curta/issues/new>
 - Chat/Element :  <https://enigma.mcia.fr>
- Officielle PlaFRIM3 :
 - Doc : <https://www.plafrim.fr>
 - Help : plafrim-support@inria.fr
 - Chat/Mattemost :  <https://mattermost.inria.fr/plafrim-users/channels/town-square>



Limitations PlaFRIM3

- Par défaut :
 - **Pas d'accès internet en sortie de PlaFRIM3**
 - sauf gitlab académiques, github, sites de référence...
 - => Problèmes pour les « installeurs automatiques »
 - Solutions :
 - Télécharger depuis une autre machine puis recopier sur PlaFRIM3 (**scp**)
 - ou Demande d'ouverture à envoyer à : plafrim-support@inria.fr
- Transferts entre MCIA et PlaFRIM
 - **scp** à initier obligatoirement depuis MCIA (pour envoyer ou récupérer)

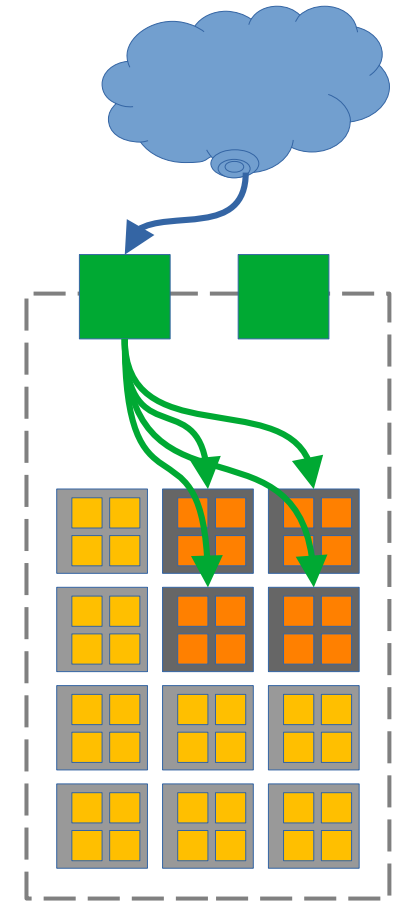
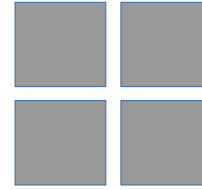


Protection contre les attaques

- Des tentatives répétées de connexions SSH infructueuses entraînent le **blocage automatique** de l'adresse (IP) de la machine source.
 - Le déblocage se fait automatiquement au bout « d'un certain temps »
 - Particulièrement sensible depuis un Wifi / EduROAM (Adresse IP communes à beaucoup de personnes)

Terminologie 1/2

- **Machine = serveur = nœud** [de calcul] = *[compute] node*
- **Cluster = groupe de nœuds identiques**
- **Frontal / Frontaux** = serveurs d'accès (tous les usagers connectés dessus)
 - Machines «**login**» sur MCIA et «**devel**» sur PlaFRIM3
 - Machines similaires, ok pour petits tests / **Ne pas lancer de gros calculs dessus ! (MCIA : limitations des ressources => programmes tués automatiquement)**
- **Socket=connecteur physique pour processeur / Boîtier CPUs** → **Cache(L3)**
- **Processeur / CPU / Cœur / core** : unité de traitement / exécution
- **Tâche / task** = processus unix, composé d'un ou plusieurs :
 - **Thread** = fil d'exécution interne au processus, partagent la même mémoire
- **Job** = réservation de ressources + lancements de tâches dedans
- **GPU = Graphics Processing Unit** (Traitement image, IA, Deep Learning, ...)



Terminologie 2/2

- Mode **interactif** – *mode classique hors HPC*
 - Vous lancez votre programme et il s'exécute sous vos yeux
 - Vous voyez les sorties (output)
 - Vous pouvez répondre à des questions (input)
- Mode **batch** – *spécifique au HPC*
 - Le programme s'exécute sans vous
 - Il peut tourner à tout moment – la nuit par exemple

HPC = High Performance Computing

vue d'ensemble

MCIA vs PlaFRIM3 ?

MCIA-Curta « régionale »

~11.000 cœurs

- Orienté : **production** = obtention de résultats (peu importe le temps), stabilité
- Un cluster principal - homogénéité
 - ~360 nœuds 96Go / 32 cœurs (dont **28 nœuds réservés IMB**)(~20 I2M)
 - 4 nœuds 3To « bigmem »
 - 4 nœuds 192Go + GPU (2 x p100)
 - 4 nœuds 192Go de visualisation (P4000)
- MCIA ~3 fois plus gros que PlaFRIM

PlaFRIM3 « 3 labos »

~4.000 cœurs

- Orienté : **développement expérimentation**, plus « réactive » versions récentes, architectures variées/exotiques, ... plus instable
- Plusieurs clusters différents
 - ~40 bora **192Go/36** cœurs (intel)
 - ~20 sirocco (intel) GPU k40m, p100, v100, rtx8000, a100
 - 22 suroit **256Go/48** coeurs (amd)
 - 1 nœud de visu
 - ... (KnightsLanding, AMD EPYC, ARM)

Slurm

- Un **Gestionnaire de jobs**
 - Démarre (et termine!) les *jobs* sur les nœuds
- Un **Scheduler** (*ordonnanceur*)
 - ordonnancement des jobs pour exploiter le cluster et satisfaire au mieux tous les utilisateurs
- => Politique de **scheduling** (à ce jour)
 - MCIA :
 - *FairShare* + FIFO: ajustement de la priorité pour répartir le temps entre les utilisateurs
 - PlaFRIM3 :
 - *FairShare homéopathique* + FIFO : en pratique FIFO = premier arrivé, premier servi !

FIFO = **F**irst **I**n **F**irst **O**ut

3 commandes pour soumettre un job :

srun, salloc, sbatch

s... comme slurm

srun : réserve des ressources et y démarre un job interactif

- **srun [options] commande arguments**
 - Ex : **srun mon_programme ...**
 - Soumet une demande de réservation-exécution en une seule ligne
 - Mode **Interactif**
 - Les ressources sont libérées à la fin de l'exécution du programme

Usage : tests, exécutions rapides ou interactives, session interactive sur un ou plusieurs nœuds de calcul (compilations/exécutions). *Attente potentielle à chaque lancement*

salloc: réserve des ressources pour une session interactive + srun

- `salloc [options]`
 - Soumet une demande de session (shell) avec des ressources réservées
 - Mode **Interactif**
 - On démarre ensuite, dans cette session, des programmes avec `srun` (ou `mpirun`)
 - Les ressources sont libérées quand la session `salloc` se termine (`exit` ou temps écoulé)

Canevas de session :

```
$> salloc [options]
$> srun [opt1] prog1 arg1
$> srun [opt2] prog2 arg2
$> exit
```



Sans `srun` les commandes s'exécutent sur la frontale

Usage : session de travail interactive de plusieurs heures, plusieurs programmes à tester.
Un seul temps d'attente pour la session entière.

sbatch: réserve des ressources utilisées en batch + **srun**

- **sbatch** [options] script_batch

Canevas de script_batch :

- Soumet une demande de ressource pour une utilisation en **mode batch**
- Le script contient des lancements de programmes avec **srun** (ou **mpirun**)
- Les ressources sont libérées quand le script se termine (fin naturelle ou temps écoulé)

```
#!/bin/bash
#SBATCH [options]
#SBATCH [options]
srun [opt1] prog1 arg1
srun [opt2] prog2 arg2
```



Sans **srun** les commandes s'exécutent sur le premier noeud

Usage : Campagne de calculs. Possibilité de tester/valider en amont avec **salloc**

Commandes Slurm – recap

lancer un job - 3 schémas types

	description	mode	ressources	Options	« run où » ?
srun	<i>all in one</i> réservation + exécution	interactif	<i>one shot</i>	CLI	nœuds réservés
salloc + srun	sessions avec ressources réservées	interactif	session	CLI	frontale puis srun...
sbatch + srun	batch avec ressources réservées	batch	session	CLI+script	1er nœud puis srun...

CLI : Commande Line Interface
= Ligne de Commande

Les principales options de Slurm

Options de Slurm ou de **srun** ?

- Les commandes Slurm de soumission de jobs (**srun**, **sbatch**, **salloc**) ont « vu de loin » les mêmes options
 - La plus part des options sont effectivement génériques
 - Quelques options sont spécifiques (à **srun** notamment)

Options de réservation de ressources

- 2 aspects selon les options :
 - 1) Sélection des nœuds de calculs
 - Trouver des nœuds répondants aux critères demandés (*mémoire, type processeur, temps, carte GPU, ...*)
 - Ex : **--cores-per-socket** (nb cœurs par chip)
 - 2) Conditions d'exécution
 - Comment utiliser les ressources sélectionnées
 - i.e. Comment répartir les processus à lancer

Options Slurm principales :

Nb Nœuds/Cœurs/Taches

- **-n** : nombre total de **tâches** à lancer (défaut : 1 sur chaque nœud) (**--ntasks**)
sur l'ensemble des nœuds du futur job
 - Autre possibilités :
--tasks-per-node --ntasks-per-core -tasks-per-socket
- **-c** : nombre de **cœurs** par tache (défaut : 1) (**--cpus-per-tasks**)
 - Mode multi-thread. Sert à sélectionner les nœuds et à contrôler l'exécution
Rq : « -c » * « -n » = nombre de cœurs demandés au total
- **-N** : nombre de **nœuds** demandés (défaut : 1) (**--nodes**)
 - Ex : **-N10** : 10 nœuds demandés
 - *Range* : **-Nmin-max**
 - **-N2-6** : au moins 2 nœuds et jusqu'à 6 si possible. Le job démarre dès 2 nœuds disponibles

Options Slurm - Remarque/syntaxe

- Généralement, ces syntaxes suivantes sont équivalentes :
 - **c10**
 - **c 10**
 - **c=10**
 - **-cpus-per-tasks=10**
 - **-cpus-per-tasks 10**
- Sans blanc si l'argument est optionnel (ex : **-I**)

Options Slurm courantes :

Durées

- **-t=J-H:M:S** : durée demandée ou *walltime* (`--time`)
(défaut 1h) *walltime* = temps écoulé (horloge, *clocktime*)
(à ne pas confondre avec ~~*cputime* = *walltime* * nb cœurs~~)
ou aussi **J-H**, **J-H:M**, **H:M:S**, **M:S**, **M**

temps dépassé => job détruit !

Exemples :

- **-t 1 ...** : job d'une minute max
- **--time 1-2** : 1j et 2h
- **-t3:4** : 3min 4sec
- **--time=0:0:30** : 30 sec

en réalité : arrondi à la minute ?

Options Slurm « courantes » :

Choix de la file d'attente (partition)

- **-p partition** : demande à ce que le job soit placé dans cette *partition*, ou file d'attente, qui a certaines caractéristiques comme : *temps maximum, nombre de nœud maximum, nb max job par utilisateur ...*
- une même partition peut contenir différents types de nœud : dans ce cas, **-C** permet de filtrer sur le type de nœuds
- si on ne précise rien, c'est la partition par défaut qui est utilisée

Options Slurm « courantes » :

Contraintes sur le choix des nœuds

- **-C type** : demande un certain type de nœud - *features constraints* (**--constraint**)
- Les *features* disponibles dépendent de la plateforme : type de nœud, architecture, réseau , ...
(plus de détails dans quelques transparents)
- Pour afficher facilement les *features* existantes, alias à mettre dans votre **\$HOME/.bashrc** :

```
alias sfeatures='sinfo -o "%30N %10c %10m %60f %10G"'
```

Exemple de *features* (ici PlaFRIM3)

NODELIST	CPU	MEMORY	AVAIL_FEATURES
sirocco[14-16]	32	385000	sirocco, intel , skylake, ..., nvidia, tesla, v100
sirocco[07-13]	32	255000	sirocco, intel , broadwell, nvidia, tesla, p100
zonda[01-21]	64	250000	amd , zen2, zonda
bora[001-044]	36	191000	bora, intel , cascadelake, omnipath
brise	96	1030000	brise, intel , broadwell, bigmem
enbata[01-02]	64	380000	amd , zen4, enbata, mellanox, amdgpu, mi210
sirocco[18-20]	40	190000	sirocco, intel , skylake, nvidia, quadro, rtx800
sirocco21	48	512000	sirocco, amd , zen2, nvidia, ampere, a100
sirocco[22-25]	64	512000	sirocco, amd , zen3, nvidia, ampere, a100
suroit[01-22]	48	250000	amd , zen4, suroit, mellanox
arm01	224	260000	arm, cavium, thunderx2
diablo[01-04]	64	255000	amd , zen2, diablo
diablo05	128	1030000	amd , zen2, diablo, bigmem
diablo[06-09]	128	1030000	amd , zen3, diablo, bigmem, mellanox
kona[01-04]	256	94000	kona, intel , knightslanding, knl, omnipath
sirocco17	40	1030000	sirocco, intel , ..., nvidia, v100 , bigmem
souris	96	2990000	souris, sgi, ivybridge, bigmem
suet01	56	250000	suet, intel , icelake, intelgpu , gpuflex , flex170
visu01	20	128000	visu
mistral[02-03,06]	20	128000	mistral

Options spécifiques **srun** courantes

- **-I** : immédiat ou délai max en secondes (**--immediat**)
 - Ex : **srun -I ...** : demande un démarrage immédiate
srun I60 ... : demande un démarrage d'ici
60 sec maximum sinon abandon
*en pratique : mettre au moins 30 à 60
(délais de prise en compte par slurm)*
- **--pty** : pour entrée/sorties interactives
 - utilisation typique : **srun ... --pty bash**
démarré un shell interactif
sur le premier nœud obtenu

Options Slurm plus occasionnelles

- **--time-min=*J-H:M:S*** : durée minimum demandée (même esprit que **-Nmin-max**)
- **--job-name=*NomDuJob* (-J)** : donne un nom au job
- **-o *output_file*** : fichier pour les sorties classiques (défaut : **slurm-JOBID.out**)
- **-e *error_file*** : fichier pour les sorties erreurs (défaut : **slurm-JOBID.err**)

Options Slurm plus occasionnelles

- **--mail-user=mon_email@math.u-bordeaux.fr**
adresse e-mail pour recevoir des notifications
- **--mail-type=BEGIN, END**
pour activer les notifications : ici au début et à la fin du job
autres possibilités : **FAIL, QUEUE, TIME_LIMIT,**
TIME_LIMIT_50,
TIME_LIMIT_80,
TIME_LIMIT_90

ALL est équivalent à **BEGIN, END, FAIL, QUEUE**

srun, salloc, sbatch

...premières utilisations concrètes

Job interactif typique avec **srun**

une session de travail sur un cœur

- MCIA & PlaFRIM3

- **srun -I60 --time=4:0:0 --pty bash**

- si un coeur est disponible dans les 60 secondes, démarre un shell (bash= interpréteur de commandes) sur un cœur pour une durée max de 4 heures

- **srun --time=4:0:0 --pty bash**

- idem mais attend indéfiniment qu'un coeur soit libre...
...peut-être à 3h du matin! (**ctrl-c** pour arrêter l'attente)

Quand le srun (bash) se termine, les ressources sont libérées

TP-0 préparatoire *optionnel*

Si vous avez communiqué votre login MClA :

- Pour utiliser les machines réservées pour la formation, ajoutez l'option « `--reservation=formation_slurm` » ou exécutez, pour la durée de la session :

```
export SLURM_RESERVATION=formation_slurm
```

Sinon, utilisez la partition « `-p preemptible` »

```
export SLURM_PARTITION=preemptible
```

*pour supprimer une variable d'environnement : **unset VARIABLE***

TP-1a srun

hostname : affiche le nom de la machine
uniq -c : compte les lignes identiques consécutives
wc -l : compte les lignes
ctrl-c : «break» la commande en cours

- Tester et interpréter les résultats des commandes suivantes :

```
- srun -t1 -n2 hostname
- srun -t1 -N2 hostname
- srun -t1 -N2-100 hostname | wc -l
- srun -t1 -N2-100 -c10 hostname | wc -l
- srun -t1 -n100 hostname
- srun -t1 -n100 hostname | uniq -c
```

TP-1b srun

hostname : affiche le nom de la machine
uniq -c : compte les lignes identiques consécutives
wc -l : compte les lignes
ctrl-c : «break» la commande en cours

- Tester et interpréter les résultats des commandes suivantes :

```
- srun -c1000 hostname  
- srun -t1 -n1000 -c50 -I30 hostname  
- srun -t1 -N1-300 hostname | wc -l  
- module load formations/toys  
- srun -t1 -n10 who_am_i
```

Job interactif typique avec **salloc**

une session de travail sur un cœur

(MCIA & PlaFRIM3)

1) **salloc -I60 --time=4:0:0** : pour réserver les ressources
réserve un coeur pour 4h si disponible dans les 60 secondes

2) **srun --pty bash** : pour aller sur le nœud
et y exécuter des commandes...

démarre un interpréteur de commande sur le coeur/noeud réservé

3) **srun mon_job_1** : lancer un programme 1

4) **srun mon_job_2** : lancer un programme 2

5) **exit**

note : **salloc** crée un nouvel interpréteur de commandes (*shell*)
via lequel les ressources sont réservées et conservées.

Quitter ce *shell* libère ces ressources.

TP-2 salloc

- Tester et interpréter les résultats des commandes suivantes :

```
- salloc -t5 -N2
  • hostname
  • srun          hostname
  • srun -n2      hostname
  • srun -n3      hostname
  • srun -c2 -n1  hostname
- exit
```

hostname : affiche le nom de la machine

uniq -c : compte les lignes identiques consécutives

wc -l : compte les lignes

ctrl-c : «break» la commande en cours

ctrl-d : «exit» de la session

Fichier Batch (non interactif)

calculs en suivant

- **Un batch** = Script bash avec des directives **#SBATCH** (en début de fichier) contenant des options Slurm

```
==== fichier mon_batch ====
#!/bin/bash
#SBATCH -n3                # on demande 3 tâches (c=1 donc 3 cœurs)
#SBATCH --time=2:0:0      # pour 2 heures
srun mon_calcul_1         #lancement de 3 instances de mon_calcul_1
srun mon_calcul_2         #lancement de 3 instances de mon_calcul_2
=====
```

- Utilisation :

```
$> sbatch mon_batch      # soumet le batch dans la file d'attente
```

- Les options en ligne de commande sont **prioritaires** sur les directives **#SBATCH** du script :
\$> **sbatch -n4** mon_batch # idem sur 4 tâches au lieu de 3

Fichier Batch (non interactif)

calculs simultanés

```
==== fichier mon_batch ====
#! /bin/bash
#SBATCH -n4 --time=2:0:0    # on demande 4 cœurs pour 2h...

srun -n2 mon_calcul_1 & #lancement de 2 instances de mon_calcul_1
srun -n2 mon_calcul_2 & #lancement de 2 instances de mon_calcul_2

wait          # attend la terminaison des 2 calculs
=====
```

Utilisation :

```
sbatch mon_batch    # soumet le batch dans la file d'attente
```

Fichier Batch

```
==== fichier mon_batch ====  
#!/bin/bash
```

hostname



```
#OK  
#SBATCH -n3 # directive ignorée  
#SBATCH --time=2:0:0 # directive ignorée  
srun mon_calcul_1 #lancement de 1 instance de mon_calcul_1  
srun mon_calcul_2 #lancement de 1 instance de mon_calcul_2  
=====
```

- Pas de commande entre **#!...** et **#SBATCH** (ignorées sinon)



- Commentaires et lignes blanches ok
- Pour commenter une directive **#SBATCH** : **##SBATCH**

TP-3 : tester `srun`, `salloc`, `sbatch`

- 1) Lancer, le plus simplement possible, un job sur **3** nœuds pour afficher les noms de ces trois nœuds (commande `hostname`) en demandant le moins de ressources possible
- 2) Démarrer une session `salloc` de **5** minutes pour pouvoir faire la même chose qu'en 1) en tapant juste « `srun hostname` »
- 3) Utiliser `sbatch` pour pouvoir faire la même chose qu'en 1) en faisant juste « `sbatch mon_batch` »

*Pour éditer des fichiers : **nano**, **emacs**, **vi**, **vim**
... ou **scp** ou **sshfs** depuis votre poste !*

TP-3 solutions

1) `srun -t0:0:1 -N3 hostname`

→ 1 seconde* - 3 nœuds - 3 cœurs

2) `salloc -t5 -N3`

→ 5 minutes - 3 nœuds - 3 cœurs

`srun hostname`

`exit`

3) Fichier `mon_batch` :

`$> sbatch mon_batch`

`#!/bin/bash`

`$> ls -alrt`

`#SBATCH -t0:0:1 -N3`

`$> cat slurm-XXXXX.out`

`srun hostname`

* : en réalité, slurm arrondi à la minute supérieure

TP-4

- Faire :
`module load formations/toys`
- Puis lancer en batch 2 instances de « `cpu_wait 30` », sur 2 nœuds, pendant 40 secondes
- Pendant ce temps, utiliser à répétition `squeue` pour suivre l'état de votre job
- Observer les fichiers résultats

TP-4 solution

```
$> module load formations/toys
$> cat > batch_30
#!/bin/bash
#SBATCH -N2 -t0:0:40
srun cpu_wait 30
(ctrl-d)
```

```
$> sbatch batch_30
```

```
$> squeue -u $USER
```

```
$> ls -alrt
```

```
$> cat slurm-XXXXX.out      # le nom exacte est affiché par « ls -alrt » ci-
dessus
```

TP-4bis

- Faire :
`module load formations/toys`
- Puis lancer en batch, sur 2 nœuds, un total de 2 instances de « `cpu_wait 180` » pendant 60 secondes
- Pendant ce temps, utiliser à répétition `squeue` pour suivre l'état de votre job
- Observer les fichiers résultats

TP-4bis solution

```
$> module load formations/toys
```

```
$> cat > batch_180
```

```
#!/bin/bash
```

```
#SBATCH -N2 -t0:0:60
```

```
srun cpu_wait 180
```

```
(ctrl-d)
```

```
$> sbatch batch_180
```

```
$> squeue -u $USER
```

```
$> ls -alrt
```

```
>$ cat slurm-3538756.out
```

```
slurmstepd: error: *** JOB 3538756 ON n355 CANCELLED AT 2020-10-26T16:14:43 DUE TO TIME LIMIT ***
```

```
srun: Job step aborted: Waiting up to 62 seconds for job step to finish.
```

```
slurmstepd: error: *** STEP 3538756.0 ON n355 CANCELLED AT 2020-10-26T16:14:43 DUE TO TIME LIMIT ***
```

Le job a été tué car il a dépassé le temps imparti

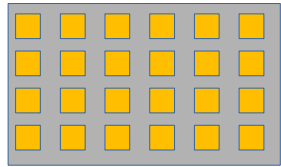
PlaFRIM3 spécifique : Options Slurm


- **--exclusive** : pour demander les nœuds en mode exclusif (personne d'autre que vous)
 - *Permet d'éviter les perturbations dues aux autres utilisateurs qui pourraient arriver sur la même machine*
(saturation mémoire, perturbation des temps de calcul et des accès disque et réseau)
 - *Mais, monopolise plus de ressources, => temps d'attente plus long, ...*
- ⚠ – Ne dispense pas de demander les cœurs/tâches dont on a besoin (1 par défaut !)
- Combiné avec **-c** (nb cœurs par tâche) : le **-n** (nb tâches) sera choisi automatiquement pour occuper au maximum tous les cœurs de tous les nœuds réservés. Ex : « **-c1** » pour lancer le maximum de tâches (de 1 coeur)

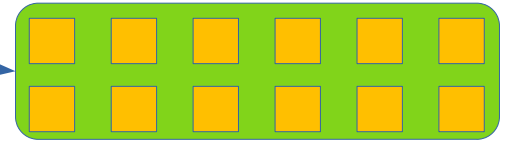
--exclusive (PlaFRIM) + range

- Exemple : comment lancer le maximum de tâches de 12 cœurs sur 3 miriels ?

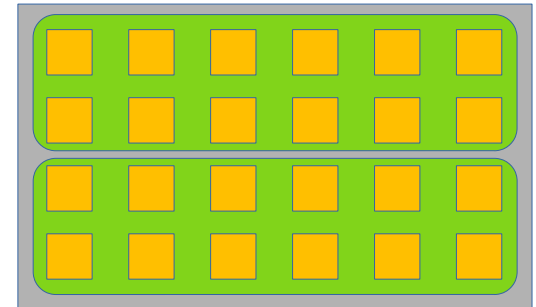
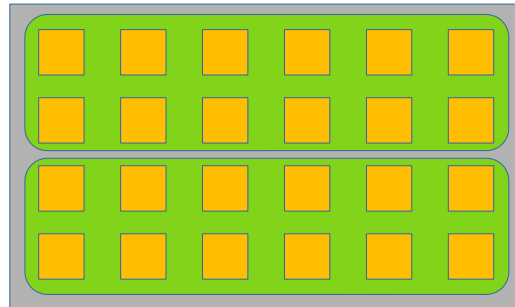
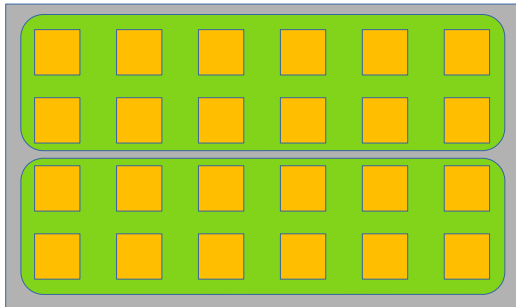
miriel



- les miriels ont 24 cœurs 
- 24 cœurs / 12 par tâche = 2 tâches par nœud
soit pour 3 nœuds : 3 nœuds * 2 tâches par nœud = 6 tâches au total



```
srun --exclusive -C miriel -N3 -c12 mon_prog  
miriel001          miriel002          miriel003
```



Très pratique
pour les
« range »



```
srun --exclusive -C miriel -N3-10 -c12 mon_prog
```

PlaFRIM3 spécifique : Options Slurm

- **-C type** : demande un certain type de nœud
features constraints (--constraint)
 - Type de cluster : **-C bora** , **-C suroit** , **-C zonda** , ...
 - Nœud GPU : **-C p100** , **-C v100** , **-C a100** , ...
 - Architecture CPU : **-C intel** , **-C amd** , **-C arm** , ...
- Pour afficher facilement les *features* existantes, alias à mettre dans votre **\$HOME/.bashrc** :

```
alias sfeatures='sinfo -o "%30N %10c %10m %60f %12G"'
```

PlaFRIM3 spécifique : *features*

NODELIST	CPU	MEMORY	AVAIL_FEATURES
sirocco[14-16]	32	385000	sirocco, intel , skylake, ..., nvidia, tesla, v100
sirocco[07-13]	32	255000	sirocco, intel , broadwell, nvidia, tesla, p100
zonda[01-21]	64	250000	amd , zen2, zonda
bora[001-044]	36	191000	bora, intel , cascadelake, omnipath
brise	96	1030000	brise, intel , broadwell, bigmem
enbata[01-02]	64	380000	amd , zen4, enbata, mellanox, amdgpu, mi210
sirocco[18-20]	40	190000	sirocco, intel , skylake, nvidia, quadro, rtx800
sirocco21	48	512000	sirocco, amd , zen2, nvidia, ampere, a100
sirocco[22-25]	64	512000	sirocco, amd , zen3, nvidia, ampere, a100
suroit[01-22]	48	250000	amd , zen4, suroit, mellanox
arm01	224	260000	arm, cavium, thunderx2
diablo[01-04]	64	255000	amd , zen2, diablo
diablo05	128	1030000	amd , zen2, diablo, bigmem
diablo[06-09]	128	1030000	amd , zen3, diablo, bigmem, mellanox
kona[01-04]	256	94000	kona, intel , knightslanding, knl, omnipath
sirocco17	40	1030000	sirocco, intel , ..., nvidia, v100 , bigmem
souris	96	2990000	souris, sgi, ivybridge, bigmem
suet01	56	250000	suet, intel , icelake, intelgpu, gpuflex, flex170
visu01	20	128000	visu
mistral[02-03,06]	20	128000	mistral

MCIA spécifique : Options Slurm

- **-C type** : demande un certain type de nœud - *features constraints*
(**--constraint**)
 - Type de nœud :
 - **-C compute** : nœuds par défaut - 32 cœurs, 96Go de RAM (non **imb**)
 - **-C bigmem** : idem avec 3To de RAM
 - **-C visu** : nœuds avec carte GPU de visualisation (3D)
 - **-C gpu** : (ou **-p gpu**) nœuds avec carte GPU p100
nécessite également l'option **--gres...**
 - **-C imb** : nœuds **compute** mais réservés IMB (ou **-p imb-resources**) (idem **i2m**)
- Pour afficher facilement les *features* existantes, alias à mettre dans votre **\$HOME/.bashrc** :

```
alias sfeatures='sinfo -o "%30N %10c %10m %60f %12G"'
```

MCIA spécifique : *features*

NODELIST	CPUS	MEMORY	AVAIL_FEATURES	GRES
n[001-035, 037-236, 243, 246, 253, 316-336]	32	92160	compute	
n[337-364]	32	92160	i2m	
	32	92160	imb	
visu[01-04]	32	192699	visu	gpu:2(S:0-1)
gpu[01-04]	32	192700	gpu	gpu:2(S:0-1)
bigmem[01-04]	64	3005500	bigmem	

MCIA spécifique : Options Slurm pour utiliser les GPU

- **-C gpu --gres=gpu:1** : demande un nœud GPU avec une carte GPU
- **-C gpu --gres=gpu:2** : demande un nœud GPU avec ses 2 cartes GPU
 - ce qui a pour effet de définir les variables d’environnement suivantes (cas avec 2 GPU demandées) :
 - **SLURM_STEP_GPUS=0,1**
 - **CUDA_VISIBLE_DEVICES=0,1**
 - **GPU_DEVICE_ORDINAL=0,1**
- **Warning** : Options purement déclaratives ! pour avoir des garanties il faut réserver tous les cœurs de la machine hôte et utiliser toutes les cartes GPU de la machine (et ne pas gâcher!)



Spécifiques MCIA : Options Slurm

- Pas de « **--exclusive** » disponible (évite le gâchis involontaire)
 - Pour simuler : s'arranger pour « réserver exactement *tous les cœurs* » des machines. Avec des machines **32 cœurs** :
 - 1) Sans préciser **n** (1 par nœud par défaut) : **c** doit être égal à **32** :
srun -c32 -N5 ...
 - 2) En précisant **n** : (et **32** être un multiple de **c** !)
 $(n \times c)$ doit être égal à $(N \times 32)$
nbTaches x cœursParTache == NbNoeuds x 32 (Cœurs)
 - Ex : **srun -n8 -c16 -N4 ...**
 $8 \times 16 = 4 \times 32 = (2 \times 4 \times 16)$
= total de 8 taches de 16 cœurs sur 4 Nœuds
= soit 2 taches de 16 cœurs sur chacun des 4 nœud

Récap Paramètres → Résultats

<i>paramètres</i>				<i>résultats</i>			
<i>-N Node</i>	<i>-c cores /task</i>	<i>-n nb tasks</i>	<i>options</i>	<i>Nodes</i>	<i>core/ task</i>	<i>task</i>	<i>commentaire</i>
		n		? (1 à n)	1	n	n taches de 1 cœur
	c	n		? (1 à ?)	c	n	n taches de c cœur
N				N	1	N	N taches d'1 cœur sur N nœuds
N	c			N	c	N	N taches de c cœurs
N	c		--exclusive (PlaFRIM)	N	c	max	le max de taches possible sur N nœuds entiers
N		n		N	1	n	n taches d'1 core/N nœuds
N	c	n		N	c	n	n taches de c cores sur N nœuds

n toujours \geq N

sinon N ajusté à la baisse

Légende :

Valeur par défaut

Partitions & Jobs « préemptables » *des ressources bonus*

- Principe :
 - il est possible d'aller au-delà des limitations habituelles pour utiliser des ressources libres, mais sans garantie d'exécution
 - les jobs lancés sur ces partitions peuvent être tués à tout moment si un job plus légitime est lancé et a besoin de ces ressources
 - ~60 secondes avant destruction : Ils reçoivent un signal (SIGTERM) de terminaison ce qui permet de sauvegarder l'état courant (*checkpointing*) pour garder le maximum de bénéfices
 - Ils sont alors susceptibles d'être automatiquement relancés une deuxième fois
- MCIA : -p preemptible
- PlaFRIM : -p preempt

Mixage d'options

sa`l`loc/sbatch + srun ?

- **sa`l`loc/sbatch**
choix des paramètres (-N -c -n ...) initiaux
- **srun** hérite par défaut de ces paramètres.
- On peut les redéfinir à conditions de rester dans l'enveloppe des ressources réservée :
N nœuds, et n*c cœurs
 - Les paramètres par défaut sont recalculés
- Conseil : testez pour vérifier que cela fait bien ce que vous espérez
- Erreur classique : oublier les paramètres hérités

Exemple de session salloc / MCI

- `salloc -N2 -c32 -t10` *#(2x32= 64 cœurs alloués sur 2 nœuds, 2 tâches)*

```
$> srun -c2 hostname | wc -l
```

```
32
```

```
$> srun -c4 hostname | wc -l
```

```
16
```

```
$> srun -c20 hostname
```

```
n358
```

```
n357
```

```
$> srun -c16 hostname
```

```
n357
```

```
n357
```

```
n358
```

```
n358
```

Très pratique
pour les
« nodes range »



```
salloc -N2-8 -c32 -t10
```

```
srun -c16 mon_prog #lance jusqu'à 16 tâches
```

Exemple de session `salloc` / MCI

- `salloc -t5 -N2 -n64` → 2 nœuds de 32 cœurs = 64 cœurs, pour 64 tâches
 - `$> srun -c16 -n4 hostname` ⇒ `-c16 -n4 -N2 hérité`
→ ((16x2x2) cœurs = 64 en 32x2x2 ok

n356 / n356 / n355 / n355
... ok, 4 réponses, 2 fois chaque nom de machine
 - `$> srun -c16 hostname` ⇒ `-c16 -N2 -n64 hérité`
srun: error: (64*16 cœurs demandés !! car **n=64**)

salloc -N2 -c32

```

SLURM_NODELIST=n[357-358]
SLURM_JOB_NAME=bash
SLURM_JOB_QOS=normal
SLURM_NNODES=2
SLURM_JOBID=3464698

SLURM_TASKS_PER_NODE=1(x2)
SLURM_CPUS_PER_TASK=32
SLURM_JOB_ID=3464698
SLURM_SUBMIT_DIR=/gpfs/home/lfacq

SLURM_JOB_NODELIST=n[357-358]
SLURM_CLUSTER_NAME=curta
SLURM_JOB_CPUS_PER_NODE=32(x2)
SLURM_SUBMIT_HOST=login02
SLURM_JOB_PARTITION=imb-resources
SLURM_JOB_ACCOUNT=imb
SLURM_JOB_NUM_NODES=2
SLURM_MEM_PER_NODE=9610

```

salloc -N2 -n64

```

SLURM_NODELIST=n[357-358]
SLURM_JOB_NAME=bash
SLURM_JOB_QOS=normal
SLURM_NNODES=2
SLURM_JOBID=3464708
SLURM_NTASKS=64
SLURM_TASKS_PER_NODE=32(x2)

SLURM_JOB_ID=3464708
SLURM_SUBMIT_DIR=/gpfs/home/lfacq
SLURM_NPROCS=64
SLURM_JOB_NODELIST=n[357-358]
SLURM_CLUSTER_NAME=curta
SLURM_JOB_CPUS_PER_NODE=32(x2)
SLURM_SUBMIT_HOST=login02
SLURM_JOB_PARTITION=imb-resources
SLURM_JOB_ACCOUNT=imb
SLURM_JOB_NUM_NODES=2
SLURM_MEM_PER_NODE=96100

```

Options Slurm spécifiques MCIA

- Accéder aux 28 machines (standard) réservées pour l'IMB (idem i2m)
 - *prérequis : appartenir au groupe **imb** (vérifier avec la commande « id »)*
 - **-p imb-resources** ou **-C imb** : cibler uniquement les nœuds IMB
 - **-p imb,imb-resources** : pour cibler tous les nœuds
 - pour systématiser, ajouter dans votre **\$HOME/.bashrc** :
export SLURM_PARTITION=imb,imb-resources
- ~~Particularité des nœuds IMB : **mode boost activé** !~~
 - ~~Fréquence CPU jusqu'à 3,7GHz au lieu de 2,1GHz selon le nombre de cœurs qui travaillent (température / consommation électrique)~~
 - ~~Meilleures performances mais reproductibilité des temps plus compliquée~~

Réserver de la mémoire

- PlaFRIM3 :
 - Ne sert qu'à sélectionner un nœud avec suffisamment de mémoire
 - Actuellement, pas de contrôle des limites mémoires =>
!! les jobs peuvent entrer en conflit pour la mémoire !!
- MCIA : limite molle
 - **par défaut : 1Go par cœur (automatique)** (3Go=mémoire_totale/nb coeurs=96Go/32)
 - option slurm : `--mem=90G` explicite mémoire max par nœud, unités : K,M,G,T
 - `--mem=0` implicite : prend automatiquement le max
à ne faire que si on en a réellement besoin ou nœuds entiers
 - molle : contrôles ponctuels de non dépassement
 - **!! les jobs peuvent entrer en conflit aussi :-)** (mais moins fréquemment)
 - **!! en cas de dépassement, le job est détruit !!**

Réserver de la mémoire

Conclusions :

- dès que l'on veut travailler de manière déterministe ou précise, il faut être en mode « --exclusif » (réel ou simulé) pour avoir des nœuds entiers.
- Au MCIA, au delà de 1Go/cœur, il faut impérativement préciser la mémoire

Commandes Slurm
pour voir l'état des jobs, des nœuds,
annuler un job, voir les priorités...

Visualiser l'état des jobs : **squeue**

- **squeue [-l|--long]**

- affiche tous les jobs en file d'attente
- États du job : *STATE*

* au MCIA : machine qui redémarre (~5min)

Notations :
[] : optionnel
| : choix

Possibilités ici :
squeue
squeue -l
squeue --long

PD=Pending → **CF**(ConfigurinG)* → **R**=Running → **CG**=Completing (fin)

- **squeue -u \$USER [-l|--long]**

- Idem mais n'affiche que vos jobs

- PlaFRIM3 : commande « **sudo squeue ...** »

sudo queue - - long (PlaFRIM3)

JOBID	PARTITION	NAME	USER	STATE	TIME	TIME_LIMIT	NODES
227780	routage	full1	xxxxxx	RUNNING	2-22:48:16	3-00:00:00	1 bora013
229994	routage	15patien	yyyyy	RUNNING	1-00:40:01	2-23:59:00	1 sirocco07
230027	routage	15patien	yyyyy	RUNNING	2-03:46:36	2-23:59:00	1 sirocco16
230947	routage	v12Multi	zzzzzz	RUNNING	2-00:17:04	3-00:00:00	4 bora[001-004]
231219	routage	single30	zzzzzz	RUNNING	1-23:19:03	3-00:00:00	4 bora[029-032]
231674	routage	sunset_t	ttt	RUNNING	1-20:16:32	2-00:00:00	3 sirocco[08-09,12]
231709	routage	NEDD	uuuuu	RUNNING	1-20:10:02	3-00:00:00	1 sirocco12
231676	routage	S_UN	uuuuu	RUNNING	1-20:11:02	3-00:00:00	1 miriel048
231677	routage	S_UN	uuuuu	RUNNING	1-20:11:02	3-00:00:00	1 miriel048

Supprimer des jobs : **scancel**

- **scancel JOBID JOBID ...**
 - Détruit les jobs dont les JOBID sont listés
- **scancel -u \$USER**
 - Détruit TOUS vos jobs

Visualiser l'état des nœuds et les caractéristiques des files : **sinfo**

- **sinfo [-l]**

- affiche les caractéristiques des files d'attentes et l'état des nœuds
- états (STATE) :
 - **Idle**=100 % libre
 - **Mixed**=Partiellement utilisé/disponible (des cœurs à prendre)
 - **Allocated**=100 % réservée
 - **Down**=arrêté
 - **Drain**=non réservable (administration)

Visualiser l'état des nœuds et les caractéristiques des files : **sinfo**

- **sinfo** [-l]

- PARTITION:nom de la partition
- TIMELIMIT : temps max que l'on peut demander
- JOB_SIZE : nb max de nœuds que l'on peut demander
- NODES : nb de nœuds dans cette *partition*
- STATE : état des nœuds (cf slide précédent)
- NODELIST : liste des nœuds dans cet état

sinfo - - long (PlaFRIM3)

PARTITION	AVAIL	TIMELIMIT	JOB_SIZE ...	GROUPS	NODES	STATE	NODELIST
routage*	up	3-00:00:00	1-infinite	all	14	down*	bora028,miriel[008,016-017,019-020,022,024,038,043-044,075,081,083]
routage*	up	3-00:00:00	1-infinite	all	12	mixed	bora013,miriel[006,018,026,045],sirocco[03,07-10,12,16]
routage*	up	3-00:00:00	1-infinite	all	42	allocated	bora[001-004,006,029-036,038-039],miriel[001-004,021,023,048,060,062-064,067-071,073,076,078-079],sirocco[01-02,04-05,11,15,17]
routage*	up	3-00:00:00	1-infinite	all	73	idle	arm01,bora[007-012,014-027,037,040],brise,diablo[01-05],kona[01-04],miriel[005,009-015,025,027-037,039-042,050-053,056-058,084-088],sirocco[13-14],souris,visu01

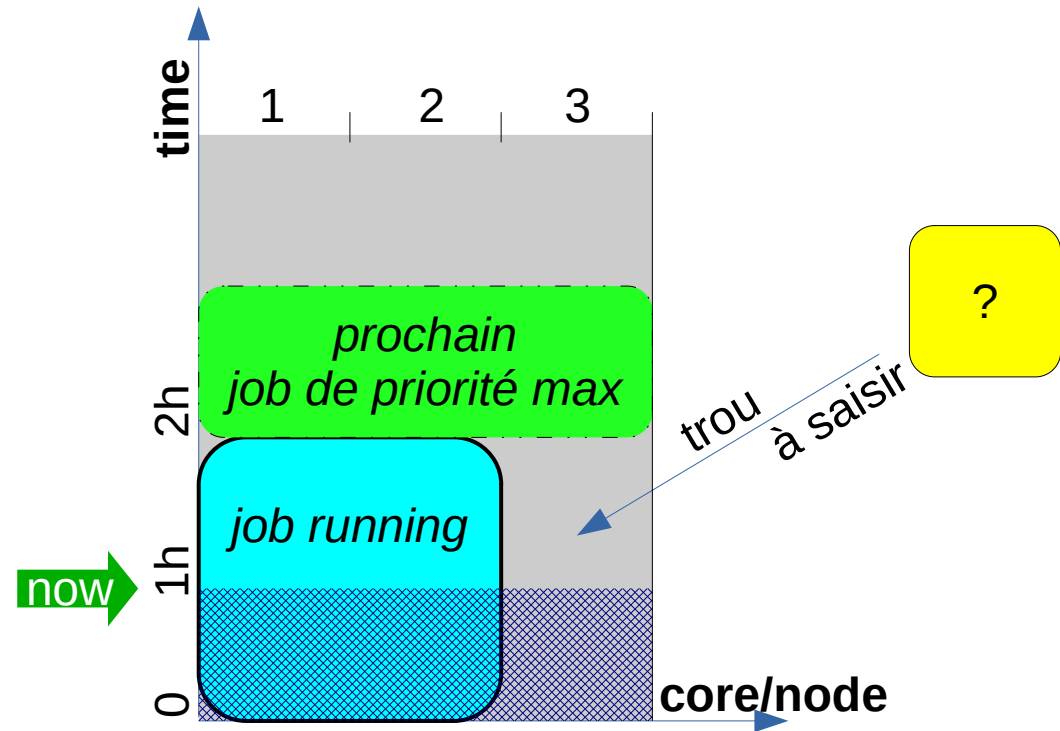
Visualiser les priorités des jobs: **sprio**

- **sprio -l** : priorités
- **sprio -l -n** : priorités normalisées
- **sprio -w** : poids des différents critères

! sur PlaFRIM3 : vous ne voyez que vos jobs !

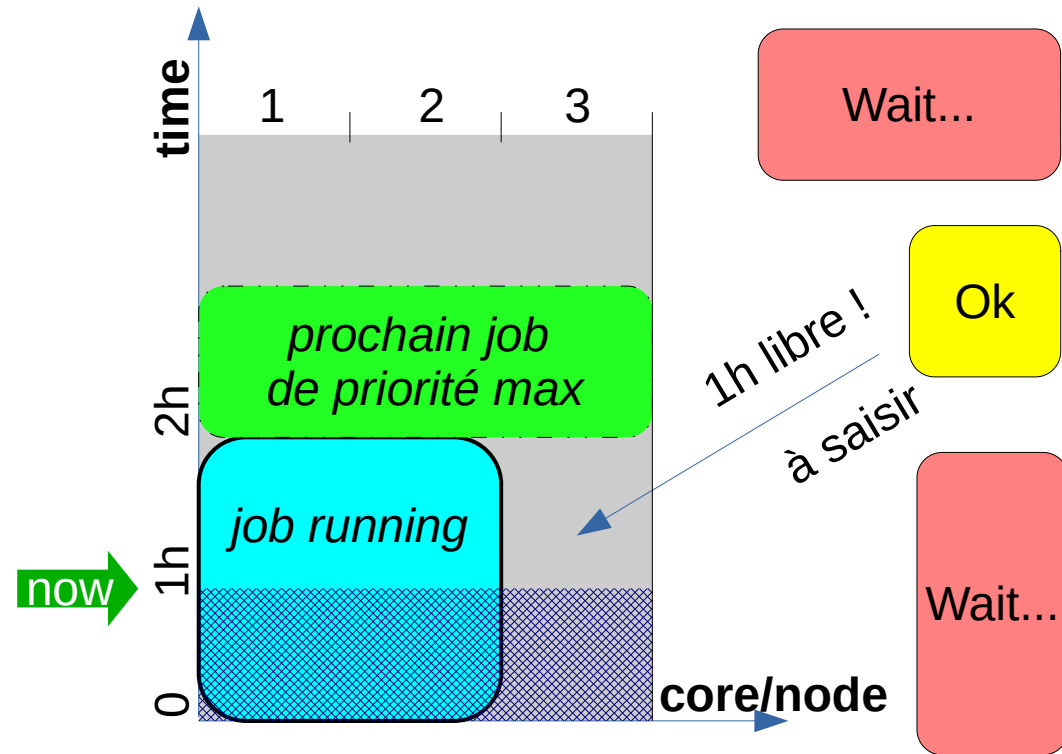
Le *backfill* ou comment faire démarrer plus rapidement vos jobs

- le *scheduler* prévoit d'exécuter les jobs par ordre de priorité
- mais cela crée des trous dans l'ordonnancement
- Il est possible de passer dans ces trous => « *tétris* »



Le *backfill* : comment faire démarrer plus rapidement vos jobs !

- ajuster au mieux son *walltime* et minimiser ses ressources pour profiter des «trous» dans le scheduling par priorités
- trop long ou trop de cœurs/nœuds ou trop de mémoire... demandés => attente...



Le choix du *walltime*

durée des jobs

- De manière générale : les choisir les plus petits possibles... modulo une granularité minimale : 5 à 10min
- Plus le *walltime* demandé est grand, plus vous allez devoir attendre
 - PlaFRIM : max 3j
 - MCIA : les partitions autorisant des jobs très longs fonctionnent au compte goutte.
 - Mode normal : <5j,
 - De 5 à 30j (*longq*) grosse limitation du nombre de jobs en parallèle
 - Dans tous les cas $NbCoeur \times NbHeures < 34.560$

Les modules

Les modules

- **applications & bibliothèques** existent en plusieurs versions
 - *En décembre 2021 : ~700 sur Curta et ~200 sur PlaFRIM*
- pour choisir une version et y avoir accès, il faut charger les modules correspondants avec la commande
module load nom/du/module
- les modules sont rangés dans des rubriques qui dépendent de la plateforme
 - Ex : MCIA : **module load gcc/9.2.0**
PlaFRIM3 : **module load compiler/gcc/9.2.0**
- les utilisateurs peuvent contribuer en installant des modules partagés
 - Il faut demander à faire partir du groupe ad-hoc
(MCIA : **swadmins** / PlaFRIM3 : **plafrim-dev**)
- il peut y avoir des dépendances entre modules = obligation de charger un module avant un autre. Dans certains cas, cela peut se faire automatiquement.
- À noter : Installation automatiques / **guix** (PlaFRIM3), **spack** (curta) ...

Les modules - commandes

- Charger un module **module load *nom/du/module*** ... Décharger un module **module rm *nom/du/module***
- Lister les modules existants sur la plateforme : **module available** (abrev : **module av**)
- Cherche un module contenant un motif : **module avail |& grep -i *motif***
ex : **module av |& grep -i matlab**
- Lister les modules chargés dans votre session courante : **module list**
- Supprimer tous les modules de votre session courante : **module purge**

Les modules – notions + avancées

- Charger un module
module load *nom/du/module[/version]*
 - *Par défaut, prend la version « la plus récente »*
 - Ex : MCIA : **module load gcc**
PlaFRIM3 : **module load compiler/gcc**
- Un module défini ou modifie des variables d'environnement
 - **PATH** : chemin de recherche des commandes
 - **LD_LIBRARY_PATH** : chemin de rech. des bibliothèques dynamiques
 - **INCLUDE_PATH** : chemin de recherche des *includes*
 - ...

Les espaces disques

backup pérenne	rapide	partagé s/nodes	effacé auto	M CIA Curta	PlaFRIM3
X		X		/gpfs/home/\$USER 128Go+	/home/\$USER 20Go
	X	X	X	/scratch/\$USER ??To	
	X	X			/beegfs/\$USER 1To
X		X			/projets/«TEAM» 200Go
	XX		X	/tmp ??Go	/tmp ??Go

iRODS (via M CIA) : 10To - 3 répliques - en mode « ftp »
IMB /scratch : ??Go - non sauvegardé

Comment accéder à distance aux fichiers de son compte : sshfs

Permet d'accéder aux fichiers comme s'ils étaient sur votre poste de travail (montage de « système de fichiers »)

- Curta

```
mkdir /tmp/curta
```

```
sshfs curta: /tmp/curta
```

```
mkdir /tmp/curta-scratch
```

```
sshfs curta:/scratch/votrelogin /tmp/curta-scratch
```

- le répertoire /tmp/curta contient maintenant les fichiers
- fonctionne en lecture écriture

- PlaFRIM3

```
mkdir /tmp/plafrim
```

```
sshfs plafrim: /tmp/plafrim
```

```
mkdir /tmp/plafrim-beegfs
```

```
sshfs plafrim:/beegfs/votrelogin /tmp/plafrim
```

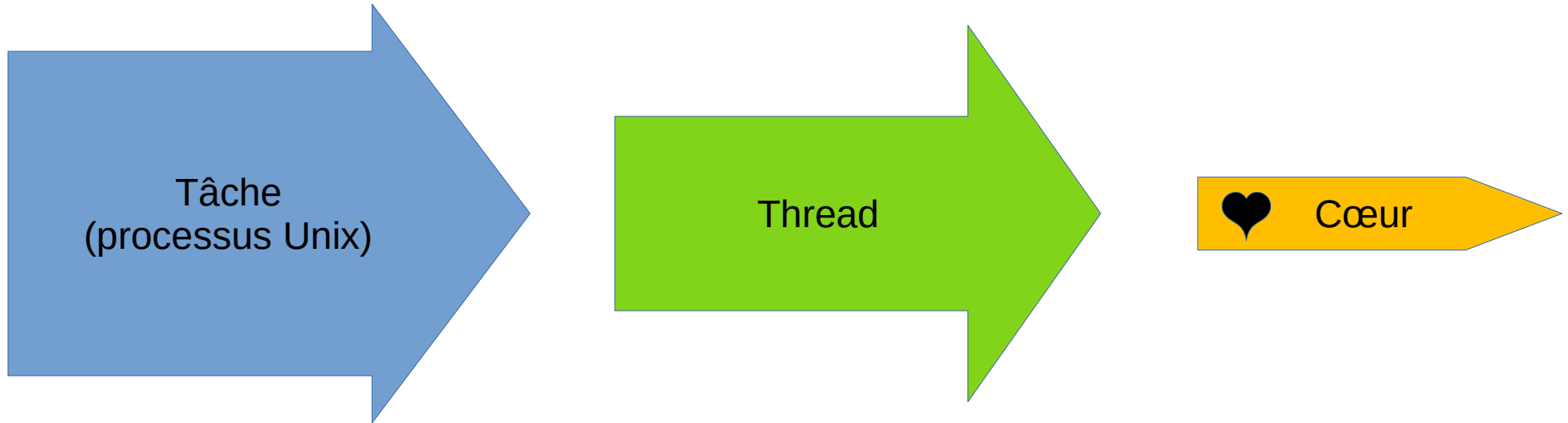
Quelques bonnes pratiques :

Contrôler l'utilisation CPU

Contrôler les cœurs affectés

Tâches, Threads, Coeurs

- Une tâche (processus unix) est par défaut constitué d'un seul thread.
 - Mais peut aussi en contenir plusieurs, la mémoire est partagée entre eux. Ex: OpenMP, pthreads
- Un thread s'exécute au maximum sur un cœur
- Efficacité max : $nb\ threads == nb\ coeurs$ (si CPU intensif)



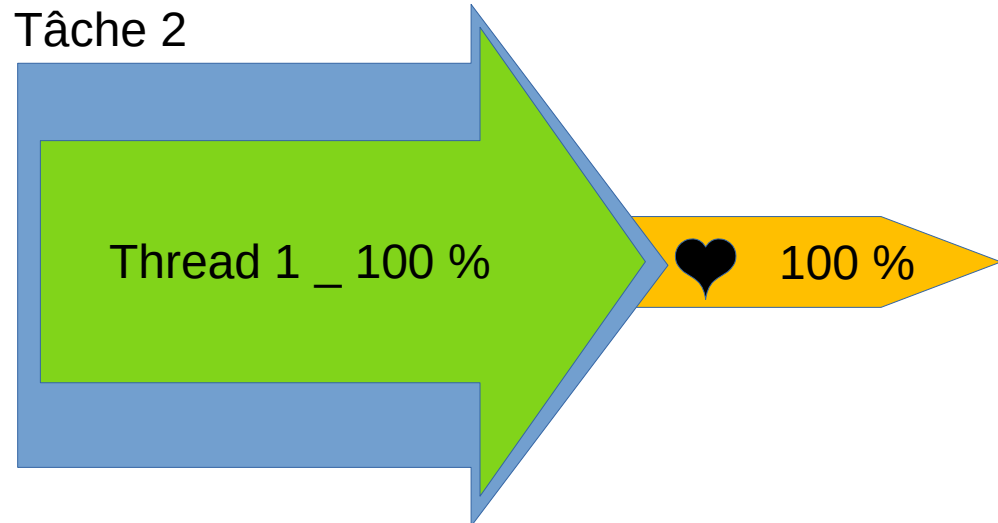
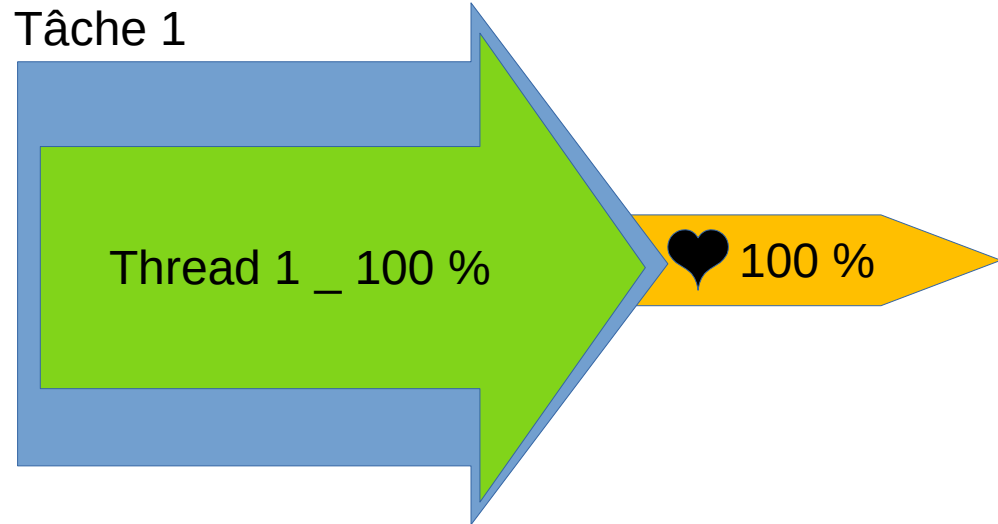
Threads, Tâche, Coeurs

monothread

1 thread / tâche
Nb threads = Nb coeurs

threads à 100 %
« tâches » à 100 %

lci : **-c1**



Threads, Processus, Coeurs

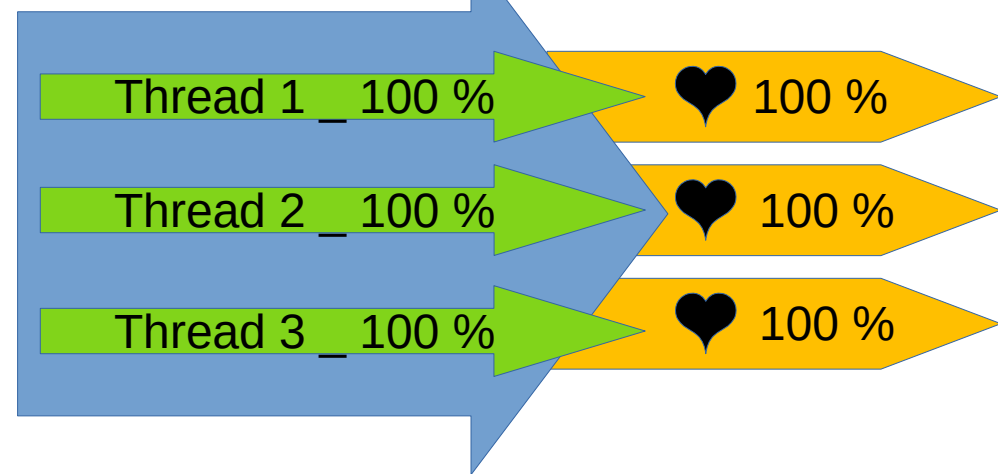
multithread

Nb threads = Nb coeurs

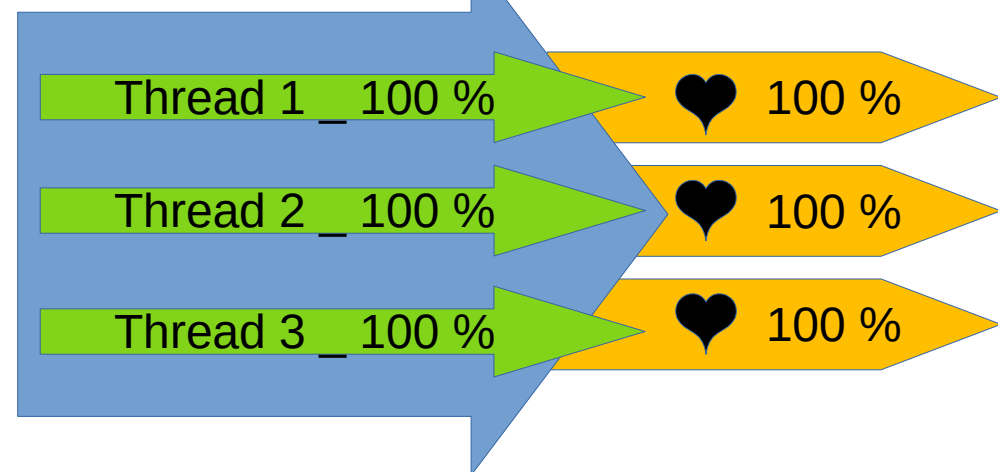
threads à 100 %
« tâches » à « 300 % »

ici : **-c3**

Processus 1

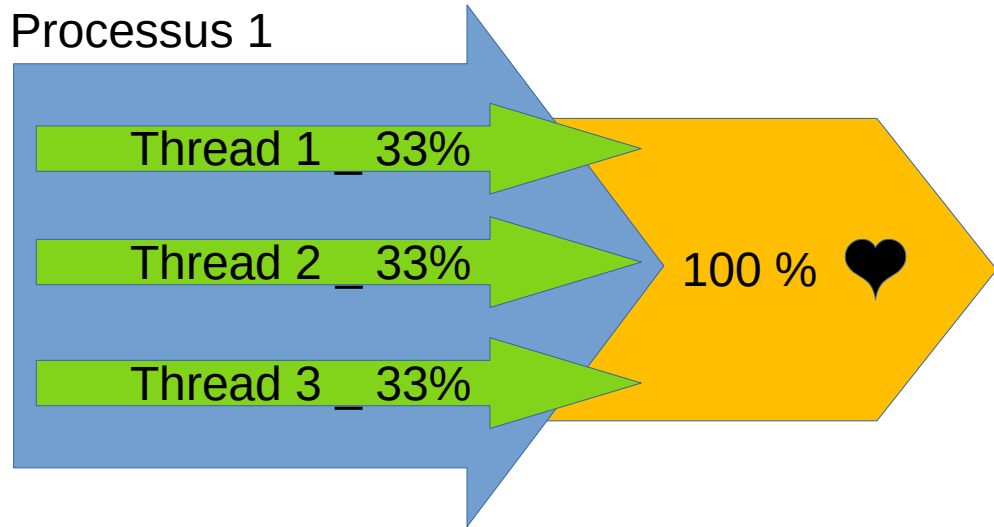


Processus 2

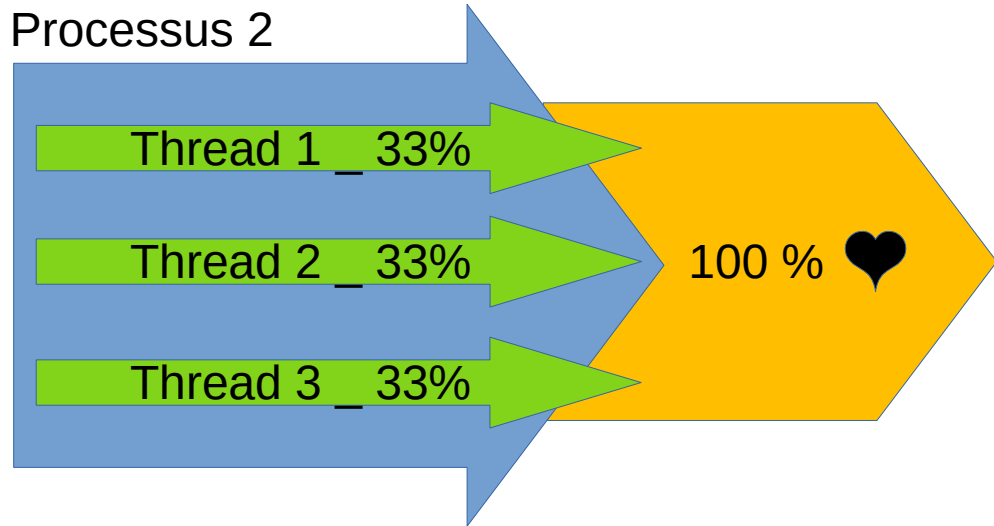


Threads, Processus, Coeurs

Processus 1



Processus 2



Nb Threads > Nb coeurs

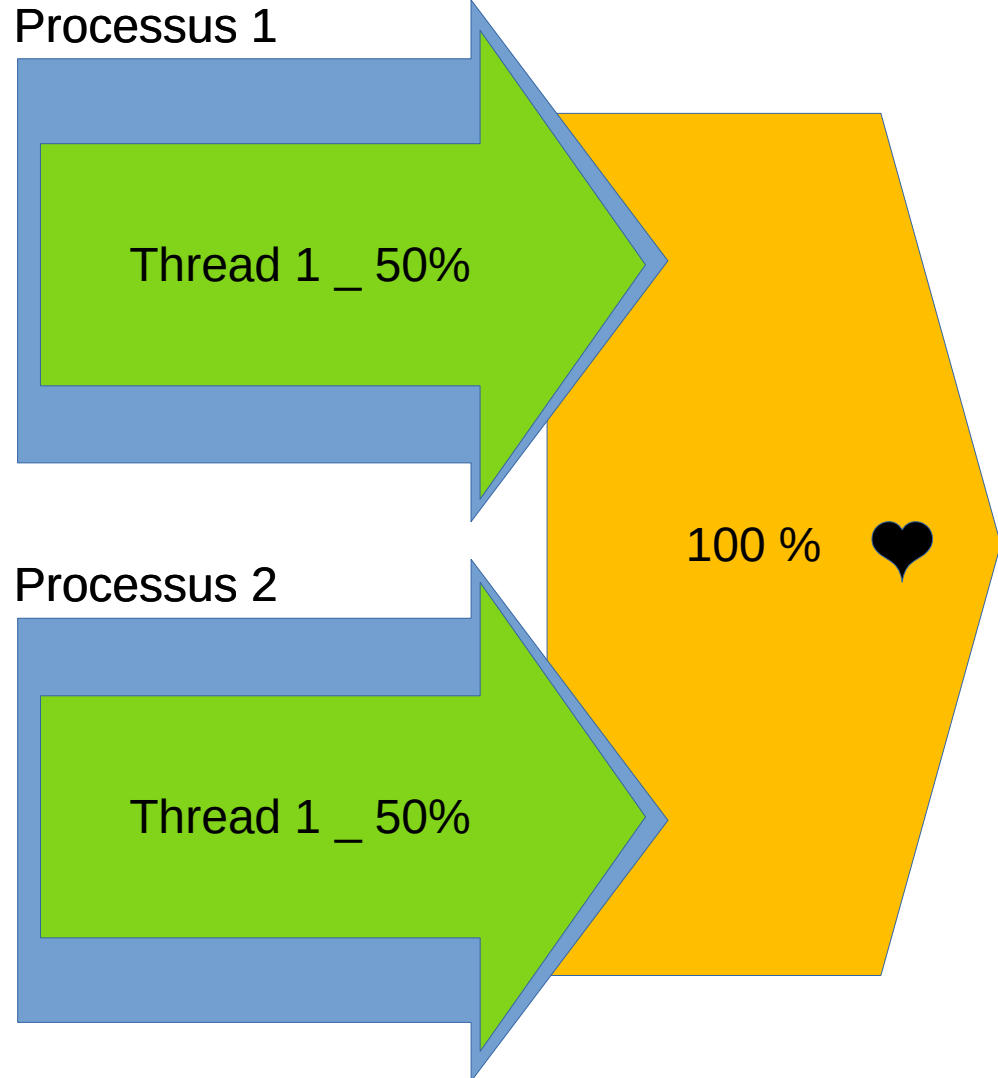
partage

Threads, Processus, Coeurs

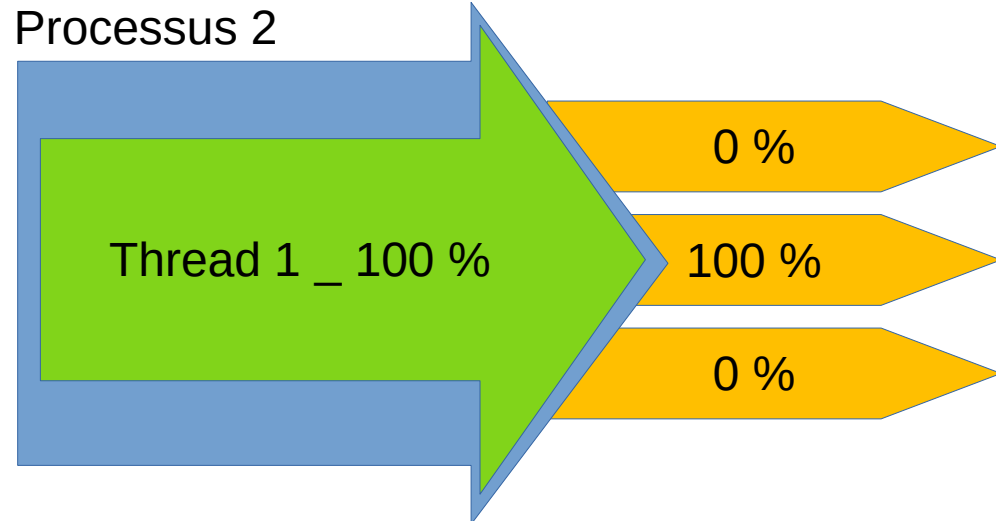
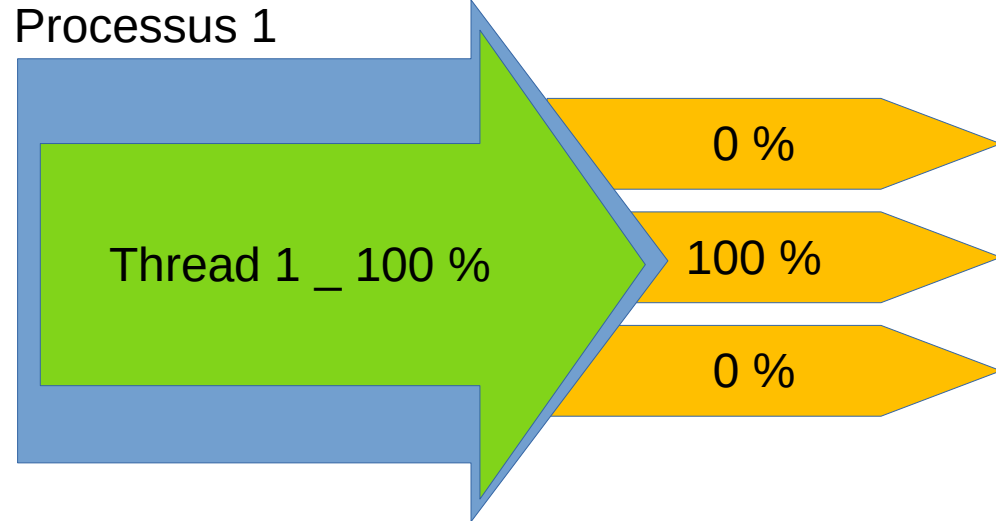
monothread

Nb tâches > Nb coeurs

partage



Threads, Processus, Coeurs



Nb coeurs > Nb Threads

Threads à 100 %

cœurs inutilisés

Bonnes pratiques :

Vérifier l'utilisation CPU

- Comment vérifier que les cœurs sont bien utilisés comme on pense qu'ils devraient l'être ?
- A faire :
 - avant de lancer une grosse campagne de calcul
 - à chaque modification dans vos scripts de lancement (batch)
 - à chaque changement de plateforme
- 2 Possibilités :
 - En live : **top**
 - En batch - permet la vérification a posteriori : **time, perf**

Vérifier l'utilisation CPU avec **top**

- Se connecter sur un nœud du job en cours d'exécution
- Lancer la commande **top**
- On regarde les cœurs
 - Touche « 1 » pour afficher (ou cacher) les cœurs
 - Vérifier qu'ils sont « tous » à ~100 %
- On regarde les processus (partie basse de l'affichage)
 - Ok si :
 - Plusieurs processus identiques à 100 % de CPU (processus mono thread)
 - Ex : 23 processus à 100 % \Leftrightarrow 23 cœurs actifs
 - Un processus unique avec une charge > 100 % (processus multithread)
 - Ex : charge CPU à 2300 % \Leftrightarrow 23 cœurs actifs

```

top - 16:19:51 up 6 days, 18:25, 2 users, load average: 2,91, 2,78, 1,34
Tasks: 163 total, 2 running, 161 sleeping, 0 stopped, 0 zombie
%Cpu0  :  0,0 us,  0,0 sy,  0,0 ni,100,0 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
%Cpu1  :  0,0 us,  0,0 sy,  0,0 ni,100,0 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
%Cpu2  :  0,0 us,  0,0 sy,  0,0 ni,100,0 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
%Cpu3  : 100,0 us,  0,0 sy,  0,0 ni,  0,0 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
%Cpu4  :  0,0 us,  0,0 sy,  0,0 ni,100,0 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
%Cpu5  :  0,0 us,  0,0 sy,  0,0 ni,100,0 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
%Cpu6  :  0,0 us,  0,0 sy,  0,0 ni,100,0 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
%Cpu7  :  0,0 us,  0,0 sy,  0,0 ni,100,0 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
MiB Mem : 128768,2 total, 122819,5 free, 2713,0 used, 3235,7 buff/cache
MiB Swap: 262144,0 total, 262144,0 free, 0,0 used. 124479,1 avail Mem

```

« top »
(touche « 1 »)

CPU

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
9334	lfacq	20	0	2246044	2,1g	15112	R	100,0	1,7	0:12.13	TKPok
1	root	20	0	106356	10860	8088	S	0,0	0,0	0:29.51	systemd
2	root	20	0	0	0	0	S	0,0	0,0	0:00.20	kthreadd
3	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	rcu_gp
4	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	rcu_par_gp
6	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	kworker/0:0H-kblockd
7	root	20	0	0	0	0	I	0,0	0,0	0:02.32	kworker/u16:0-rpciod

Processus

top - 16:19:51 up 6 days, 18:25, 2 users, load average: 2,91, 2,78, 1,34

Tasks: 163 total, 2 running, 161 sleeping, 0 stopped, 0 zombie

%Cpu0 : 0,0 us, 0,0 sy, 0,0 ni,100,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st

%Cpu1 : 0,0 us, 0,0 sy, 0,0 ni,100,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st

%Cpu2 : 0,0 us, 0,0 sy, 0,0 ni,100,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st

%Cpu3 : 100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st

%Cpu4 : 0,0 us, 0,0 sy, 0,0 ni,100,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st

%Cpu5 : 0,0 us, 0,0 sy, 0,0 ni,100,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st

%Cpu6 : 0,0 us, 0,0 sy, 0,0 ni,100,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st

%Cpu7 : 0,0 us, 0,0 sy, 0,0 ni,100,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st

MiB Mem : 128768,2 total, 122819,5 free, 2713,0 used, 3235,7 buff/cache

MiB Swap: 262144,0 total, 262144,0 free, 0,0 used. 124479,1 avail Mem

« top »

phase de calcul
séquentielle

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
9334	lfacq	20	0	2246044	2,1g	15112	R	100,0	1,7	0:12.13	TKPok
1	root	20	0	106356	10860	8088	S	0,0	0,0	0:29.51	systemd
2	root	20	0	0	0	0	S	0,0	0,0	0:00.20	kthreadd
3	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	rcu_gp
4	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	rcu_par_gp
6	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	kworker/0:0H-kblockd
7	root	20	0	0	0	0	I	0,0	0,0	0:02.32	kworker/u16:0-rpciod

```

top - 16:20:17 up 6 days, 18:25, 2 users, load average: 3,03, 2,80, 1,38
Tasks: 163 total, 2 running, 161 sleeping, 0 stopped, 0 zombie
%Cpu0  : 0,0 us, 0,0 sy, 0,0 ni,100,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu1  : 98,4 us, 0,0 sy, 0,0 ni, 1,6 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu2  : 96,9 us, 0,0 sy, 0,0 ni, 3,1 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu3  : 100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu4  : 98,4 us, 0,0 sy, 0,0 ni, 1,6 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu5  : 96,8 us, 0,0 sy, 0,0 ni, 3,2 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu6  : 96,8 us, 0,0 sy, 0,0 ni, 3,2 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu7  : 0,0 us, 0,0 sy, 0,0 ni,100,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
MiB Mem : 128768,2 total, 122081,6 free, 3450,9 used, 3235,7 buff/cache
MiB Swap: 262144,0 total, 262144,0 free, 0,0 used. 123741,2 avail Mem

```

« top »
phase de
calcul
parallèle
sur
6 cœurs

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
9334	lfacq	20	0	3367776	2,8g	15112	R	585,7	2,3	1:24.15	TKPok
1	root	20	0	106356	10860	8088	S	0,0	0,0	0:29.51	systemd
2	root	20	0	0	0	0	S	0,0	0,0	0:00.20	kthreadd
3	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	rcu_gp
4	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	rcu_par_gp
6	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	kworker/0:0H-kblockd
7	root	20	0	0	0	0	I	0,0	0,0	0:02.32	kworker/u16:0-rpciod

Se connecter sur un nœud attribué à un de mes jobs

- Solution 1 - PlaFRIM3 & MCIA :

srun --overlap --jobid *ID* --pty bash

- on arrive par défaut sur le premier nœud du job, préciser si besoin **-w NODE**
- **--overlap** : option pour forcer l'accès, au risque de perturber un peu le job)

Solution 2 - PlaFRIM3 Only : **ssh NODE**

Dans les 2 cas : possible uniquement pendant l'exécution d'un job

TP-5 observer un job qui tourne

- Créer un batch pour lancer 4 instances de « `cpu_burn 600` », sur 2 nœuds, pendant 5 minutes
- Lancer le batch puis connectez vous sur le premier nœud du jobs pour lancer « `top` » et contrôler la consommation CPU
- Que constatez-vous ?

TP-5 solution 1/2

```
$> module load formations/toys
```

```
$> cat > batch_600
```

```
#!/bin/bash
```

```
#SBATCH -N2 -n4 -t5
```

```
srun cpu_burn 600
```

```
(ctrl-d)
```

```
$> sbatch batch_600
```

```
$> queue -u $USER
```

```
$> srun --overlap --jobid XXXXXX --pty bash
```

```
$nodeXX> top
```

```
# touche « 1 » pour afficher les cœurs
```

TP-5 solution 2/2

On constate que la répartition est 3 / 1 au lieu de 2 / 2 à laquelle on s'attend...
La stratégie utilisée est : « une tâche sur chaque nœud, puis on remplit les nœuds »

Pour constater la répartition « 3 / 1 au lieu de 2 / 2 » il suffit de faire :

```
$> srun -t1 -N2 -n4 hostname
```

```
n294 } 1  
n293 }  
n293 } 3  
n293 }
```

Vérifier l'utilisation CPU avec **time** (1/2)

- Remplacer dans votre batch les lignes du type :
srun ... MonProgramme
 - Par :
srun ... /bin/time --verbose MonProgramme
 - La commande **time** récupère plusieurs compteurs liés à l'exécution dont la **charge CPU moyenne*** consommée (et une approximation de la consommation mémoire : *Maximum resident...*)
 - Comme avec **top** un processus avec une charge > 100 % signifie plusieurs coeurs utilisés
 - Ex : charge CPU à 2300 % => 23 coeurs actifs (équivalent à)
- * : *ou puissance CPU moyenne*

/bin/time exemple d'output

\$ /bin/time --verbose ls

...

Command being timed: "ls"

User time (seconds): 0.00

System time (seconds): 0.00

Percent of CPU this job got: 30%

Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.01

Average shared text size (kbytes): 0

Average unshared data size (kbytes): 0

Average stack size (kbytes): 0

Average total size (kbytes): 0

Maximum resident set size (kbytes): 1024

Average resident set size (kbytes): 0

Major (requiring I/O) page faults: 0

Minor (reclaiming a frame) page faults:
323

Voluntary context switches: 24

Involuntary context switches: 26

Swaps: 0

File system inputs: 0

File system outputs: 0

Socket messages sent: 0

Socket messages received: 0

Signals delivered: 0

Page size (bytes): 4096

Exit status: 0

Vérifier l'utilisation CPU avec **time** (2/2)

- Avec un code de calcul « CPU intensif » (CPU bound) vous devriez atteindre :
 - 100 % CPU pour chaque code séquentiel (un processus par cœur)
 - $c * 100\%$ CPU pour un code avec 'c' thread par processus
- Dans le cas contraire, vos paramètres d'exécution sont probablement incorrectes
- Vérifiez avec **top** pendant l'exécution

Identifier les n° des cœurs réservés



```
srun -c12 grep Cpu /proc/self/status  
Cpus_allowed:000000....00000,00fc003f  
Cpus_allowed_list: 0-5,18-23
```

- Cpu_allowed : masque binaire
- Cpu_allowed_list : liste des cœurs réservés
- Dans vos scripts : **grep Cpu /proc/self/status**

Note : la réservation précise des cœurs peut parfois être défailante

Autres infos diverses

« worker-real » : solution pour traiter un grand nombre de petits jobs

- Cible
 - Lancer un grand nombre de tâches tournant chacune sur moins d'un nœud
 - Gestion de l'état de chaque tâche (traitée ou non) / reprise sur problème/crash
- Prérequis
 - Dans un répertoire, écrire un fichier (script) par tâche
- Fonctionnement
 - Des agents puisent dans ce répertoire les tâches à exécuter
- Avantages
 - Simple, Robuste, Efficace
- <https://plmlab.math.cnrs.fr/laurent.facq/worker-real>

déport graphique (X11)

ex : matlab, visit, paraview...

- Visualisation 3D, traitement d'image, graphiques interactifs, ...
- Simple...
 - **srun --x11** ... mais très peu performant (voir inutilisable)
- Avec TurboVNC
 - Optimisé/Accéléré
 - Procédure PlaFRIM3 :
<https://www.math.u-bordeaux.fr/imb/cellule/plafrim3-visualisation-deportee-ex-matlab>
 - Procédure MCIA
https://redmine.mcia.fr/projects/cluster-curta/wiki/Visualisation_d%C3%A9port%C3%A9e
 - Sur machines dédiées pour la visualisation 3D avec cartes GPU d'accélération
 - PlaFRIM3 : visu01
 - Curta : visu01 à visu04

Fin

Merci pour votre attention !

et surtout merci pour votre feedback à venir :

- Des éléments essentiels manquants ?
- Des éléments pas clairs ou mal dits ?
- **Des éléments trompeurs qui vous ont induit en erreur ?**

=> merci de m'en faire part afin d'améliorer cette présentation :-)