Les 10 choses à connaître pour bien utiliser Froggy

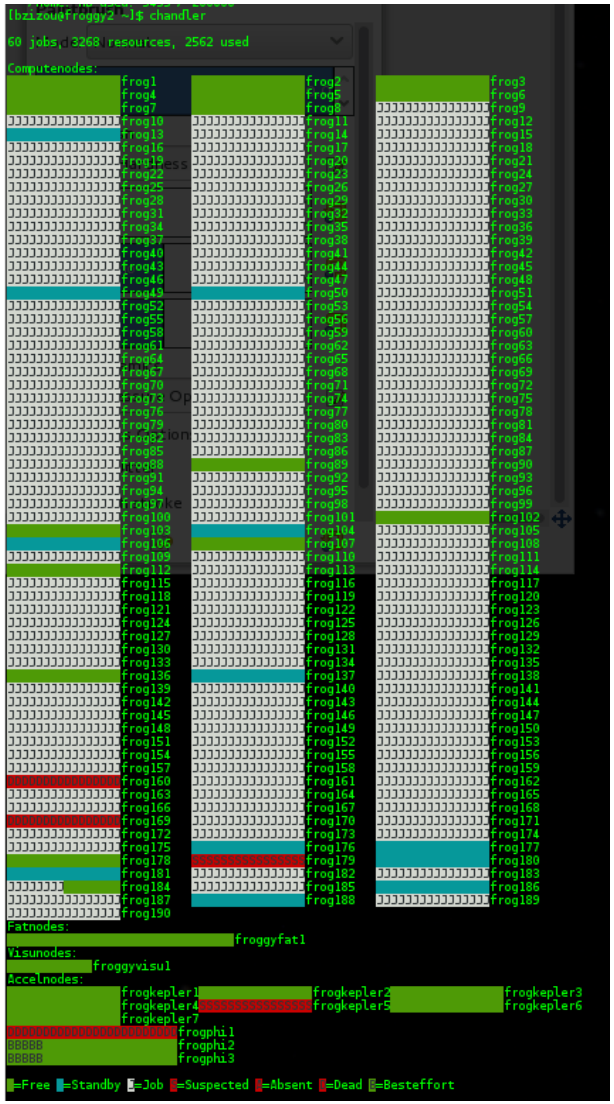
Journée des utilisateurs CIMENT – B. Bzeznik, F. Roch
14 mai 2014

Préliminaire : Chandler

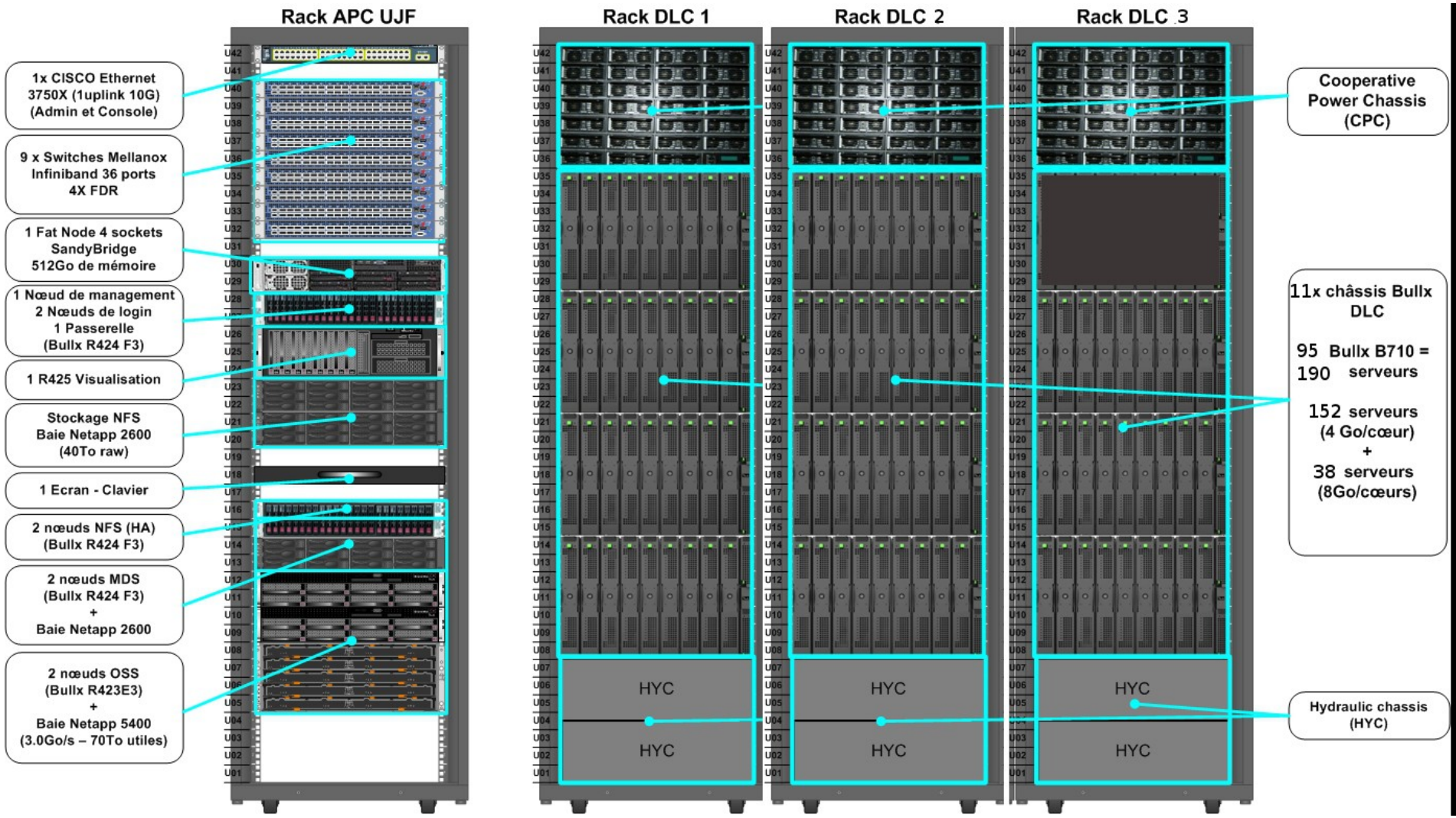
- La 11ème chose à connaître :

  La commande ***chandler*** lancée depuis une frontale permet de visualiser un aperçu de la machine et de l'état de ses nœuds

- Page wiki des 10 choses : 10_choses_froggy



1. Connaître l'architecture de la machine



1. Connaître l'architecture de la machine

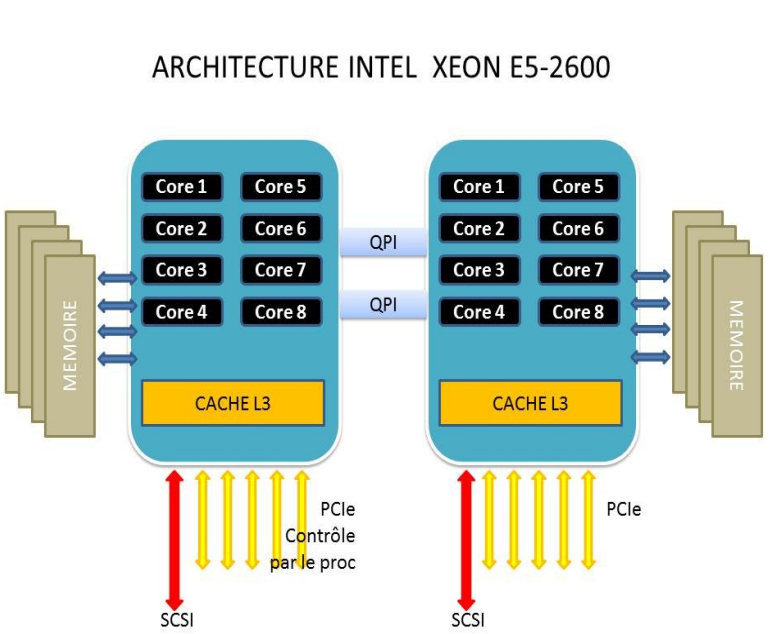
Machine Bullx constituée de :

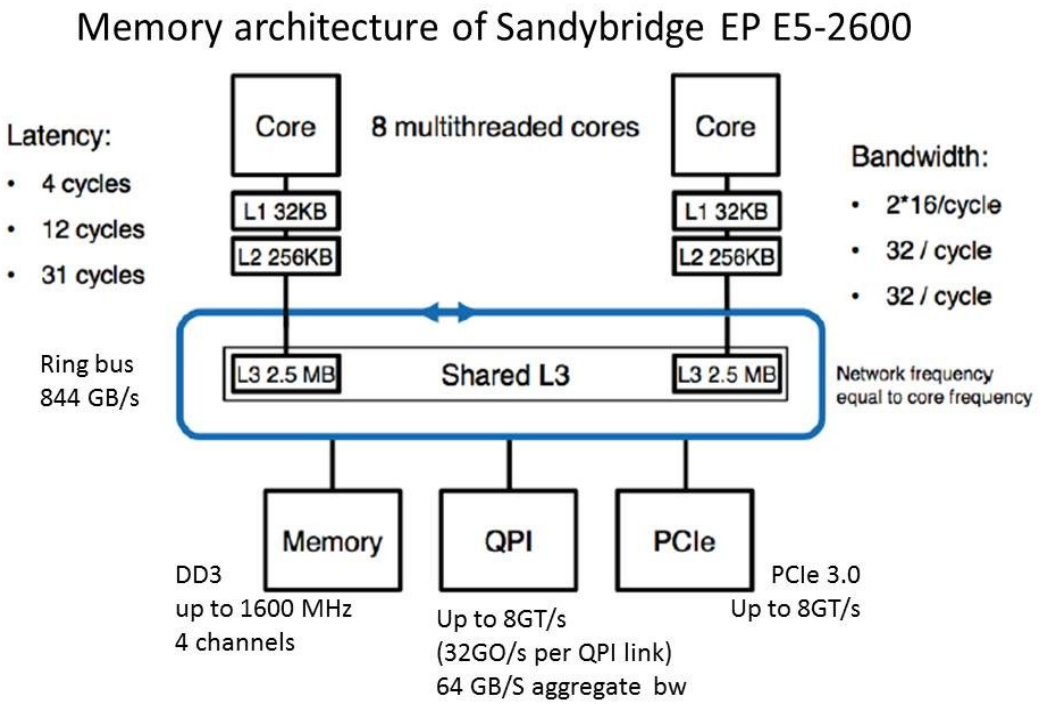
- 190 noeuds bi-processeur Sandybridge E5-2600 à 2.6 GHz, interconnectés par un réseau Infiniband FDR non bloquant
- Un espace de stockage distribué Lustre de 90 To (/scratch)
- 2 noeuds de login : *froggy1* et *froggy2*
- 2 sortes de noeuds de calcul, en fonction de la mémoire dont ils disposent :

  frog[1-108,137-175,182-184,190] ont 64 Go

  frog[109-136,176-181,185-189] ont 128 Go
- 7 noeuds avec GPU : *frogkepler[1-7]*
- 3 noeuds en prêts : *frogphi[1-3]*
- 1 noeud de visualisation : *froggyvisu1*
- 1 noeud quadri-pros : *froggyfat1*

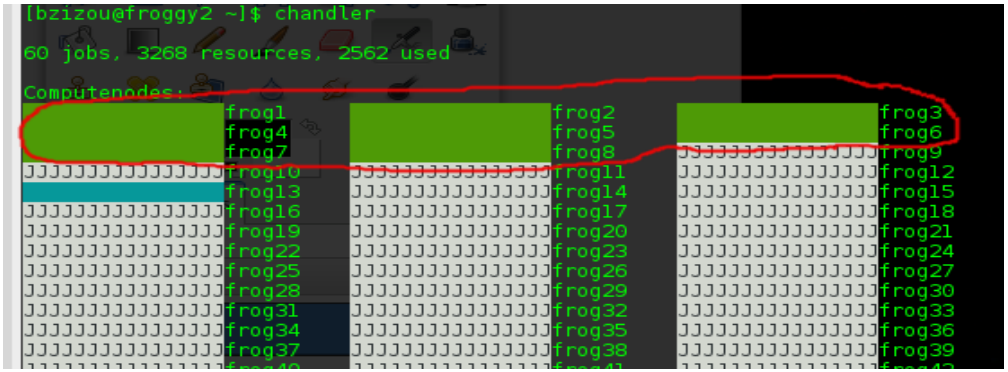
1. Connaître l'architecture de la machine





- Chaque processeur a 3 niveaux de cache
- Une partie de la mémoire lui est physiquement rattachée

2.Utiliser le bac à sable

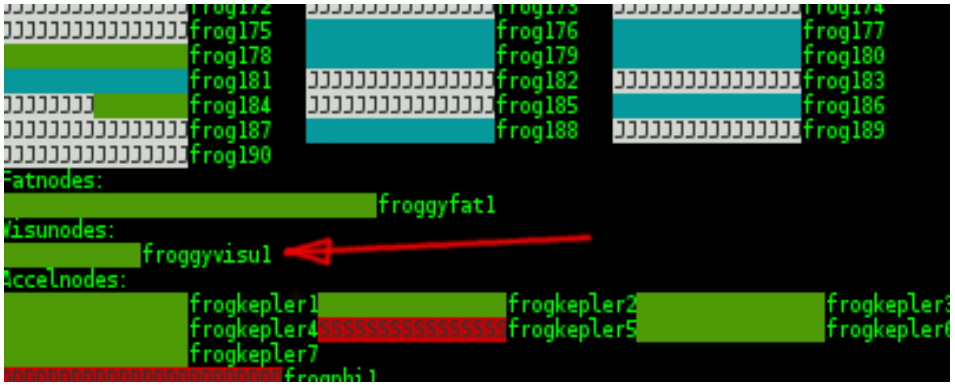


- Tests intéractifs à répétition
- **8 premiers nœuds** (128 cores) réservés pour les jobs de moins de **30 minutes**
- Soumission :

```
oarsub -I -t devel --project test
```

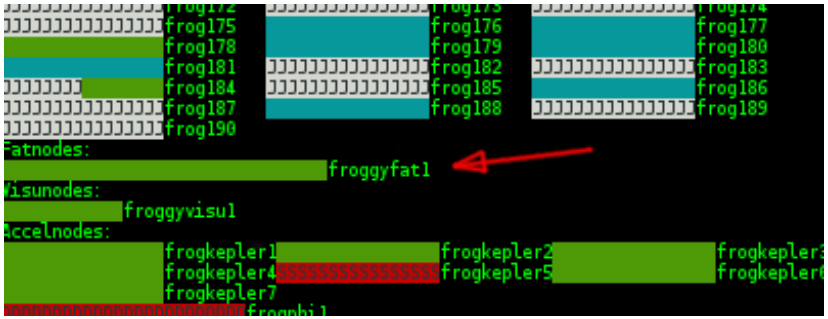
- En phase de tests, vous **DEVEZ** l'utiliser !
- La limite de 30min peut être dépassée pour les jobs « best-effort »

3.Utiliser le noeud de visu in situ



- Nœud dédié équipé de 2 cartes graphiques Nvidia Quadro 6000 accédé par TurboVNC / VirtualGL
  - Carte graphique puissante pas forcément dispo sur votre poste de travail (environ 4000€ la carte!)
  - Données directement accessibles sur les systèmes de fichiers partagés sur le réseau rapide de Froggy
  - Possibilité de faire des jobs hybrides avec visu interactive (calculs sur les nœuds de calcul, et visu sur le nœud de visu déportée sur votre poste de travail)
- Script de soumission : **visu_sub**

4.Utiliser le Fat node



- Froggyfat1 : 4 proc SandyBridge
- 512 Go de mémoire
- Soumission :

```
oarsub -l /core= 4 -t fat –project test
```

Applications cibles :

jobs OpenMP jusqu'à 32 threads (1 thread/core)

jobs séquentiels nécessitant une large mémoire

5.Bien choisir son espace de stockage et connaître ses contraintes (part 1)

Désignation	Point de montage	Techno	Variable env.	Volumétrie	Partage
Scratch	/scratch	LUSTRE	$SHARED_SCRATCH_DIR	90To (quot. 300Go)	Noeuds de Froggy
Local scratch	/tmp	EXT4	$LOCAL_SCRATCH_DIR	45Go par noeud	Non
Home	/home	NFS	$HOME	30To (quot. 50Go)	Noeuds de Froggy
Ciment Global Storage		iRods		>700To	Noeuds de tous les clusters CIMENT

5.Bien choisir son espace de stockage et connaître ses contraintes (part 2)

Désignation	Utilisation	Attention !
Scratch (/scratch)	• Calculs avec accès // aux fichiers • Bonnes perfs (3Go/s) mais partagées !	Surcharge => CLUSTER EN PANNE !
Local scratch (/tmp)	• Calculs sans accès // aux fichiers • Perfs indépendantes de la charge (disque local, 100Mo/s)	• Surcharge => nœud en panne • Seulement 45Go/noeud !
Home (/home)	• Stockage fichier vitaux (programmes, librairies, résultats finaux,...) • Perfs médiocres !	NE PAS EXPLOITER PENDANT LE CALCUL !
Ciment Global Storage (*iRods*)	• Archivage • Calculs sur grille • Import/export vers le mone extérieur • Un « cloud » de stockage • Bonnes perfs si on distribue les fichiers	• Non POSIX (utilise put/get) • Commandes échouent en cas de surcharge

5.Bien choisir son espace de stockage et connaître ses contraintes (part 3)

- Aucun des systèmes de fichiers proposés ne supporte sans problèmes la surcharge d'I/O !
- Aucun ne traite efficacement de très nombreux petits fichiers !

=> *les solutions sont au niveau de l'optimisation des codes*

- Pas de sauvegardes !
  https://ciment.ujf-grenoble.fr/wiki/index.php/About_backups

6.Comment choisir son environnement de développement (1/2)

Désignation	Chargement

Environnement Site (contient aussi BullX_DE)	`source /applis/site/env.bash`
	`module avail`
	`module load [...]`
	`# Pour BullX DE:`
	`module load bullxde`

Environnement grille	`source /applis/ciment/v2/env.bash`
	`module avail`
	`module load [...]`

Environnement par défaut	*rien à charger*

6.Comment choisir son environnement de développement (2/2)

Env.	Avantages	Inconvénients
Site	• Libs optimisées • Simple	Spécifique à Froggy
Grille	• Le même partout (CIMENT) • Beaucoup de modules (du fait de la mutualisation)	Afin de fonctionner, il embarque sa propre librairie C; ce qui peut rendre plus complexe l'utilisation -> nombreuses dépendances de modules
Défaut	Très simple, rien n'est à charger	• Il faut se contenter de ce qu'il y a! • Peut changer au fil des mises à jour du système

7.Savoir quels outils sont disponibles pour optimiser, profiler, debugger

- Commande **/usr/bin/*time*** LINUX/UNIX : mesure du temps global ***user, system, elapsed***

  Résolution de l'ordre de 1-10ms
- Mesure de temps sur une partie de code :
  - ***cpu_time()*** ou ***date_and_time()*** (Fortran90/95)
  - ***system_clock()***
- Commande GNU ***gprof*** : mesure du temps passé dans les sous-programmes, arbre des appels. Compiler et linker avec ***-p*** *et* ***-g***.

  Outil non intrusifs. Un fichier **gmon.out** est généré à l'exécution.

7.Savoir quels outils sont disponibles pour optimiser, profiler, debugger

Valgrind :

- Outil de debug et de profiling dynamique, non intrusif, option ***-g*** conseillée
- Valgrind prend le contrôle du programme dès son démarrage
- Il vérifie instruction par instruction la présence de variables non initialisées, d'accès mémoire invalides, de fuites mémoire ...
- Plusieurs outils en 1 :
  - Valgrind **--tool=memcheck** ./myprog # détection de pb mémoire
  - Valgrind **--tool=cachegrind** ./myprog # simulation de cache
  - Valgrind **--tool=callgrind** ./myprog # outil de profiling
  - ...
- Attention : Valgrind dégrade les performances

7.Savoir quels outils sont disponibles pour optimiser, profiler, debugger

Outils de debugging , utiliser l'option ***-g*** du compilateur (et éventuellement d'autres options de debug)

- Outil GNU ***gdb***

  Option du compilateur ***-g***

  ```
  gdb ./myprog
  ```

  Pour un program Intel MPI

  ```
  mpiexec -gdb .... ./myprog
  ```

- Outil Intel ***idb***, *support MPI*

  Session graphique

  ```
  idb ./myprog
  ```

  Session en ligne de commande

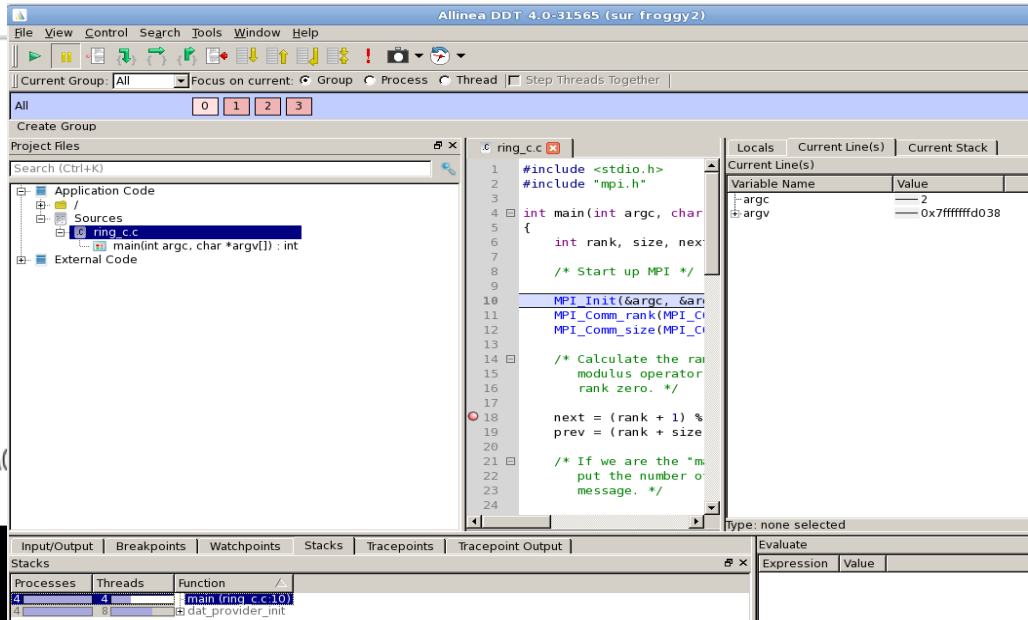
  ```
  idbc ./myprog
  ```

7.Savoir quels outils sont disponibles pour optimiser, profiler, debugger

Allinea ***ddt***

- **Debugger parallèle** avec interface graphique
- 2licences de 32 process simultannés **sur froggy2**
- Option du compilateur : ***-g -debug***
- Module ***allinea-ddt***
- Préparer un **template**

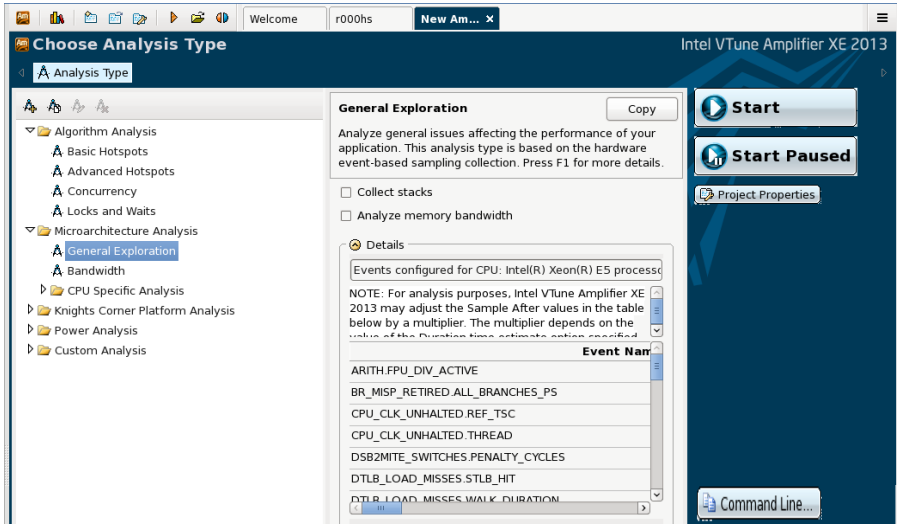
```
#!/bin/bash
#OAR -l nodes=4,walltime=0:05:00
#OAR -t devel
#OAR --project admin
source /applis/site/env.bash
module load intel-devel
ulimit -l 4086160
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/lib64
export I_MPI_FABRICS=shm:dapl
mpiexec.hydra -genvall -f $OAR_NODE_FILE -bootstrap-exec oarsh -n 4 \
  DDTPATH_TAG/bin/ddt-debugger NUM_PROCS_TAG PROGRAM_TAG PROGRAM_ARGUMENTS_TAG
```



7.Savoir quels outils sont disponibles pour optimiser, profiler, debugger

Outil Intel ***Vtune*** pour analyser un code MPI Intel

- Plusieurs catégories d'analyse
- Identifier des « hotspots »
- Possibilité d'accès aux compteurs hardware (PAPI)

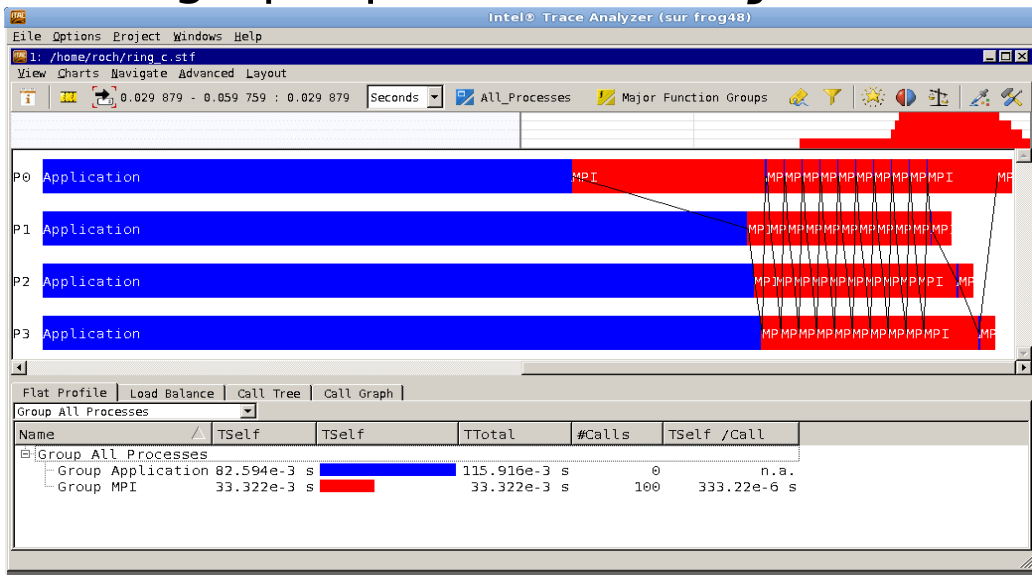


1- Utiliser l'interface graphique **amplxe-gui** pour sélectionner un type d'analyse (New-Analysis)
2- Utiliser le bouton "**Command Line**" pour récupérer la ligne de commande permettant de collecter les données sur un noeud de calcul
3- Inclure la ligne de commande au niveau de **mpirun**, dans un script OAR que vous soumettez via la commande oarsub -S
4- Analyser les résultats via **amplxe-gui** (Open-Results).

7.Savoir quels outils sont disponibles pour optimiser, profiler, debugger

Outil Intel ***tracecollector/traceanalyzer (itac)*** pour analyser un code MPI Intel

- Module ***intel-devel***, compiler avec l'option ***-trace***
- Exécuter le code pour générer un fichier de trace (***.stf***) avec ***tracecollector***
- Analyser les traces avec l'interface graphique ***traceanalyzer***
  - Frises chronogiques
  - Statistiques sur les communications
  - Statistiques d'exécution



Voir également l'outil MPIAnalyser

module load intel bullxde mpianalyser

et SCALASCA (pour d'autres MPI)

8.Savoir écrire des scripts pour OAR

Entête

```
#!/bin/bash
#OAR --project test
#OAR --name Test_Job
set -e
```

Directives OAR

Environnement

```
source /applis/site/env.bash
module load intel-devel/14
WORK_DIR=$SHARED_SCRATCH_DIR/$USER/Test_Job
TMP_DIR=$SHARED_SCRATCH_DIR/$USER/oar.$OAR_JOB_ID
```

Préparation et lancement
De l'application

```
mkdir -p $WORK_DIR
cd $WORK_DIR
mpirun [...]
```

Lancement : `oarsub -S`

9.Etre au courant des règles de partage des ressources

Règles communes :

- Charte
- OAR : FIFO + Fairsharing sur fenêtre glissante de 3 mois
- Durée max des jobs : 4 jours
- Temps CPU max des jobs : 8192 h (4h sur 2048 cœurs)
- Pas de limite bac-à-sable pour les jobs best-effort
- Priorité au calcul parallèle

Règles spécifiques :

- Karma (indice fairshare) pondéré pour les contributeurs
- Queue particulière pour les contributeurs de nœuds spécifiques (GPU)

10.Savoir obtenir de l'aide

- Se tenir au courant (listes de diffusion)
- Trouver les compétences locales (responsable désigné lors de la signature de la charte labo)
- Utiliser le wiki ! **https://ciment.ujf-grenoble.fr/wiki**
- Pour signaler un problème : ouvrir un ticket sur la plateforme GLPI de Ciment ou envoyer un mail à **sos-ciment@ujf-grenoble.fr**

Le Wiki des 10 choses !

Toutes les rubriques de cette présentation sont détaillées sur la page wiki du même nom !

Donc, pour approfondir, rendez-vous ici :

https://ciment.ujf-grenoble.fr/wiki/index.php/10_choses_froggy