Comment executer un programme sur le cluster de l'UCI de l'universite d'Oran1 ?

Le cluster de l'UCI de l'universite d'Oran1 est un Bullx[1], dont le gestionnaire des travaux (jobs) est
Slurm[2]. En se connectant sur le noeud de management du cluster, vous aurez a votre disposition :

- les compilateurs standards C/C++, Fortran, et Java
- les librairies mathematiques standards (e.g., Lapack, Blas, ...)
- les compilateurs pour les programmes MPI

Pour executer un programme parallele, il est necessaire de proceder en deux etapes :

1. Compiler les sources avec des compilateurs MPI (ou des compilateurs standards, si votre
programme est sequentiel, ou en OpenMP)
2. Executer le programme parallele avec la commande ≪ sbatch ≫ du gestionnaire des travaux
Slurm en utilisant un script fixant les parametres d'execution

Exemple d'un programme parallele

Soit un programme de calcul parallele de π, donne a la fin de ce memo [3].

Comment compiler ?

```
mpicc -Wall -O2 pi-parallel.c -o pi-parallel
```

Comment executer ?

On lance l'execution avec la commande suivante :

```
sbatch job.bat
```

Le contenu du fichier job.bat :

```
#!/bin/bash
#SBATCH -n12
#SBATCH -ntasks-per-node=12
echo $SLURM_JOB_ID $SLURM_TASKS_PER_NODE
mpirun pi-parallel 10000000
```

La ligne ≪ #SBATCH -n12 ≫ precise que le programme utilisera 12 taches paralleles (1 tache si le
programme est sequentiel).
La ligne ≪ #SBATCH --ntasks-per-node=12 ≫ precise que le programme lancera 12 taches par noeud.
Enfin, la ligne ≪ mpirun pi-parallel 10000000 ≫ montre la commande d'execution du programme.
Ici, 10 000 000 est le nombre de trapezes a utiliser dans le calcul parallele de π (pour plus de details sur
ce programme, voir le lien dans la note de bas de page.).
Nous insistons a ce que le script utilise un nombre raisonnable de noeuds, suffisant par rapport a vos
besoins. Toute exageration dans le nombre de noeuds demandes va simplement sanctionner les autres

utilisateurs du cluster.

1 http://www.bull.fr/extree-co m mputing/bullx.html

2 https://computing.llnl.gov/linux/slurm/

3 http://www.unixgarden.com/index.php/gnu-linux-magazine/programmation-sur-un-cluster-calculer-pi-avec-mpi

Programme π

http://www.unixgarden.com/index.php/gnu-linux-magazine/programmation-sur-un-cluster-calculer-pi-a

vec-mpi

```
/*
* pi - calculer pi
* sur un cluster avec MPI
* Copyright 2006 Thibault GODOUET
*/
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
/* Inclure les declarations
* des fonctions MPI */
#include "mpi.h"
/* Nombre de noeuds dans le cluster,
* et rang de ce noeud */
int size, rank;
/* Pour calculer le temps de calcul: */
double start_time, end_time;
/* printf() avec le rang du noeud
* affiche au debut de la ligne */
void
xprintf(char *fmt, ...)
{
va_list args;
va_start(args, fmt);
printf("noeud %d: ", rank);
vprintf(fmt, args);
} /* Sortir propremment du programme,
* et afficher le temps de calcul */
void
xexit(int code)
{
if ( rank == 0 ) {
end_time = MPI_Wtime();
xprintf("TEMPS DE CALCUL: "
"%f secondes¥n",
end_time - start_time);
```

```
}
MPI_Finalize();
exit(code);
} /* Fonction a
integrer
* pour calculer pi: */
/* NOTE: le "inline" n'est la que pour
* (peut-etre!) ameliorer
* les performances et peut etre enlever
* sans probleme */
inline
double
f(double x)
{
return (4.0 / (1.0 + x*x));
}
/* Calcul de la partie de la somme
* approximant l'integrale de f()
* a faire par ce noeud */
double
sub_sum(long long int start_i,
long long int stop_i, double h)
{
double f_xi, f_xi1;
long long int i = 0;
double sum = 0;
double x = 0;
xprintf("Sous-somme de %Ld a %Ld¥n",
start_i, stop_i);
sum = 0;
x = h*(start_i-1);
f_xi1 = f(x);
for ( i = start_i; i <= stop_i; i++ ) {
f_xi = f_xi1;
f_xi1 = f(x+h);
sum += (f_xi+f_xi1);
x += h;
}
sum *= h/2.0;
return sum;
}
int
main(int argc, char **argv)
{
```

```
/* nombre de trapezes: */
long long int num_trap = 0;
long long int i = 0,
start_i = 0, stop_i = 0;
double h = 0;
double sum = 0;
/* buffer pour recuperer
* les sous-sommes des differents
* noeuds: */
double *sum_buf = NULL;
size = rank = -1;
/* En cas de probleme (deadlock,
* boucle infinie, etc), arreter le
* programme au bout de 60 secondes: */ alarm(600);
MPI_Init(&argc, &argv);
MPI_Barrier(MPI_COMM_WORLD);
start_time = MPI_Wtime();
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
if (rank == 0) {
/* Noeud maitre*/
xprintf("Taille du cluster: %d¥n",
size);
/* Lire le nombre de trapezes a
* utiliser pour le calcul de
* l' integrale */
num_trap = atoll(argv[1]);
xprintf("Nombre de trapezes: %Ld¥n",
num_trap);
}
/* Envoyer/recevoir le nombre
* de trapezes */
MPI_Bcast((void *) &num_trap, 1,
MPI_LONG_LONG_INT, 0,
MPI_COMM_WORLD);
/* Calculer notre partie
* de la somme: */
h = ( 1.0 - 0.0 ) / (double)num_trap;
start_i = 1 + num_trap * rank / size;
stop_i = num_trap * (rank + 1) / size;
sum = sub_sum(start_i, stop_i, h);
/* recuperer toutes les valeurs
* des sous-sommes */
if ( rank == 0 )
```

```
/* Noeud maitre: c' est nous qui
* recuperons les valeurs, nous
* devons donc allouer de la memoire
* a cette fin */
sum_buf = (double*)malloc((sizeof(double))*size);
MPI_Gather((void *)&sum, 1,
MPI_DOUBLE, sum_buf, 1,
MPI_DOUBLE, 0,
MPI_COMM_WORLD);
if ( rank == 0 ) {
/* noeud maitre: sommer
* les sous-sommes pour obtenir
* la valeur de pi */
sum = 0;
for (i=0 ; i < size ; i++ ) {
xprintf("somme=%.12f pour le "
"noeud %d¥n",
sum_buf[i], i);
sum += sum_buf[i];
}
xprintf("pi=%.12f¥n", sum);
}
xexit(0);
/* Nous n'arrivons jamais ici,
* mais la ligne suivante est
* necessaire pour eviter un warning
*/
return 0;
}
```