

TP1 – CALCUL DISTRIBUE DE LA VALEUR DE PI

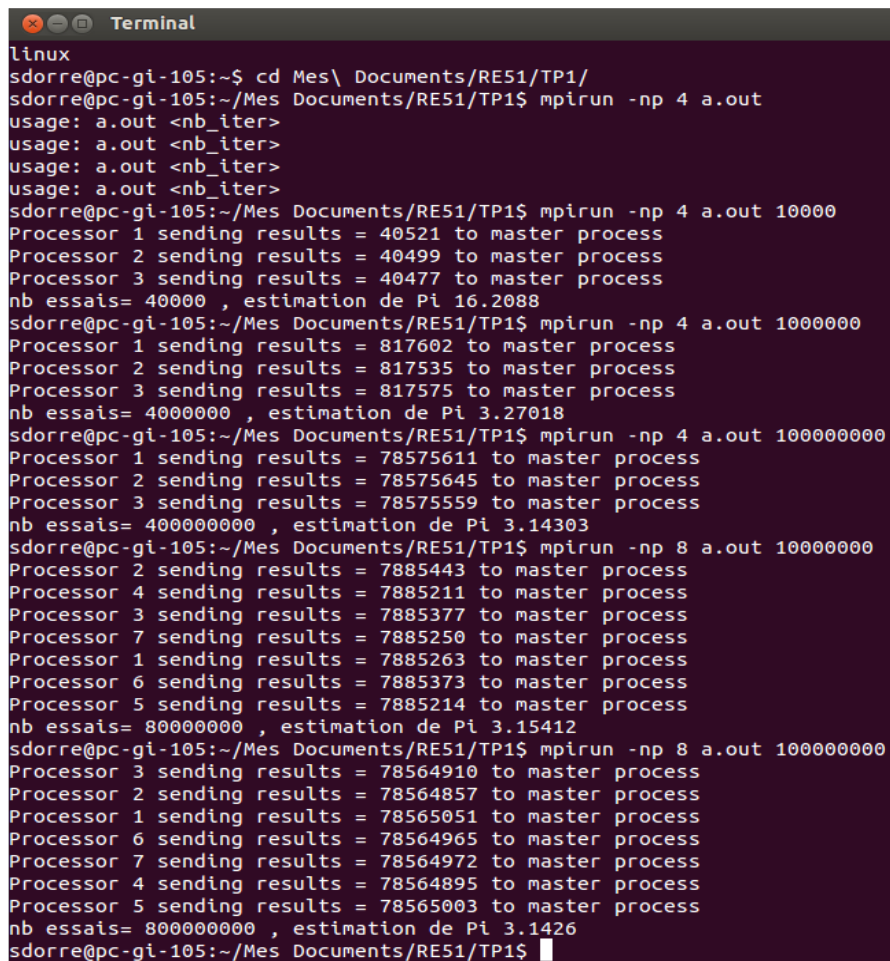
Exercice 1

Le travail consiste à adapter le code fourni dans le listing 1 afin de le rendre exécutable dans un environnement distribué. Le code doit être réalisé en C à l'aide de la librairie OpenMPI.

Réponse :

Le programme réalisé est joint avec ce document. Il s'agit du fichier « exo1.c ». La distribution du calcul permet de multiplier le nombre d'itérations demandées. En effet, chaque processus va effectuer le même nombre d'itérations. Quand chaque processus a fini ses calcul, il renvoie le résultat au processus principal qui va se charger de regrouper les valeurs et calculer la valeur de PI.

Voici le résultat après plusieurs exécutions du programme :



```
linux
sdorre@pc-gi-105:~$ cd Mes\ Documents\RE51\TP1/
sdorre@pc-gi-105:~/Mes Documents/RE51/TP1$ mpirun -np 4 a.out
usage: a.out <nb_iter>
usage: a.out <nb_iter>
usage: a.out <nb_iter>
usage: a.out <nb_iter>
sdorre@pc-gi-105:~/Mes Documents/RE51/TP1$ mpirun -np 4 a.out 10000
Processor 1 sending results = 40521 to master process
Processor 2 sending results = 40499 to master process
Processor 3 sending results = 40477 to master process
nb essais= 40000 , estimation de Pi 16.2088
sdorre@pc-gi-105:~/Mes Documents/RE51/TP1$ mpirun -np 4 a.out 1000000
Processor 1 sending results = 817602 to master process
Processor 2 sending results = 817535 to master process
Processor 3 sending results = 817575 to master process
nb essais= 4000000 , estimation de Pi 3.27018
sdorre@pc-gi-105:~/Mes Documents/RE51/TP1$ mpirun -np 4 a.out 100000000
Processor 1 sending results = 78575611 to master process
Processor 2 sending results = 78575645 to master process
Processor 3 sending results = 78575559 to master process
nb essais= 400000000 , estimation de Pi 3.14303
sdorre@pc-gi-105:~/Mes Documents/RE51/TP1$ mpirun -np 8 a.out 100000000
Processor 2 sending results = 7885443 to master process
Processor 4 sending results = 7885211 to master process
Processor 3 sending results = 7885377 to master process
Processor 7 sending results = 7885250 to master process
Processor 1 sending results = 7885263 to master process
Processor 6 sending results = 7885373 to master process
Processor 5 sending results = 7885214 to master process
nb essais= 800000000 , estimation de Pi 3.15412
sdorre@pc-gi-105:~/Mes Documents/RE51/TP1$ mpirun -np 8 a.out 1000000000
Processor 3 sending results = 78564910 to master process
Processor 2 sending results = 78564857 to master process
Processor 1 sending results = 78565051 to master process
Processor 6 sending results = 78564965 to master process
Processor 7 sending results = 78564972 to master process
Processor 4 sending results = 78564895 to master process
Processor 5 sending results = 78565003 to master process
nb essais= 8000000000 , estimation de Pi 3.1426
sdorre@pc-gi-105:~/Mes Documents/RE51/TP1$
```

On peut remarquer que le plus le nombre d'itérations est élevé plus la valeur de PI est exacte. Mais cela requiert un très grand nombre d'itérations. Le calcul devient alors très long (plusieurs dizaines de secondes).

Exercice 2

Le travail consiste à réaliser un programme en C, à l'aide de la librairie OpenMPI et utilisant la formule de Ramanujan présentée précédemment pour estimer la valeur de π .

Réponse :

Le programme est présent dans le fichier nommé « `exo2_queue.c` ». Dans ce programme, le calcul est distribué par le processus principal à tous les processus fils. Pour se faire, une queue a été utilisée. Au début de l'exécution, le processus principal remplit la queue avec les éléments à calculer. Quand ce dernier a fini, il attend les réponses des autres processus.

Les autres processus vont calculer chaque élément de la somme (dans la formule de Ramanujan) et renvoyer le résultat au processus principal. Ils interrogent la queue et, tant qu'il y a des éléments, effectuent les calculs. Quand la queue est vide, les processus se terminent. Le processus principal peut alors calculer la valeur de π .

Voici le résultat d'une exécution : (d'autres sont disponibles dans les fichiers de log fournis avec le dossier.

```
# mpirun -np 4 ramanujan 15 > ramanujan0.log

send first work number:0 to process 1
send first work number:1 to process 2
send first work number:2 to process 3
Child 1: receive value:0
Child 2: receive value:1
Child 2: send my computation 0.000027
Master: received result_n = 0.000027 and send value : 3 to child 2
Child 2: receive value:3
Child 2: send my computation 0.000000
Master: received result_n = 0.000000 and send value : 4 to child 2
Child 2: receive value:4
Child 2: send my computation 0.000000
Master: received result_n = 0.000000 and send value : 5 to child 2
Child 2: receive value:5
Child 2: send my computation -0.000000
Master: received result_n = -0.000000 and send value : 6 to child 2
Child 2: receive value:6
Child 2: send my computation -0.000000
Master: received result_n = -0.000000 and send value : 7 to child 2
Child 2: receive value:7
Child 2: send my computation -0.000000
Master: received result_n = -0.000000 and send value : 8 to child 2
Master: received result_n = 1103.000000 and send value : 9 to child 1
Child 2: receive value:8
Child 2: send my computation -0.000000
Master: received result_n = -0.000000 and send value : 10 to child 2
Child 2: receive value:10
Child 2: send my computation 0.000000
Master: received result_n = 0.000000 and send value : 11 to child 2
Child 2: receive value:11
Child 2: send my computation 0.000000
Master: received result_n = 0.000000 and send value : 12 to child 2
```

Child 2: receive value:12
Child 2: send my computation 0.000000
Child 1: send my computation 1103.000000
Master: received result_n = 0.000000 and send value : 13 to child 2
Child 2: receive value:13
Child 2: send my computation 0.000000
Master: received result_n = 0.000000 and send value : 14 to child 2

Queue empty
end while - count = 1103.000027
Child 2: receive value:14
Child 2: send my computation 0.000000
Child 1: receive value:9
Child 1: send my computation 0.000000
Child 3: receive value:2
Master: I send something
Master: I send something
Master: I send something
Child 3: send my computation 0.000000
pi = 3.141593
process 1 : I stop
process 2 : I stop
process 3 : I stop

Cette seconde méthode est clairement plus efficace puisqu'il ne faut que quelques éléments de la somme de la formule pour obtenir la valeur correcte de PI.