MPI Java (http://mpj-express.org)

Exercice 1 : Installez MPI et MPJ-express
http://mpj-express.org/docs/guides/linuxguide.pdf
JavaDoc http://mpj-express.org/docs/javadocs/index.html

puis testez le programme suivant :

```
import mpi.*;
 2
 3 public class HelloWorld {
 4
 5   public static void main(String args[]) throws Exception {
 6
 7     MPI.Init(args);
 8     int size = MPI.COMM_WORLD.Size();
 9     int rank = MPI.COMM_WORLD.Rank();
10
11     System.out.println("I am process <"+rank+">"+"of "+size+" processors");
12
13     MPI.Finalize();
14   }
15 }
```

On compile avec javac -cp .:$MPJ_HOME/lib/mpj.jar HelloWorld.java puis on exécute avec mpjrun.sh -np 4 HelloWorld ; C'est la version multi-core. Pour un cluster, il faut plusieurs machines sur un réseau via ssh (n'oubliez pas le fichier « machine » avec les adresses IP pour mpi)
Recopiez, analysez et testez le programme suivant (reception de buffers par le processeurs 0):

```
1  import mpi.*;
2
3  public class ToyExample {
4
5      public static void main(String[] args) throws Exception {
6
7          MPI.Init(args); int rank = MPI.COMM_WORLD.Rank(); int size = MPI.COMM_WORLD.Size() ;
8          int unitSize=4, tag=100, master=0;
9
10         if(rank == master) {  /* master */
11
12           int sendbuf[] = new int[unitSize*(size-1)];
13
14           for(int i=1; i<size; i++)
15             MPI.COMM_WORLD.Send(sendbuf, (i-1)*unitSize, unitSize, MPI.INT, i, tag);
16
17           for(int i=1; i<size; i++)
18             MPI.COMM_WORLD.Recv(sendbuf, (i-1)*unitSize, unitSize, MPI.INT, i, tag);
19
20
21           for(int i=0 ; i<unitSize*(size-1) ; i++)
22             System.out.print(sendbuf[i]+" ");
23
24         } else { /* worker */
25
26           int recvbuf[] = new int[uniSiz];
27           MPI.COMM_WORLD.Recv(recvbuf, 0, unitSize, MPI.INT, master, tag);
28
29           for(int i=0 ; i<unitSize; i++) recvbuf[i] = rank;  /* computation loop */
30
31           MPI.COMM_WORLD.Send(recvbuf, 0, unitSize, MPI.INT, master, tag);
32         }
33
34         MPI.Finalize();
35     }
36 }
```

Avec mpjrun.sh -np 5 ToyExample, vous devriez avoir
1 1 1 1 2 2 2 2 3 3 3 3 4 4 4 4

Attention, si pour -np vous mettez une valeur plus grande que votre nombre de coeurs, plusieurs processus tourneront par cœurs et cela dégrade ÉNORMÉMENT les performances !

Exercice 2 : Le « inner product » (des fois appelé « dot product ») consiste, pour 2 vecteurs V={V_1,...V_n} et W={W_1, … W_n} à calculer la somme total des V_i*W_i. On suppose que chaque processeur possède une sous partie de taille n/p de chacun des 2 vecteurs. Alors les multiplications peuvent être effectuées de manière asynchrone ; puis il faut faire la somme total (fold) ; Implantez une version directe ou une des versions logarithmiques vues en cours (on supposera un nombre puissance de 2 de processeurs).

Exercice 3 : Implantez le tri (de int) par échantillonnage vu en cours avec les primitives asynchrones… puis avec les primitives collectives et comparez les codes ! Puis testez les performances avec des tableaux d'entier aléatoires et avec p=2,3,4 coeurs puis en augmentant la taille des tableaux.
Pour les operations collectives, voir :
http://mpj-express.org/docs/guides/listofmpjmethods.pdf
http://mpj-express.org/docs/javadocs/index.html

Bonus, pour les courageux, implantez la multiplication de matrice la MapReduce proposé ici :
http://www.mathcs.emory.edu/~cheung/Courses/554/Syllabus/9-parallel/matrix-mult.html