Práctica 9: MPI-2

Ejercicio 1: comunicación monolateral

El programa *RMA_TODO.py* (a completar) utiliza comunicación monolateral de modo que el proceso 0 comparte su memoria para que el resto de los procesos escriban en ella.

- Los procesos 1 en adelante generan un vector de 1000 números aleatorios que siguen una distribución normal de medias [0, 0.9, -1.2] y varianzas unidad.
- Para especificar la posición en la que se escribe en el objeto ventana, se puede usar el argumento *target* en la operación *Put()*. Por ejemplo, la siguiente línea escribe los datos *data* en la posición *pos* de la ventana aportada por el proceso de rango 0

```
win.Put(data, target_rank=0, target=pos)
```

Ejemplo de funcionamiento:

```
(mpi) $ mpirun -np 4 ./RMA_sol.py
P1: 1.363151 0.826877
P2: -0.301050 0.876993
P3: -1.200500 -0.950824
P0 vec1: 1.363151 0.826877
P0 vec2: -0.301050 0.876993
P0 vec3: -1.200500 -0.950824
(mpi) $
```

Ejercicio 2: lectura paralela

En el fichero data.bin (creado por el programa *RMA_sol.py* completado en el ejercicio 1) se almacenan secuencialmente 3 vectores de 1000 coeficientes cada uno. Se trata de implementar un programa MPI que lleve a cabo las siguientes tareas:

- Deberán lanzarse tres procesos
- Cada proceso leerá del fichero un vector de 1000 elementos. La lectura se hará en paralelo
- Cada proceso mostrará la media del vector de 1000 elementos que ha leído

OJO: el tipo por defecto de los vectores en numpy se corresponde con el tipo *MPI.DOUBLE* de MPI

Ejemplo de funcionamiento:

```
(mpi) $ mpirun -np 3 ./ParallelFile.py
Vector 0/3 - 1000 elementos: media = -0.016
Vector 1/3 - 1000 elementos: media = 0.876
Vector 2/3 - 1000 elementos: media = -1.201
(mpi) $
```

Ejercicio 3: N-body problem (3)

Seguimos con la implementación de un sistema de objetos en el vacío que interaccionan gravitatoriamente dos a dos. Se plantea ahora el uso de funcionalidades de MPI-2:

- **Acceso paralelo a ficheros**: en la práctica anterior, el proceso 0 leía el fichero y el reparto se hacía usando técnicas de comunicación colectiva. Esta parte debe ser sustituída por un acceso a fichero compartido, teniendo en cuenta los siguientes puntos:
  - El acceso a ficheros compartidos se realiza siempre desde y hacia ficheros en formato binario, por lo que el fichero data.txt debe ser convertido antes de poder ser utilizado. Debe implementarse un script en python para hacer esta conversión
  - El proceso 0 debe abrir el fichero de datos (binario), y leer el primer entero del mismo, (el número de objetos) y a continuación cerrar el fichero. Se comprobará el número de procesos y de ficheros, y a continuación se hará la lectura compartida (definiendo la vista del fichero y leyendo los datos en paralelo).
- **Memoria compartida**: como se ha podido comprobar en las versiones anteriores, el orden en que se muestran en pantalla los resultados es aleatorio. Resolver este problema es una mejora bastante necesaria. Para ello, las líneas que muestran las posiciones de los objetos actualizadas serán ejecutadas sólo por el proceso 0. Este proceso reservará espacio en memoria para que el resto de los procesos almacenen en él las posiciones de los objetos, y una vez hecha la escritura, el objeto 0 mostrará las posiciones de manera ordenada. Observar que no es necesario realizar la comunicación monolateral en cada iteración, sino cada NUM_ITER_SHOW iteraciones