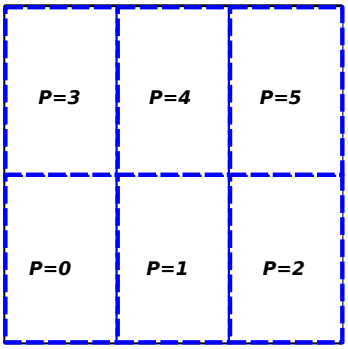
Curso HPC en MC. goo.gl/m6qbhV
TPL2. Trabajo Práctico de Laboratorio. [2018-05-29]

PASSWD PARA EL ZIP: VPGQ CQ4G QWJV

Ejercicios

[Ej. 1] **[scatter-prtcls]** Dada un conjunto de partículas en 2D identificadas por sus posiciones $(x, y)_j$ en el cuadrado unitario $[0, 1] \times [0, 1]$ dividirlas por procesadores de manera lo más equitativa posible. Por ejemplo si **`size=6`** entonces podemos dividirlas en un arreglo de 6 rectángulos de $(1/3) \times (1/2)$ (ver figura).



Nota: Se provee una función **`int getproc(double *x,int size);`** que devuelve el número de procesador dada las coordenadas **`x`** de la partícula.

Consigna: Escribir una función **`void scatter_prtcls(vector<double> &xprtcls,MPI_Comm comm);`** que realiza las comunicaciones necesarias para que en el **`xprtcls`** de cada procesador queden las partículas que corresponden a su caja. Las partículas están en un arreglo **`xprtcls`** de **`ndim*nprtcls`** almacenadas por partícula, es decir **`x0,y0,x1,y1,...`**

Ejemplo: Si **`size=2`** entonces tenemos dos cajas **`0<=x<0.5`** en **`myrank=0`**y **`0.5<=x<=1.0`** en **`myrank=1`**. Si las partículas son inicialmente

`xprtcls: P0={[0.7,0.3],[0.2,0.4]}, P1={[0.1,0.3],[0.6,0.7],[0.3,0.8]}`

Entonces después de **`scatter_prtcls`** debe quedar

`xprtcls: P0={[0.2,0.4],[0.1,0.3],[0.3,0.8]}, P1={[0.7,0.3],[0.6,0.7],}`

Nota: Notar que el número de partículas inicial puede no ser igual entre todos los procesadores (en este caso es [2,3]) y tampoco el número final (en este caso [3,2]). Además en un dado procesador el número de partículas inicial puede no ser igual al final.

Ayuda:

- Aplicar la función **`getproc()`** a cada una de las partículas para contar las partículas que deben ir a cada uno de los otros procesadores, es decir el **`scounts`**.
- Reordenar las partículas en un contenedor auxiliar **`sbuff`** del mismo tamaño que el **`xprtcls`**, de manera de que queden primero las partículas que van el **`P0`** despues al **`P1`** etc...
- Hacer un **`Alltoall`** de los **`scounts`** a **`rcounts`**.

- Calcular la cantidad total de partículas que terminarán en este procesador sumando los **rcounts**.
- Resizear apropiadamente el **xprtcls** para que almacene la nueva cantidad de partículas.
- Calcular los **sdispls** y **rdispls** como **cumsum** de los **counts**.
- Hacer al **Alltoallv** del **sbuff** al **xprtcls**.

Verificación: Hay una función **void scatter_prtcls_test(int kase);**:

- Con **kase=0** y **numprocs=2** corre el caso de 5 partículas descripto más arriba.
- Con **kase=1** y **numprocs=6** debe imprimir:

```
Input prtcls:
[0] mean coords 0.566731 0.645338
[1] mean coords 0.527512 0.649395
[2] mean coords 0.538865 0.577784
[3] mean coords 0.560657 0.642877
[4] mean coords 0.583882 0.396645
[5] mean coords 0.489545 0.408175
Output user:
[0] mean coords 0.131111 0.142648
[1] mean coords 0.538060 0.243364
[2] mean coords 0.800246 0.321294
[3] mean coords 0.200268 0.745429
[4] mean coords 0.469143 0.699672
[5] mean coords 0.777354 0.738373
```

[Ej. 2] **[mech-props]** Dada un conjunto de partículas en 2D $\mathbf{x}_j$ con massas m_j y velocidades $\mathbf{v}_j$ calcular las siguientes propiedades mecánicas del conjunto,

$$\begin{aligned} M &= \sum m_j, && \text{masa total,} \\ \mathbf{x}_{CG} &= M^{-1} \sum m_j \mathbf{x}_j, && \text{centro de masa,} \\ \mathbf{v}_{CG} &= M^{-1} \sum m_j \mathbf{v}_j, && \text{velocidad media,} \\ \mathbf{I}_{kl} &= \sum m_j (\mathbf{x}_j - \mathbf{x}_{CG})_k (\mathbf{x}_j - \mathbf{x}_{CG})_l, && \text{tensor de inercia,} \end{aligned} \tag{1}$$

Consigna: Escribir una función

```
void mech_props(MatrixXd &xp,VectorXd &masses,MatrixXd &vels,
                double &mass,VectorXd &xcg,
                VectorXd &vmean, MatrixXd &I,MPI_Comm comm);
```

que realiza la tarea indicada.

Cada procesador recibe un número diferente de partículas **np**, las matrices **xp** y **vels** son de **np*ndim**, las masas **masses** es un vector de longitud **np**. Los argumentos de salida son la masa total **mass** y los vectores **xcg** y **vmean** de long 3, y el tensor de inercia **I(ndim,ndim)**.

Ayuda: Computar las contribuciones locales de las sumatorias, reducir, y calcular los valores. Por ejemplo en el caso del **xcg** debe primero calcularse la suma de masas local, la suma de los **xj*mj**, **reducir**, y después hacer el cociente.

Verificación: Hay una función de verificación `void mech_props_test(int kase);`. Se generan 50 partículas aleatorias dependientes de **kase**. El resultado (masa, centro de gravedad...) debe ser independiente de el número de procesadores, y debe dar.

- Para **kase=0**:

```
mass 25.1776
xcg 0.0286166 0.0318712
vmean  -0.0484778 -0.00197599
I  2.11819 0.506483 0.506483  1.99177
```

- Para **kase=1**.

```
mass 26.1616
xcg  0.0244786 -0.0641701
vmean 0.0767616 0.0365774
I    2.0499 -0.500295 -0.500295   1.96422
```

Si hay problemas para compilar con el Makefile reemplazar las lineas

```
CPP := aedtools/util_btree.cpp aedtools/util.cpp aedtools/util_tree.cpp
..
program.bin: force
	$(CXX) -I. -I$(EIGEN) -std=c++0x -g3 -O0 -o $@ program.cpp $(CPP)
```