Curso HPC en MC. goo.gl/m6qbhV
TPL1. Trabajo Práctico de Laboratorio. [2018-04-24]

PASSWD PARA EL ZIP: 7HRW HMG3 HM6V8

Ejercicios

[Ej. 1] **[transpose]** En una corrida en paralelo de tamaño **nproc** cada procesador tiene una serie de **nproc** arreglos de tamaño **N** (es decir **vector<vector<int>> sbuff**) y debe enviar el arreglo **sbuff[j]** al procesador **j**. Para ello cada procesador tiene una serie de **nproc** arreglos de recepción **vector<vector<int>> rbuff** de manera que en **rbuff[k]** recibe el arreglo enviado por el procesador **k**.

Consigna: Escribir una función
void transpose0(vector<vector<int>> &sbuff,vector<vector<int>> &rbuff); que realiza la tarea indicada.

Ejemplo: **nproc=3, N=4**

```
Send buffers:
[rank=0] sb[0]=(100,101,102,103) sb[1]=(104,105,106,107) sb[2]=(108,109,110,111)
[rank=1] sb[0]=(200,201,202,203) sb[1]=(204,205,206,207) sb[2]=(208,209,210,211)
[rank=2] sb[0]=(300,301,302,303) sb[1]=(304,305,306,307) sb[2]=(308,309,310,311)

Recv buffers (after transpose):
[rank=0] rb[0]=(100,101,102,103) rb[1]=(200,201,202,203) rb[2]=(300,301,302,303)
[rank=1] rb[0]=(104,105,106,107) rb[1]=(204,205,206,207) rb[2]=(304,305,306,307)
[rank=2] rb[0]=(108,109,110,111) rb[1]=(208,209,210,211) rb[2]=(308,309,310,311)
```

Ayuda (versión 1): En principio uno piensa en el siguiente algoritmo

```
for (int j=0; j<size; j++) {
  Send(sbuff[j],...,j...);
  Recv(rbuff[j],...,j...);
}
```

El problema es que esto produciría *dead-lock* ya que todos quieren primero enviar y después recibir. Una posibilidad es invertir los Send/Recv entre los procesadores **myrank,j** dependiendo si **myrank<j** o **myrank>j**,

```
for (int j=0; j<size; j++) {
  if (myrank<j) {
     Send(sbuff[j],...,j...);
     Recv(rbuff[j],...,j...);
  } else if (myrank>j) {
     Recv(rbuff[j],...,j...);
     Send(sbuff[j],...,j...);
```

```
  } else { // myrank==j
    // Direct copy of sbuff[j] to rbuff[j]
  }
}
```

Notar que en el caso de **myrank==j** estamos comunicando dentro del mismo procesador, entonces no hace falta realizar comunicación si no que directamente copiamos el **sbuff** to el **rbuff**. De hecho, al hacer el envío a si mismo puede producir dead-lock.

Nota: **sbuff[j]** es en realidad un **vector<int>** de manera que al recibir y enviar debe usarse el método **data()** para obtener el puntero al buffer interno (de tipo **int***).

Nota: Para obtener el tamaño **N** de los buffers basta con tomar el tamaño de cualquiera de ellos (por ejemplo **sbuff[0]**) ya que son todos iguales.

Ayuda (versión 2): Tal vez una posibilidad más simple es usar **MPI_Sendrecv()**

```
for (int j=0; j<size; j++)
   Sendrecv(sbuff[j],...,j..., rbuff[j],...,j...);
```

Nota: En mi opinión esta es la versión más simple.

Ayuda (versión 3) (OJO AVANZADA): Usar comunicación no bloqueante

```
MPI_Request requestv[size];
for (int j=0; j<size; j++) {
  Isend(sbuff[j],...,j...,&requestv[j]);
  Recv(rbuff[j],...,j...);
}
// Esperar hasta que todos los envios se concreten
for (int j=0; j<size; j++)
   MPI_Wait(&requestv[j],MPI_STATUS_IGNORE);
```

[Ej. 2] **[parfind]** Escribir una función
void parfind(vector<int> &v,int x,int &procfirst,int &kfirst,int &count); tal que recibe un vector de enteros **v** (de longitud diferente en cada proceso) y un entero **x** y debe determinar si **x** está en alguno de los **v** (en los diferentes procesadores). Concretamente debe devolver en **count** el número total de ocurrencias de **x** en todos los procs, y en **procfirst** y **kfirst** la primera ocurrencia de **x** (si la hay) es decir el primer procesador en el que ocurre y la primera posición **kfirst** dentro del mismo. Si **x** no aparece en ninguno de los **v** entonces debe retornar **count=0**, **procfirst=size**, **kfirst=-1**.

Ejemplos:

```
Si: size=3, rank=0: v=[1,2,3,4,5], rank=1: v=[4,5,6,7,8], rank=2: v=[9,10],
x=1  => count=1, procfirst=0, kfirst=0
x=4  => count=2, procfirst=0, kfirst=3
x=20 => count=0, procfirst=3, kfirst=-1
x=9  => count=1, procfirst=2, kfirst=0
```

Ayuda:

- En cada procesador recorrer los elementos contando cuantas veces aparece **v** con un contador **countj** y retener (si existe) la primera posición **kfirst**.
- Hacer una **Allreduce** the **countj** a **count** por la suma.
- En cada procesador definir una variable **procfirstj** que es **myrank** si **x** aparece en este procesador, y si no **size**.
- Hacer una **Allreduce** the **procfirstj** a **procfirst** por el mínimo.
- En cada procesador chequear el valor de **procfirst**, si es **procfirst<size** quiere decir que **x** está en alguno y el primero es **procfirst**.
- Si **x** fue encontrado hacer un **Bcast** de **kfirst** desde el procesador correspondiente **procfirst**.
- Si **x** NO fue encontrado dejar **kfirst** en -1.

Instrucciones generales

- El examen consiste en que escriban las funciones descriptas más abajo; impleméntandolas en C++ de tal forma que el código que escriban **compile y corra correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.
- Pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.
- Se incluye un template llamado **program.cpp**. En principio sólo tienen que escribir el cuerpo de las funciones pedidas.
- Hay una función de evaluación, por ejemplo si **f** es la función a evaluar tenemos

  ```
  ev.eval1(f,vrbs);
  ```

  ev.eval1(f,vrbs); toma una serie de casos de prueba de entrada, le aplica la función del usuario **f** y compara la salida del usuario (**user**) con respecto a la esperada (**ref**). Si la verbosidad (el argumento **vrbs**) se pone en uno, entonces la función evaluadora reporta por consola los datos de entrada, la salida de la función de usuario y la salida esperada