Seminario 2

Autor: Antonio Jesús Heredia

Ejercicio 1

Para el ejercicio 1 he introducido este código. Lo que primero miro es si el proceso tiene un id par o impar. Una vez que miro eso miro si es el primero de ese "grupo". Es decir el el proceso 1 y 0 enviara a sus correspondientes procesos pero no recibirá. El resto de procesos recibirán y si quedan procesos a los que enviar, enviara también.

```
    if (rank % 2 == 0)
    {
        if (rank != 0)
        {
            MPI_Recv(&value, 1, MPI_INT, rank - 2, 0, MPI_COMM_WORLD,
&status);

            if (size > rank + 2)
                MPI_Send(&value, 1, MPI_INT, rank + 2, 0, MPI_COMM_WORLD);
        }
        else
            MPI_Send(&value, 1, MPI_INT, rank + 2, 0, MPI_COMM_WORLD);
    }
    else
    {
        if (rank != 1)
        {
            MPI_Recv(&value, 1, MPI_INT, rank - 2, 0, MPI_COMM_WORLD,
&status);

            if (size > rank + 2)
                MPI_Send(&value, 1, MPI_INT, rank + 2, 0, MPI_COMM_WORLD);
        }
        else
            MPI_Send(&value, 1, MPI_INT, rank + 2, 0, MPI_COMM_WORLD);
    }
```

Ejercicio 2

En este ejercicio lo que me encargo es de ver cual sera el inicio y el final del intervalo en el que se calculara pi. Para ello cada proceso calculara un intervalo de tamaño Bsize. Donde Bsize = ceil(n / numprocs). El ultimo proceso calculara hasta como mucho n, de hay el "if" en el calculo de iend, para no pasarnos. Como empezamos en el 1, tenremos que llegar al <= del iend.

```
    int istart = myid*Bsize+1;
    int iend;
```

```
    //Si en este caso  (el ultimo proceso) hay mas que lo que indicamos en
lo argumentos
    //nos quedamos como final el numero n introducido
    if( myid * Bsize + Bsize > n)
        iend = n;
    else
        iend =  myid * Bsize + Bsize;

    for (i = istart; i <=iend; i++)
    {

        x = h * ((double)i - 0.5);
        sum += 4.0 / (1.0 + x * x);
    }
```

Ejercicio 3

Al igual que en el ejercicio anterior la dificultad de este ejercicio radica en calcular, el inicio y el final del vector que tiene que calcular cada proceso. Lo realizo de la misma forma que en el ejercicio anterior. El for se realiza desde istart hasta iend, pero necesito otra variable a la hora de guardarlo en **VectorBlocal**, ya que se empieza desde el 0 hasta Bsize.

```
    int Bsize = ceil(tama / size);
    //Calculo el inicio y el final del intervalo para este proceso
    int istart = rank * Bsize;
    int iend;
    //Si en este caso  (el ultimo proceso) hay mas que lo que indicamos en
lo argumentos
    //nos quedamos como final el numero n introducido

    if (rank * Bsize + Bsize > tama)
        iend = tama;
    else
        iend = rank * Bsize + Bsize;
    int x = 0;
    for (long i = istart; i < iend; ++i){
        VectorBLocal[x] = (i + 1) * 10;
        x++;
    }
```

Ejercicio 4

Lo primero que creo es un vector para repartirlo entre los numeros impares y una variable para que cada proceso guarde lo que recibe del scatter. Para inicializar el vector lo primero que realizo es mirar si el **color** es 1, esto quiere decir que son los comunicadores impares, luego mirare si el rank de ese proceso es el 0, esto quiere decir que es el proceso 1 del MPI_COMM_WORLD. Despues para todos los color=1 (es decir todos los impares) realizo e scatter guardando en la variable recibido. Para probar que funciona simplemente lo muestro por pantalla.

```
    //Creo el vector
    vector<int> vec;
    vec.resize(size_nuevo, 0);
    int recibido = -1;
    //Si es el proceso 0 de los impares inicializo el vector
    if(color == 1){
        if(rank_nuevo == 0){
            for(int i = 0; i < size_nuevo; i++)
                vec[i] = i;
        }
        //Si es cualquier impar realizado en scatter
        MPI_Scatter(&vec[0],      // Valores a compartir
                1,      // Cantidad que se envia a cada proceso
                MPI_INT,         // Tipo del dato que se enviara
                &recibido, // Variable donde recibir los datos
                1,      // Cantidad que recibe cada proceso
                MPI_INT,         // Tipo del dato que se recibira
                0,                // proceso principal que reparte los
datos
                comm);  // Comunicador (En este caso, el global)
    }
```