Nombre y apellido: Enric Gil Gallen

Nombre y apellido: Victor Granados Segara

Tiempo: 0:30

1 Implementa un programa con MPI en el que cuatro procesos escriban:

`Hola, soy el proceso <i>!`

siendo `i` el identificador numérico de cada proceso.

```
#include <stdio.h>
#include "mpi.h"

void main(int argc, char * argv[]){
    MPI_Status s;
    int numProcs, miId;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numProcs);
    MPI_Comm_rank(MPI_COMM_WORLD, &miId);

    printf("Hola, soy el proceso %d de los %d \n",miId, numProcs);

    MPI_Finalize () ;
}
```

2 Realiza las siguientes tareas.

2.1) Escribe un programa en MPI en el que el proceso 0 lea un valor del teclado y lo almacene en la variable `n`. Una vez el proceso 0 haya leído del teclado el valor, todos los procesos deberán imprimir el contenido de la variable `n`. Es decir, cada proceso debe imprimir en una misma línea su identificador y el contenido de la variable `n`, tal y como sigue:

`Proceso <i>con n = <n>`

Escribe a continuación únicamente la parte central del código.

```
#include <stdio.h>
#include "mpi.h"

void main(int argc, char * argv[]){
    MPI_Status s;
    int numProcs, miId, i;
    float num = -1;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numProcs);
    MPI_Comm_rank(MPI_COMM_WORLD, &miId);

    if( miId == 0){
        printf ("Dame numero : ") ;
```

```
        scanf ("%f" ,&num ) ;
    }
    printf("Proceso %d con n = %f \n",miId, num);

    MPI_Finalize () ;
}
```

2.2) ¿Todos los procesos tienen el valor leído por el proceso 0 en sus variables `n`? ¿Por qué?

- No, ya que no existe ningún tipo de sincronización por lo que cada proceso va por su cuenta.

2.3) Modifica el anterior programa para que una vez el proceso 0 haya leído el número, lo envíe él mismo al resto de procesos. Para ello deberá utilizar operaciones de comunicación punto a punto, enviando el contenido de la variable `n` al proceso 1, en primer lugar, luego al proceso 2, y continuando con el resto.

Así, tras esta fase de comunicaciones, todos los procesos deberían tener el valor leído por el proceso 0 en la variable `n`. Finalmente, cada proceso debe imprimir en una misma línea su identificador y el contenido de `n`, tal y como se comentó con anterioridad.

Escribe a continuación únicamente la parte central del código.

```
#include <stdio.h>
#include "mpi.h"

void main(int argc, char * argv[]){
        MPI_Status s;
        int numProcs, miId, i;
        float num = -1;

        MPI_Init(&argc, &argv);
        MPI_Comm_size(MPI_COMM_WORLD, &numProcs);
        MPI_Comm_rank(MPI_COMM_WORLD, &miId);

        if( miId == 0){
        printf ("Dame numero : ") ;
        scanf ("%f" ,&num ) ;
        for (i=1; i < numProcs; i++){
                MPI_Send (&num, 1,MPI_INT,i ,88, MPI_COMM_WORLD);
        }

        } else{
        MPI_Recv(&num, 1,MPI_INT,0 ,88, MPI_COMM_WORLD, &s);
        }

        printf("Proceso %d con n = %f \n",miId, num);

        MPI_Finalize () ;
}
```

2.2) ¿Todos los procesos tienen el valor leído por el proceso 0 en sus variables `n`? ¿Por qué?

Sí, todos tienen el valor leído por el proceso 0 gracias a la función MPI_Send(&num,....)

3 En este ejercicio se va a implementar el algoritmo ping-pong para medir la latencia y el ancho de banda de la red de comunicaciones que interconecta dos procesos.

3.1) Introduce en el programa anterior, el código que permite que el proceso 0 envíe el vector vecArgs al resto de procesos.
Fíjate que estas líneas se deben insertar a continuación de la línea marcada con "(A)".
Escribe a continuación la parte de tu código que realiza tal tarea:

3.2) Introduce en el programa anterior, el código que permite que el proceso 0 envíe `num` mensajes de tamaño `tam` bytes al proceso 1 y que éste devuelva un mensaje de tamaño 0 bytes cuando reciba el último de ellos.
Incluye también las líneas que permite al proceso 0 identificar cuando se inician (`t1`) y finalizan (`t2`) las operaciones de comunicación.
Fíjate que estas líneas se deben insertar a continuación de la línea marcada con "(B)".
Escribe a continuación la parte de tu código que realiza tal tarea:

3.3) Introduce en el programa anterior, el código que permite al proceso 0 medir el coste de cada comunicación, así como la duración media del envío de cada mensaje y el ancho de banda de la comunicación.
Fíjate que estas líneas se deben insertar a continuación de la línea marcada con "(C)".
Escribe a continuación la parte de tu código que realiza tal tarea:

```
// El proceso 0 prepara el vector con las cinco variables.
if( miId == 0 ) {
  vecArgs[ 0 ] = numArgs;
  vecArgs[ 1 ] = numMensajes;
  vecArgs[ 2 ] = minTam;
  vecArgs[ 3 ] = maxTam;
  vecArgs[ 4 ] = incTam;
}

// EJERCICIO 3.1 (A)
// Difusion del vector vecArgs con operaciones punto a punto.
  if( miId == 0 ) {
      int i;
      for (i=1; i < numProcs; i++){
          MPI_Send (vecArgs, 5,MPI_INT,i ,88, MPI_COMM_WORLD);
      }
  } else{
      MPI_Recv(vecArgs, 5,MPI_INT,0 ,88, MPI_COMM_WORLD, &s);
  }

// El resto de procesos inicializan las cinco variables con la
// informacion del vector. El proceso 0 no tiene que hacerlo porque
// ya habia inicializado las variables.
if( miId != 0 ) {
  numArgs     = vecArgs[ 0 ];
  numMensajes = vecArgs[ 1 ];
  minTam      = vecArgs[ 2 ];
```

```
  maxTam      = vecArgs[ 3 ];
  incTam      = vecArgs[ 4 ];
}

// Todos los procesos comprueban el numero de argumentos de entrada.
if( ( numArgs != 3 )&&( numArgs != 5 ) ) {
  if ( miId == 0 ) {
    fprintf( stderr, "\nUso: a.out numMensajes minTam [ maxTam incTam ]\n\n" );
  }
  MPI_Finalize();
  return( -1 );
}

// Imprime los parametros de trabajo.
if( miId == 0 ) {
  printf( "  Numero de procesos:  %5d\n", numProcs );
  printf( "  Numero de mensajes:  %5d\n", numMensajes );
  printf( "  Tamanyo inicial   :  %5d\n", minTam );
  printf( "  Tamanyo final     :  %5d\n", maxTam );
  printf( "  Incremento        :  %5d\n", incTam );
}

// Crea un vector capaz de almacenar el espacio maximo.
if( maxTam != 0 ) {
  ptrWorkspace = ( char * ) malloc( maxTam );
  if( ptrWorkspace == NULL ) {
    if ( miId == 0 ) {
      fprintf( stderr, "\nError en Malloc: Devuelve NULL.\n\n");
    }
    MPI_Finalize();
    return( -1 );
  }
} else {
  ptrWorkspace = NULL;
}

// Imprime cabecera de la tabla.
if ( miId == 0 ) {
  printf( " Comenzando bucle para envio de informacion\n\n" );
  printf( " Tamanyo(bytes)   tiempoTotal(s.)" );
  printf( "  tiempoPorMsg(microsec.)  AnchoBanda(MB/s)\n");
  printf( " ----------------------------------" );
  printf( "----------------------------------------\n");
}

// Sincronizacion de todos los procesos
MPI_Barrier( MPI_COMM_WORLD );

// Bucle para pruebas de tamanyos.
for( tam = minTam; tam <= maxTam; tam += incTam ) {

  // Sincronizacion de todos los procesos
  MPI_Barrier( MPI_COMM_WORLD );

  // Bucle de envio/recepcion de "numMensajes" de tamanyo "tam" y toma de tiempos.
  // ... (B)
```

```
    if(miId == 0){
        t1 = MPI_Wtime();

        for (i = 0; i < numMensajes; i++){
            MPI_Send (ptrWorkspace, tam,MPI_CHAR,1 ,88, MPI_COMM_WORLD);
        }

        MPI_Recv(ptrWorkspace, 0,MPI_CHAR,1 ,88, MPI_COMM_WORLD, &s);

        t2 = MPI_Wtime();
    }
    if(miId == 1){

        for (i = 0; i < numMensajes; i++){

            MPI_Recv(ptrWorkspace, tam ,MPI_CHAR,0 ,88, MPI_COMM_WORLD, &s);
        }

        MPI_Send (ptrWorkspace, 0,MPI_CHAR,0 ,88, MPI_COMM_WORLD);
    }
    // Calculo de prestaciones: tiempoTotal, tiempoPorMensajeEnMicroseg,
    // anchoDeBandaEnMbs.
    // ... (C)
    if(miId == 0){
        tiempoTotal = t2 - t1;
        tiempoPorMensajeEnMicroseg = (tiempoTotal /numMensajes) * 1000000;
        anchoDeBandaEnMbs = tam/tiempoPorMensajeEnMicroseg;
    }
```

3.4) Verifica que el código funciona correctamente incluyendo la cantidad de mensajes a enviar y su tamaño como argumento en la línea de órdenes. Por ejemplo, la orden `mpirun -np 4 a.out 1000 1024` realizará el envío de 1000 mensajes de tamaño 1024 bytes.
Escribe el resultado de esta ejecución:

```
al387320@lynx:~/PcP/e8$ mpirun --oversubscribe -np 4 punt1a.out 1000 1024
Proceso 3  Se ejecuta en: lynx.uji.es
Proceso 1  Se ejecuta en: lynx.uji.es
Proceso 2  Se ejecuta en: lynx.uji.es
Proceso 0  Se ejecuta en: lynx.uji.es
  Numero de procesos:      4
  Numero de mensajes:   1000
  Tamanyo inicial   :   1024
  Tamanyo final     :   1024
  Incremento        :      1
 Comenzando bucle para envio de informacion

 Tamanyo(bytes)   tiempoTotal(s.)  tiempoPorMsg(microsec.)  AnchoBanda(MB/s)
 ------------------------------------------------------------------------
       1024          0.002104             2.104                  486.65
 ------------------------------------------------------------------------
Fin del programa
```

3.5) Verifica que el código funciona incluyendo todos los parámetros: el número de mensajes a enviar, el tamaño mínimo y máximo de los mensajes, así como el incremento en el tamaño del mensaje. Así, la orden `mpirun -np 4 a.out 1000 0 10240 1024` realizará el envío de 1000 mensajes de tamaño 0 (0K), 1000 mensajes de tamaño 1024 (1K), 1000 mensajes de tamaño 2048 (2K), y así sucesivamente hasta enviar 1000 mensajes de tamaño 10240 (10K). **Ejecuta la prueba anterior en patan** y completa la siguiente tabla, calculando el ancho de banda en Megabytes por segundo y redondeando el resultado con dos decimales.

Tamaño	Tiempo por mensaje (microseg.)	Ancho de banda (MB/s)
0		
1024		
2048		
3072		
4096		
5120		
6144		
7168		
8192		
9216		
10240		

Justifica los resultados.

Tamaño	Tiempo por mensaje (microseg)	Ancho de banda (MB / s)
0	2.067	0
1024	9.265	110.52
2048	17.978	113.92
3072	26.924	114.10
4096	35.329	115.94
5120	44.211	115.81
6144	53.316	115.24
7168	61.784	116.02
8192	70.153	116.77
9216	78.987	116.68
10240	87.757	116.69

3.6) ¿Cuál es la latencia de las comunicaciones? ¿Cómo lo has calculado? ¿Cómo influye el tamaño de mensaje en el ancho de banda? ¿Qué valor tomarías como el ancho de banda real?

- El Tº en el que se envía un mensaje hasta que el receptor recibe el primer octeto.
- Tiempo total / Nº de mensajes
- El tamaño del mensaje es directamente proporcional al ancho de banda.
- 2n/t