Arquitectura de Computadores

Grado en Ingeniería Informática

Práctica 2

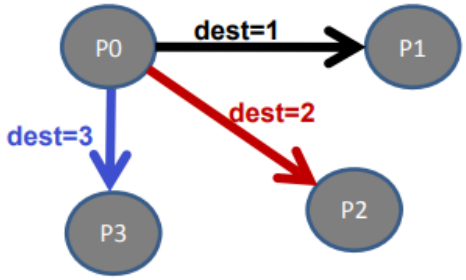
UNIVERSIDAD NEBRIJA

Fundamentos básicos de paralelización: Modelo de comunicación punto a punto

Introducción

MPI define dos tipos de comunicación:

- **Punto a punto** (Fig 1): involucra a 2 procesos (emisor y receptor). Esta comunicación puede ser:
  - Bloqueante: El proceso queda bloqueado hasta que la operación finalice (MPI_Send/MPI_Recv).
  - No bloqueante: el proceso no espera a la finalización de la operación de comunicación y puede continuar solapando cómputo y comunicación. Debe comprobar más adelante si ha terminado la operación (MPI_Isend/MPI_Irecv)
- **Colectiva** (Fig 2): Involucra a todos los procesos de un comunicador.



Ejemplo:

```
MPI_Send (..,dest, ..)
MPI_Recv (..,source, ..)
```

Fig 1 Comunicación punto a punto

Ejemplo:

```
MPI_Bcast (..,MPI_COMM_WORLD, ..)
```

Fig 2 Comunicación colectiva

Funciones send y receive bloqueantes

```
int MPI_Send(void* buf, int count, MPI_Datatype datatype,
             int dest, int tag, MPI_Comm comm);
```

IN	buf	initial address of send buffer
IN	count	number of entries to send
IN	datatype	datatype of each entry
IN	dest	rank of destination
IN	tag	message tag
IN	comm	communicator

```
int MPI_Recv(void* buf, int count, MPI_Datatype datatype,
     int source, int tag, MPI_Comm comm, MPI_Status *status);
```

OUT	buf	initial address of receive buffer
IN	count	max # of entries to receive
IN	datatype	datatype of each entry
IN	source	rank of source
IN	tag	message tag
IN	comm	communicator
OUT	status	return status (inf. acerca del mensaje recibido)

Ejemplo de comunicación punto a punto

```
#include <stdio.h>
#include "mpi.h"
 int main(int argc, char **argv) {
  int rank, count;
  char msg[20];
  MPI_Status status;

  MPI_Init(&argc, &argv);

  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  ...
if (rank==0) {
    printf ("I am master. Sending the message.\n\n");
    strcpy(msg,"Hello World!");
    MPI_Send(msg, 13, MPI_CHAR, 1, 100, MPI_COMM_WORLD);
 }
 else {
    printf ("I am the slave %d. Receiving the message.\n", rank);
    MPI_Recv(msg, 13, MPI_CHAR, 0, 100, MPI_COMM_WORLD, &status);
    printf ("The message is: %s\n", msg);

    MPI_Get_count(&status, MPI_CHAR, &count)
    printf("Recibidos %d caracteres de %d con tag= %d\n",
              count, status.MPI_SOURCE, status.MPI_TAG);
 }
  MPI_Finalize();
}
```

Objetivos

En esta práctica se aprenderá el concepto de comunicación punto a punto entre procesos de MPI.
Los objetivos fijados son los siguientes:

- Compilación de programas MPI.
- Ejecución de programas en varios procesos de forma paralela.
- Estructura de un programa MPI.
  - Iniciar y finalizar el entorno MPI con MPI_Init y MPI_Finalize.

  - Identificador (rango) del proceso con MPI_Comm_rank.
  - Consultar el número de procesos lanzados con MPI_Comm_size.
- Primitivas send y recieve.

Compilación y ejecución de programas MPI

- Compilación: `mpicc codigo_fuente.c -o ejecutable`
- Ejecución: `mpirun –np <number> ejecutable`

Entregables

- Memoria
- Código fuente de los siguientes ejercicios:

Ejercicio 1 (1 punto)

Explicar en qué consiste la comunicación punto a punto, los tipos de primitivas que hay en MPI y las ventajas y desventajas de este tipo de comunicación.

Ejercicio 2 (4 puntos)

Implementar un código donde cada proceso del comunicador inicializa una variable con un **valor aleatorio** (distinto para cada proceso) y se la envía al proceso 0 utilizando las primitivas bloqueantes de comunicación punto a punto. Una vez recibidas todas, el proceso 0 imprime por pantalla los **valores ordenados** por número de proceso.

Ejemplo de la salida por consola con 4 procesos:

Cada proceso imprime: “`Soy el proceso X y he inicializado la variable con el dato X`”
El proceso 0 imprime tantas veces como procesos haya: “`Soy el proceso 0 y he recibido el valor X del proceso X`”

Por último, el proceso 0 imprime una vez: “`Soy el proceso 0 y ya he recibido los datos de los X procesos: datoProceso0 datoProceso1 datoProceso2 datoProceso3 `”

Ejercicio 3 (4 puntos)

Implementar un código donde utilizando comunicación punto a punto tres procesos rebotan continuamente los mensajes entre sí, hasta que deciden detenerse una vez alcanzado límite autoimpuesto.