Universidad de Alcalá
DEPARTAMENTO DE AUTOMÁTICA
ARQUITECTURA Y TECNOLOGÍA DE COMPUTADORES

Laboratorio de Arquitectura e Ingeniería de Computadores
PRÁCTICA V

COMPUTACIÓN PARALELA (PASO DE MENSAJES)

OBJETIVO

El objetivo de esta práctica es introducirse en la programación paralela utilizando el paradigma de Paso de Mensajes. En primer lugar se ejecutarán una serie de programas que realizan tareas muy simples con el fin de familiarizarse con los conceptos expuestos en la teoría. En segundo se propone la realización de un programa que resuelve el cálculo de una integral definida. Para todo ello, se usará el entorno de programación MPI y el lenguaje C.

INTRODUCCIÓN TEÓRICA

El siguiente ejemplo consiste en un programa elemental en entorno MPI. Muestra el uso de comandos básicos de inicialización que permiten compilar y ejecutar un programa en un sistema paralelo.

```
# include "mpi.h"
int main(int argc, char *argv[])
{
int myrank, nprocs;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
  printf("nprocs = %d, My rank = %d\n", nprocs,
           myrank);
  MPI_Finalize();
  return 0;
}
```

Los siguientes ejemplos ilustran el uso de subrutinas de comunicación en MPI. En este primer ejemplo se muestra cómo operaría una instrucción de comunicación global como es el caso de MPI Bcast. Esta subrutina permite difundir el contenido del mensaje de un determinado proceso fuente al resto de procesos.

```
# include "mpi.h"
int main(int argc, char *argv[]) {
  int men[4], nprocs, myrank, i;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
  if (myrank == 0) {
     for(i=0;i<4;i++)
         men[i] = i;
     }
     else {
         for(i=0;i<4;i++)
              men[i] = 0;
     }
  printf("Antes: %d %d %d %d\n", men[0], men[1], men[2],
              men[3]);
  MPI_Bcast(men, 4, MPI_INTEGER, 0, MPI_COMM_WORLD);
  printf("Después: %d %d %d %d\n", men[0], men[1], men[2],
              men[3]);
  MPI_Finalize();
  return 0;
}
```

La subrutina MPI Reduce se encarga de realizar operaciones globales entre todos los procesos, como puede ser una suma global de determinadas variables, y enviar el resultado a un proceso destino fijado.

```
# include "mpi.h"
int main(int argc, char *argv[]) {
  int nprocs, myrank, inicio, final, i;
  float a[1000], sum, tmp;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
  sum = 0.0;
  inicio = 3*myrank + 1;
  final = inicio + 2;
  for (i = inicio; i <= final; i++)
    a[i] = 1.0;
  for (i = inicio; i <= final; i++)
    sum += a[i];
  MPI_Reduce(&sum, &tmp, 1, MPI_REAL, MPI_SUM, 0, MPI_COMM_WORLD);
  if (myrank == 0) printf("Suma: %f\n", tmp);
  MPI_Finalize(); return 0;
}
```

En este último ejemplo se ilustra el uso de subrutinas de comunicación “punto a punto” mediante las ordenes MPI Send y MPI Recv, que permiten enviar y recibir mensajes entre determinados procesos elegidos convenientemente por el programador.

```
# include "mpi.h"
int main(int argc, char *argv[]) {
  int nprocs, myrank, i;
  int valor, nuevo_valor, sum;
  int left, right, tag;
  MPI_Status status;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
  tag = 205;
  right = (myrank + nprocs + 1)%nprocs;
  left = (myrank + nprocs - 1)%nprocs;
  sum =0;
  valor = myrank;
  nuevo_valor = 0;

  for(i=0;i<nprocs;i++){
    MPI_Send(&valor, 1, MPI_INTEGER, right, tag,
    MPI_COMM_WORLD);
    MPI_Recv(&nuevo_valor, 1, MPI_INTEGER, left, tag,
         MPI_COMM_WORLD, &status);
    sum += nuevo_valor;
    valor = nuevo_valor;
    printf("Suma parcial: %d, Proceso: %d\n", sum, myrank);
  }
  if (myrank == 0)
  printf("Resultado final: %d\n", sum);
  MPI_Finalize();
  return 0;
}
```

ACCESO A LOS SISTEMAS

Para la realización de la práctica, debemos conectarnos a la siguiente máquina:

```
172.29.23.181
```

Para conectarse es necesario usar el protocolo ssh.

COMPILACIÓN Y EJECUCIÓN

La compilación de un programa C se realizará de la siguiente manera:

```
mpicc -o ejemploX ejemploX.c -lm
```

Una vez compilado, el ejecutable correspondiente podrá ser ejecutado en paralelo en un número arbitrario de sistemas, en principio cuantos estén disponibles. Previamente deberán ser inicializados una serie de procesos auxiliares, uno en cada sistema de los que se vayan a utilizar. Para ello, crearemos (si no existe ya) un fichero de texto con el nombre mpd.hosts en el directorio propio, en donde figurará el nombre de cada uno de los sistemas que se desee utilizar.

En nuestro:

```
172.29.23.181:32
```

Una vez creado el fichero se ejecutará el comando:

```
mpdboot -n 1 -f mpd.hosts -r ssh
```

Esta operación sólo es necesario realizarla una vez, y arranca un conjunto de procesos auxiliares en las máquinas que entran en el anillo MPI. Hecho esto, se puede proceder a ejecutar cualquier programa MPI. Podemos comprobar qué máquinas integran el cluster mediante el comando:

```
mpdtrace
```

Para la ejecución del programa en paralelo una vez compilado, la sintaxis correspondiente es la siguiente:

```
mpiexec -np <n> ejemploX
```

donde <n> es el número de procesos paralelos que se desee utilizar para la ejecución de ejemploX. Por último, para terminar los procesos auxiliares (que, de otro modo, seguirían en ejecución aunque finalicemos nuestra sesión interactiva), debemos usar, antes de despedirnos, el siguiente comando:

```
mpdallexit
```

DESARROLLO DE LA PRÁCTICA

Desarrollar un programa en MPI que calcule numéricamente la siguiente integral definida:

$$\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} e^{-\left(x^2+y^2\right)/2}\,dx\,dy \approx \int_{-x_N}^{+x_N}\int_{-y_M}^{+y_M} e^{-\left(x^2+y^2\right)/2}\,dx\,dy \approx \sum_{i=0}^{N}\sum_{j=0}^{M} h^2 e^{-\left(x^2+y^2\right)/2}$$

$$x_i = -x_N + ih, \qquad i = 0,...,N.$$
$$y_i = -y_M + jh, \qquad j = 0,...,M.$$

Sabiendo que el valor exacto de la integral es 2π, determinar el error cometido en la resolución numérica para diversos valores de N y M. Estudiar también la ganancia (speed-up) en función del número de procesadores disponibles.