PRÁCTICA ARQUITECTURAS AVANZADAS

DISTRIBUCIÓN DEL CALOR

ANTÓN FERNÁNDEZ, DANIEL

KRASIMIROV IVANOV, TODOR

ÍNDICE

1. OBJETIVO

El objetivo de esta práctica es paralelizar la distribución del calor en una lámina cuadrada 2D representada por una matriz (NxN) de números reales donde cada elemento representa la temperatura de una zona en un instante, siendo los bordes de la lámina las fuentes de calor. También se obtendrá la aceleración y la eficiencia mediante pruebas con distinto número de núcleos.

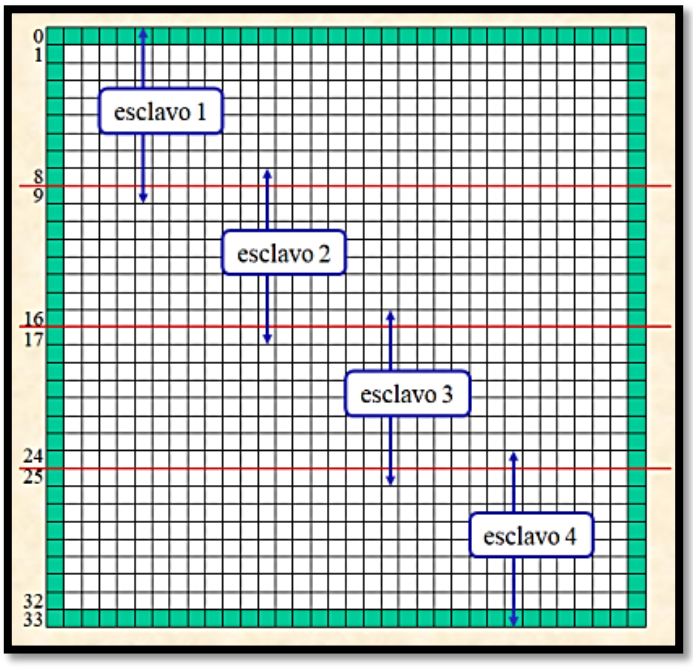
2. RESOLUCIÓN DEL PROBLEMA

La parte paralelizable será el cómputo de los elementos de la matriz mediante la media de sus vecinos (los bordes no se computan). El proceso *maestro*(0) enviará a los procesos *esclavos*(1..N) la matriz inicializada y las señales de continuación y terminación del cómputo.

La matriz se dividirá en secciones de filas enteras (división horizontal) y cada proceso *esclavo* se encargará de computar los cambios de temperatura de su respectiva sección, teniendo que comunicarse con otros *esclavos* para enviar y recibir los cambios producidos en las filas extremas entre secciones. Esto se considera un intercambio (*sendrec*). En la figura de abajo se representa un ejemplo de reparto e intercambio con 4 esclavos.

Para evitar interbloqueos, los procesos pares intercambiarán primero con el proceso colindante superior y los procesos impares lo harán con el proceso colindante inferior, intercambiando posteriormente con el proceso del otro extremo. Es necesario definir que el primer *esclavo* no ha de intercambiar con el proceso superior a él, ya que no existe. De forma análoga, el último *esclavo* tampoco intercambiará con el proceso inferior.

El cómputo finalizará cuando la distribución del calor se haya estabilizado, es decir, cuando los cambios de temperatura globales en la placa se hayan reducido. En este momento es cuando el *maestro* dará la señal de terminación a los esclavos y acabará el programa.



3. ACELERACIÓN Y EFICIENCIA

# ESCLAVOS	TIEMPO	ACELERACIÓN	EFICIENCIA
1	3,482 s	-	-
2	2,729 s	1,276	0,638
3	4,091 s	0,851	0,284
7	22,827 s	0,152	0,022
11	26,951 s	0,129	0,012

*Tiempos medidos con una matriz de 194x194 y con el modo 1 de ejecución.

En este problema el proceso *maestro* no trabaja, por lo que en lugar de basar los cálculos en el número de cores se basará en el número de esclavos, siendo el tiempo de ejecución con 1 esclavo el tiempo secuencial del programa.

Como se puede observar, existe algo de aceleración y una eficiencia aceptable con 2 esclavos puesto que el cómputo de la matriz se divide en 2 trozos paralelos (superior e inferior) y los *esclavos* solo han de intercambiarse una fila por iteración de cómputo.

A medida que el número de *esclavos* va aumentando, las comunicaciones entre ellos también lo hacen y requieren más tiempo que el propio cómputo de las submatrices, produciendo cada vez más desaceleración e ineficiencia.

Como conclusión, el mejor de los tiempos de ejecución será con 1 maestro y 2 esclavos.

4. CÓDIGO PARALELO

```
//-------------------------------------------------------------+
// PCM. Arquitecturas Avanzadas Curso 17/18 ETSISI       12/02/18 |
//                                                              |
// heat2dpar.c: Prueba de distribucion de calor en dos dimensiones |
//          Se da un valor de temperatura en el vertice superior  |
//          izquierda y se aumenta circularmente la temperatura en |
//          los bordes en el sentido de las agujas del reloj, hasta|
//          recorrer el rango de temperaturas contemplado.
//-------------------------------------------------------------+

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#include <math.h>

#include "mpi.h"

#include "mapapixel.h"

#define N              194   // Dimension de la placa 2D
                            //   194, 258, 322, 386 y 450
#define TEMP_MIN       0.0
#define TEMP_MAX       100.0
#define MIN_DIFF_TEMP  0.0001 // Para considerar temperaturas iguales
#define TEMP_MEDIA     71.5    // Temperatura soportable
```

```
#define AZUL            448    // Codigo color azul
#define ROJO              7    // Codigo color rojo
#define ITERACIONES     300    // Cada cuantas iteraciones se visualiza

typedef enum {superior, derecho, inferior, izquierdo} tBorde;

static int modo;               // 0 => Monocromatico 1 => Policromatico
static int yo, numEsclavos, filasPorEsclavo;
static double x[N][N], y[N][N];

//---------------------------------------------------------------------
void inicializarBorde(tBorde elBorde, double temperatura) {
  int i;
  double delta;

  delta = TEMP_MAX / (double) N;
  for (i=0; i<N; i++) {
    switch (elBorde) {
      case superior  : x[ 0 ][ i ] = temperatura; break;
      case derecho   : x[ i ][N-1] = temperatura; break;
      case inferior  : x[N-1][ i ] = temperatura; break;
      case izquierdo : x[ i ][ 0 ] = temperatura; break;
    }
    temperatura += delta;
    if (temperatura > TEMP_MAX) delta *= -1.0;
  }
}
```

```
//---------------------------------------------------------------
void inicializarMatriz(void) {
  int i,j;

  inicializarBorde(superior,  84.02);
  inicializarBorde(derecho,   39.44);
  inicializarBorde(inferior,  78.31);
  inicializarBorde(izquierdo, 79.84);

  // Inicializo el interior
  for (i=1; i<N-1; i++)
    for (j=1; j<N-1; j++)
      x[i][j] = 0.0;
}

//---------------------------------------------------------------
//    Funcion que permite depurar
//---------------------------------------------------------------
void imprimirMatriz(double M[][N]) {
  int i,j;

  printf ("----------------------------------------\n");
  for (i=0; i<N; i++) {
    for (j=0; j<N; j++)
      printf ("%6.2f ", M[i][j]);
    printf ("\n");
  }
  printf ("----------------------------------------\n");
```

```
}



//            CODIGO DEL ESCLAVO
//---------------------------------------------------------------
void intercambiarFilasFrontera(double M[][N], int filaInicial, int filaFinal) {
  MPI_Status estado;

  if(yo % 2 == 0) { // Si soy un proceso par
                    // Intercambiar con el de arriba (nunca puede ser el primer esclavo)
                    MPI_Sendrecv(&M[filaInicial][0], N, MPI_DOUBLE, yo-1, 1,
                    &M[filaInicial-1][0], N, MPI_DOUBLE, yo-1, 1, MPI_COMM_WORLD,
                    &estado);

                    if (yo != numEsclavos) // Intercambiar con el de abajo si no soy el ultimo esclavo
                       MPI_Sendrecv(&M[filaFinal][0], N, MPI_DOUBLE, yo+1, 1,
                            &M[filaFinal+1][0], N, MPI_DOUBLE, yo+1, 1, MPI_COMM_WORLD,
                            &estado);
           }
  else { //Si no (soy un proceso impar)
                    if (yo != numEsclavos) // Intercambiar con el de abajo si no soy el ultimo esclavo
                            MPI_Sendrecv(&M[filaFinal][0], N, MPI_DOUBLE, yo+1, 1,
                            &M[filaFinal+1][0], N, MPI_DOUBLE, yo+1, 1, MPI_COMM_WORLD,
                            &estado);
                    if (yo != 1) // Intercambiar con el de arriba si no soy el primer esclavo
                       MPI_Sendrecv(&M[filaInicial][0], N, MPI_DOUBLE, yo-1, 1,
                       &M[filaInicial-1][0], N, MPI_DOUBLE, yo-1, 1, MPI_COMM_WORLD,
                       &estado);
```

```
            }
}

//---------------------------------------------------------------
int computarFilas (int filaInicial, int filaFinal) {
  int i, j, iter, cambios;
  double XX[N][N]; // Valores iniciales para computar cambios

  memcpy (&XX[filaInicial][0], &x[filaInicial][0], filasPorEsclavo*N*sizeof(double));
  for (iter=0; iter<ITERACIONES; iter+=2) {
    for (i=filaInicial; i<=filaFinal; i++)
      for (j=1; j<N-1; j++)
        y[i][j] = 0.25 * (x[i][j-1]+x[i-1][j]+x[i][j+1]+x[i+1][j]);
    intercambiarFilasFrontera(y, filaInicial, filaFinal);
    for (i=filaInicial; i<=filaFinal; i++)
      for (j=1; j<N-1; j++)
        x[i][j] = 0.25 * (y[i][j-1]+y[i-1][j]+y[i][j+1]+y[i+1][j]);
    intercambiarFilasFrontera(x, filaInicial, filaFinal);
  }
  // Registro los cambios
  cambios = 0;
  for (i=filaInicial; i<=filaFinal; i++)
    for (j=1; j<N-1; j++)
      if (fabs(XX[i][j] - x[i][j]) > MIN_DIFF_TEMP) cambios++;
  return cambios;
}

//---------------------------------------------------------------
```

```
void esclavo (void) {
  MPI_Status estado;
  int cambios, filaInicial, filaFinal, seguir;

  // Calcular filaInicial y filaFinal
  filaInicial = ((yo-1) * filasPorEsclavo) + 1;
  filaFinal = yo * filasPorEsclavo;
  // Recibir la matriz inicial en x[][]
  MPI_Recv (&x[0][0], N*N, MPI_DOUBLE, 0, 1, MPI_COMM_WORLD, &estado);
  // Replico la Matriz x en la Matriz y (usar memcpy)
  memcpy(&y[0][0], &x[0][0], N*N*sizeof(double));
  do {
          cambios = computarFilas(filaInicial, filaFinal);
          MPI_Send(&x[filaInicial][0], N*filasPorEsclavo,MPI_DOUBLE,0,1,MPI_COMM_WORLD);
          MPI_Send(&cambios,1,MPI_INT,0,1,MPI_COMM_WORLD);
          MPI_Recv(&seguir, 1,MPI_INT,0,1,MPI_COMM_WORLD,&estado);
  //  Computar filas hasta que  maestro detecte el fin
  } while (seguir);
}

//            CODIGO DEL MAESTRO

//-----------------------------------------------------------------
void dibujar (void) {
  int i, j;

  for (i=0; i<N; i++)
    for (j=0; j<N; j++)
```

```
    if     (modo == 1)          mapiDibujarPunto (i, j, (int) x[i][j]);
    else if (x[i][j] < TEMP_MEDIA) mapiDibujarPunto (i, j, AZUL);
    else                       mapiDibujarPunto (i, j, ROJO);
}

//---------------------------------------------------------------
void computar (void) {
  int cambiosLocal, cambiosGlobal, vueltas=0, esc, fila, seguir=1;
  struct timeval t0, tf, t;
  MPI_Status estado;

  inicializarMatriz();
  //printf ("Maestro matriz inicial\n"); imprimirMatriz(x);
  gettimeofday (&t0, NULL);
  // Difundir matriz x[][] a los esclavos
  for (esc = 1; esc <= numEsclavos; esc++)
      MPI_Send(&x[0][0],N*N, MPI_DOUBLE,esc,1,MPI_COMM_WORLD);
  // Repetir hasta detectar el fin (cambios <= 20)
  printf("\nTraza 0, MASTER: %d\n",yo);
  do {
    cambiosGlobal = 0;
    vueltas++;
    // Indicar a los esclavos que hay que seguir computando
    // Recoger filas de esclavos: Las computadas
    for (esc = 1; esc <= numEsclavos; esc++){
          MPI_Send(&seguir,1,MPI_INT,esc,1,MPI_COMM_WORLD);
          fila = 1 + (esc - 1) * filasPorEsclavo;
          MPI_Recv(&x[fila][0], N*filasPorEsclavo,
```

```
                    MPI_DOUBLE,esc,1,MPI_COMM_WORLD,&estado);
            MPI_Recv(&cambiosLocal,1,MPI_INT,esc,1,MPI_COMM_WORLD,&estado);
            cambiosGlobal += cambiosLocal;
    }
    printf("Cambios globales : %d\n",cambiosGlobal);
    //printf ("Maestro matriz devuelta\n"); imprimirMatriz(x);
    dibujar(); // Para ir viendo la evolucion del computo
  } while (cambiosGlobal > 20);
  // Enviar indicacion de terminacion a los esclavos
  printf("TERMINA\n");
  seguir = 0;
  for (esc = 1; esc <= numEsclavos;esc++)
      MPI_Send(&seguir,1,MPI_INT,esc,1,MPI_COMM_WORLD);
  gettimeofday (&tf, NULL);
  timersub (&tf, &t0, &t);
  printf ("Tiempo = %ld:%ld (seg:mseg) iteraciones totales = %d\n",
          t.tv_sec, t.tv_usec/1000, ITERACIONES*vueltas);
}

//-------------------------------------------------------------------
void finalizar() {
  // Envio mensaje de terminacion a los esclavos
  MPI_Finalize();
  exit (0);
}

//-------------------------------------------------------------------
int main (int argc, char *argv[]) {
```

```
  MPI_Init (&argc, &argv);
  MPI_Comm_rank (MPI_COMM_WORLD, &yo);
  MPI_Comm_size (MPI_COMM_WORLD, &numEsclavos);

  numEsclavos--;  // El maestro no trabaja
  filasPorEsclavo = (N-2) / numEsclavos;
  modo = atoi(argv[1]);
  if (yo == 0) {
      mapiProfundidadColor (3);
      mapiInicializar (N, N, computar, NULL, finalizar);
  } else
      esclavo ( );
  MPI_Finalize();
  return (0);
}
```