UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

CÓMPUTO DE ALTO DESEMPEÑO

Proyecto MPI

Grupo

2

Integrantes

Ferrusca Ortiz Jorge Luis
Hernández González Ricardo Omar
Torres Caballero Bruno

Profesor

M.I. Oscar René Valdez Casillas

Semestre

2020-1

Fecha de Entrega

21 de noviembre de 2019

Indice:

Objetivo y definición del problema

Se necesita un programa que pueda leer archivos de imágenes en formato pnm, obtenga la imagen en escala de grises, y obtenga la imagen con los colore invertidos. Las imágenes obtenidas serán guardadas en archivos.

Se tiene como objetivo la realización de este programa lo más paralelizado posible, de tal manera que el proceso de obtención de imágenes sea lo más óptimo y rápido posible.

Desarrollo

a) Análisis y Diseño de la paralelización

Para el desarrollo del programa en paralelo, se ponderó el uso de la descomposición de dominio o datos, ya que durante el análisis nos dimos cuenta si bien no es un programa que requiera realizar múltiples tareas, los datos que procesan las pocas tareas realizadas son bastantes y están dados en base al tamaño o resolución de una imagen. Tan solo para una imagen de 512x512 se tenía que hacer un procesamiento de los pixeles de dicha imagen multiplicados por el numero de colores de la misma, que en este caso eran 3.

Las posibles distribuciones de datos se centraban en la parte de procesamiento de todas las matrices de color o bien la transformación a los valores inversos para obtener la imagen en negativo, por ejemplo:

Para la obtención de los valores de la matriz de escala de grises:

```
  for(int i = (offset * rank)+1; i <= ((offset * rank) + offset); ++i) {
    grayscale_matrix[set_counter] = (rgb_matrix[i*3] + rgb_matrix[(i*3)+1] +
rgb_matrix[(i*3)+2]) / 3;
    ++set_counter;
  }
```

Para la obtención de los valores negativos:

```
  for(int i = (num_lines_per_proc * rank)+1; i <= ((num_lines_per_proc * rank) +
num_lines_per_proc); ++i) {
    rgb_negative_matrix[set_counter] = MAX_COLOR_VALUE - rgb_matrix[i];
    ++set_counter;
  }
```

En donde cada uno de los procesos tenía su propia matriz local de valores tanto en escala de grises como en negativo, posteriormente dichos valores eran agrupados para obtener las imágenes de resultado.

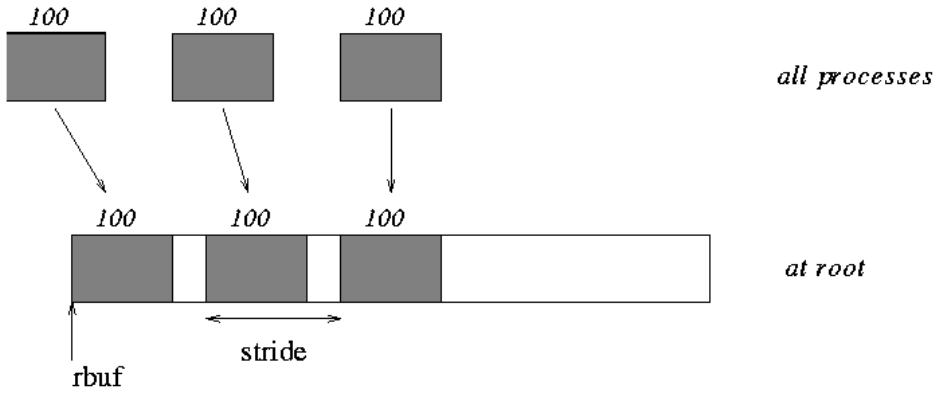
Cabe señalar que en este punto, los ciclos ***for*** fueron divididos entre el numero de procesos y a cada uno de ellos le correspondía una parte para acceder a los datos globales de la matriz RGB original, sin embargo dicha matriz RGB podía ser también descompuesta en matrices locales por proceso, para que dichos procesos pudieran realizar un ciclo ***for*** local. Esto podía lograrse mediante el paso de mensajes, sin embargo no lo consideramos tan necesario, ya que el acceso a una variable global creemos que si bien no es lo ideal, puede ser más rápido al no necesitar repartir los datos.

Respecto al patrón de programación empleado, se utilizó el de Master-Slave (maestro-esclavo) ya que los esclavos se utilizaron para el procesamiento de los datos de la imagen, y el maestro escribía sobre el archivo resultante. Esto es importante ya que al intentar escribir con un file descriptor convencional de forma paralela no era soportado. Unicamente el primer hilo que lo obtuviera iba a poder escribir. Para solucionar este problema se utilizaron las funciones open(), pwrite() y pread()de la biblioteca estándar de *Unix,* sin embargo el problema ahora era en que el algoritmo era **poco eficiente**, ya que toda vez que cada proceso tenía un segmento de colores de los pixeles de la imagen resultante (en una matriz

local), se tenía que estar recorriendo todo el archivo destino para saber el punto exacto donde se iban a escribir dichos valores de color. Por consecuencia, se utilizó al maestro para escribir directamente sobre el archivo resultante.

b) Funciones de MPI usadas y justificación para el caso de MPI

Se utilizó el modelo basado en la ejecución segmentada en el caso de los ciclos, además de utilizar la función MPI_Scatterv(), la cual nos permitía recoger los valores de cada matriz local. A diferencia de MPI_Scatter(), la función que empleamos nos permitía especificar el offset de cada chuck de datos a la hora que se iba a agrupar, permitiendo mantener el orden respecto al número de proceso o *rank*. Con esto último, asegurábamos que sin importar el orden en que los procesos terminaban de procesar los datos, siempre se iba a respetar el orden de los datos resultantes, para que la imagen conservara su orden. El principio gráficamente es el siguiente:



Código serial:

```
#include <stdio.h>
#include <stdlib.h>

int** allocate_memory();
int* allocate_lineal_memory(int length);
void get_headers(FILE *f);
```

```
void set_headers(FILE *f, const char* file_signature, char* comment_buffer, int rows, int columns, int max_color);
void validate_args(int argc);
void check_file(FILE *f);
void free_mem (int **matriz);
int* gray_scale_matrix (int *red_matrix, int *green_matrix, int *blue_matrix);
int* negative_matrix (int *matriz);
void generate_grayscale_file (char* filename, int* matrix);
void generate_negative_file (char* filename, int* red, int* green, int* blue);

char comment_buffer[100];
char file_signature[3];
int MAX_COLOR_VALUE, ROWS, COLUMNS;
const char* GRAYSCALE_SIGNATURE = "P2";
const char* NEGATIVE_SIGNATURE = "P3";
static int PIXELS = 0;

int main(int argc, char *argv[]){
  validate_args(argc);
  FILE *file;
  int r,g,b, count = 0;
  int *red_matrix, *green_matrix, *blue_matrix;
  file = fopen(argv[1], "r");
  check_file(file);
  get_headers(file);

  red_matrix = allocate_lineal_memory(PIXELS);
  green_matrix = allocate_lineal_memory(PIXELS);
  blue_matrix = allocate_lineal_memory(PIXELS);

  //printf("header: %s\nCommentario: %sTamaño: %d X %d\nMaximo: %d\n",header,comment_buffer,COLUMNS,ROWS,MAX_COLOR_VALUE);

  while(fscanf(file,"%d\n%d\n%d\n",&r,&g,&b) != EOF){
    red_matrix[count] = r;
```

```
    green_matrix[count] = g;
    blue_matrix[count] = b;
    ++count;
  }

  generate_grayscale_file("lena_bn.pnm", gray_scale_matrix(red_matrix,green_matrix,blue_matrix));
  generate_negative_file("lena_neg.pnm", negative_matrix(red_matrix),
negative_matrix(green_matrix), negative_matrix(blue_matrix));

  free(red_matrix);
  free(green_matrix);
  free(blue_matrix);
  return 0;
}

/**
 * Function that gets the file .pnm headers
 * Following the structure:
 * [File_signature]
 * [Here goes a comment]
 * [width] [height]
 * [max RGB value]
 */
void get_headers(FILE *file) {
  fscanf(file,"%s\n", file_signature);
  fgets(comment_buffer, 100, file);
  fscanf(file,"%d %d\n", &COLUMNS, &ROWS);
  PIXELS = ROWS * COLUMNS;
  fscanf(file,"%d\n",&MAX_COLOR_VALUE);
}

void generate_negative_file(char* filename, int* red, int* green, int* blue){
  FILE *file;
  file = fopen(filename, "w");
  check_file(file);
```

```
  set_headers(file, NEGATIVE_SIGNATURE, comment_buffer, ROWS, COLUMNS,
MAX_COLOR_VALUE);
  for(int i=0; i< PIXELS; ++i) {
    fprintf(file, "%d\n%d\n%d\n", red[i], green[i], blue[i]);
  }
}

void generate_grayscale_file(char* filename, int* matrix){
  FILE *file;
  file = fopen(filename,"w");
  check_file(file);
  set_headers(file, GRAYSCALE_SIGNATURE, comment_buffer, ROWS, COLUMNS,
MAX_COLOR_VALUE);
  for(int i=0; i < PIXELS; ++i) {
    fprintf(file, "%d\n",matrix[i]);
  }
}

int** allocate_memory() {
  int **x;
  int i;
  x = (int **)malloc(COLUMNS*sizeof(int*));
  for (i=0;i<COLUMNS;++i) {
    x[i] = (int*)malloc(ROWS*sizeof(int));
  }
  return x;
}

int* allocate_lineal_memory(int length) {
  return (int *)malloc(length * sizeof(int));
}

int* gray_scale_matrix(int *red_matrix, int *green_matrix, int *blue_matrix){
  int* matrix = allocate_lineal_memory(PIXELS);
  for(int i = 0; i < PIXELS; ++i) {
```

```
    matrix[i] = (red_matrix[i] + green_matrix[i] + blue_matrix[i]) / 3;
  }
  return matrix;
}

int* negative_matrix(int* matrix) {
  int* negative_matrix = allocate_lineal_memory(PIXELS);
  for(int i = 0; i < PIXELS; ++i) {
    negative_matrix[i] = MAX_COLOR_VALUE - matrix[i];
  }
  return negative_matrix;
}

void validate_args(int argc) {
  if(argc < 2){
    printf("Modo de uso:  ejecutable <imagen.pnm>\n");
    exit(1);
  }
}

void check_file(FILE *file) {
  if(file == NULL){
    printf("Error al procesar");
    exit(1);
  }
}

void set_headers(FILE *file, const char* file_signature, char* comment_buffer, int rows, int columns,
int max_color_value) {
  fputs(file_signature,file);
  fputs("\n",file);
  fputs(comment_buffer,file);
  fprintf(file,"%d %d\n", columns, rows);
  fprintf(file, "%d\n", max_color_value);
}
```

Código paralelo:

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#include <time.h>

int** allocate_memory();
int* allocate_lineal_memory(int length);
void get_headers(FILE *f);
void set_headers(FILE *f, const char* file_signature, char* comment_buffer, int rows, int columns, int max_color);
void validate_args(int argc);
void check_file(FILE *f);
void free_mem (int **matriz);
int* gray_scale_matrix (int *rgb_matrix, int *green_matrix, int *blue_matrix);
int* negative_matrix (int *matriz);
void generate_grayscale_file (char* filename, int* matrix);
void generate_negative_file (char* filename, int* red, int* green, int* blue);

char comment_buffer[100];
char file_signature[3];
int MAX_COLOR_VALUE, ROWS, COLUMNS;
const char* GRAYSCALE_SIGNATURE = "P2";
const char* NEGATIVE_SIGNATURE = "P3";
static int PIXELS = 0;
int rank, world_size, set_counter;
double start, end;

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

  validate_args(argc);

  FILE *original_file;
  int *rgb_matrix;
```

```
int colorvalue;

MPI_Status status;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &world_size);

MPI_Barrier(MPI_COMM_WORLD); /* getting time */
start = MPI_Wtime();

original_file = fopen(argv[1], "r");
check_file(original_file);
get_headers(original_file);

int count = 0;

rgb_matrix = allocate_lineal_memory(PIXELS*3);

while(fscanf(original_file,"%d\n",&colorvalue) != EOF){
  rgb_matrix[count] = colorvalue;
  ++count;
}

/***** GENERATING GRAYSCALE MATRIX *****/

int num_pixels_per_proc = (PIXELS/world_size);

int offset = num_pixels_per_proc;
int remainder = PIXELS % world_size;

int *local_rgb_matrix = allocate_lineal_memory(num_pixels_per_proc);

MPI_Scatter(rgb_matrix, num_pixels_per_proc, MPI_INT, local_rgb_matrix,
            num_pixels_per_proc, MPI_INT, 0, MPI_COMM_WORLD);
```

```
  int* grayscale_matrix_recv = allocate_lineal_memory(PIXELS);
  int* grayscale_matrix = allocate_lineal_memory(num_pixels_per_proc);
  int* displs = (int *)malloc(world_size*sizeof(int));
  int* recv_counts = (int *)malloc(world_size*sizeof(int));
  for (int i=0; i<world_size; ++i) {
    displs[i] = i*num_pixels_per_proc+1;
    recv_counts[i] = num_pixels_per_proc;
  }

  set_counter = 0;
  for(int i = (offset * rank)+1; i <= ((offset * rank) + offset); ++i) {
    grayscale_matrix[set_counter] = (rgb_matrix[i*3] + rgb_matrix[(i*3)+1] + rgb_matrix[(i*3)+2]) / 3;
    ++set_counter;
  }

  MPI_Gatherv(grayscale_matrix, num_pixels_per_proc, MPI_INT,
    grayscale_matrix_recv, recv_counts, displs, MPI_INT, 0, MPI_COMM_WORLD);

  /****************** GENERATING GRAYSCALE FILE *********************/

  if (rank == 0) {
    FILE *grayscale_file;
    grayscale_file = fopen("lena_bn.pnm", "w");
    check_file(grayscale_file);
    set_headers(grayscale_file, GRAYSCALE_SIGNATURE, comment_buffer, ROWS, COLUMNS,
MAX_COLOR_VALUE);
    for(int i=0; i < PIXELS; ++i) {
      fprintf(grayscale_file, "%d\n", grayscale_matrix_recv[i]);
    }
  }

  /************** GENERATE RGB-NEGATIVE MATRIX ******************/
  int num_lines_per_proc = (PIXELS*3)/world_size;
  int* rgb_negative_matrix_recv = allocate_lineal_memory(PIXELS*3);
  int* rgb_negative_matrix = allocate_lineal_memory(num_lines_per_proc);
```

```
  // Reassigning gather values
  for (int i=0; i<world_size; ++i) {
    displs[i] = i*num_lines_per_proc+1;
    recv_counts[i] = num_lines_per_proc;
  }

  set_counter = 0;
  for(int i = (num_lines_per_proc * rank)+1; i <= ((num_lines_per_proc * rank) +
num_lines_per_proc); ++i) {
    rgb_negative_matrix[set_counter] = MAX_COLOR_VALUE - rgb_matrix[i];
    ++set_counter;
  }

  MPI_Gatherv(rgb_negative_matrix, num_lines_per_proc, MPI_INT,
    rgb_negative_matrix_recv, recv_counts, displs, MPI_INT, 0, MPI_COMM_WORLD);

  /************** GENERATING NEGATIVE FILE ******************/

  if (rank == 0) {
    FILE *negative_file;
    negative_file = fopen("lena_neg.pnm", "w");
    check_file(negative_file);
    set_headers(negative_file, NEGATIVE_SIGNATURE, comment_buffer, ROWS, COLUMNS,
MAX_COLOR_VALUE);
    for(int i=0; i< PIXELS; ++i) {
      fprintf(negative_file, "%d\n%d\n%d\n", rgb_negative_matrix_recv[i*3],
rgb_negative_matrix_recv[(i*3)+1], rgb_negative_matrix_recv[(i*3)+2]);
    }
  }

  if (rank == 0) {
    free(grayscale_matrix_recv);
    free(rgb_matrix);
  }
```

```
  free(grayscale_matrix);
  free(rgb_negative_matrix);

  MPI_Barrier(MPI_COMM_WORLD); /* ending */
  end = MPI_Wtime();

  MPI_Finalize();
  if (rank == 0) {
    printf("Runtime = %f\n", end-start);
  }
  return 0;
}

/**
 * Function that gets the file .pnm headers
 * Following the structure:
 * [File_signature]
 * [Here goes a comment]
 * [width] [height]
 * [max RGB value]
 */
void get_headers(FILE *file) {
  fscanf(file,"%s\n", file_signature);
  fgets(comment_buffer, 100, file);
  fscanf(file,"%d %d\n", &COLUMNS, &ROWS);
  PIXELS = ROWS * COLUMNS;
  fscanf(file,"%d\n",&MAX_COLOR_VALUE);
}

int* allocate_lineal_memory(int length) {
  return (int *)malloc(length * sizeof(int));
}

void validate_args(int argc) {
  if(argc < 2){
```

```
    printf("Modo de uso:  ejecutable <imagen.pnm>\n");
    exit(1);
  }
}

void check_file(FILE *file) {
  if(file == NULL){
    printf("Error al procesar");
    exit(1);
  }
}

void set_headers(FILE *file, const char* file_signature, char* comment_buffer, int rows, int columns,
int max_color_value) {
  fputs(file_signature,file);
  fputs("\n",file);
  fputs(comment_buffer,file);
  fprintf(file,"%d %d\n", columns, rows);
  fprintf(file, "%d\n", max_color_value);
}
```

d) Manual de uso del programa

Para compilar:

$ mpicc parallelMPI.c .o parallelMPI

Para ejecutar:

$ mpirun -np <numero_de_procesos> ./parallelMPI <nombre_de_imagen>

e) Conclusiones

Ferrusca Ortiz Jorge Luis

La implementación de este proyecto nos llevó al análisis de los distintos tipos de soluciones del problema. Si bien existen muchas formas de realizar un programa de manera secuencial, para el caso del paralelo teníamos aún más opciones sobre como realizarlo, aunque cabe señalar que no muchas funciones de MPI nos eran de utilidad dada la naturaleza del problema, como por ejemplo el hecho de que los datos de salida o pixeles tenían que conservar su orden respecto a cuando fueron repartidos.

Por otro lado pudimos ver con mayor claridad el rendimiento que pueden tener los programas paralelos en relación a los secuenciales, aunque como dije anteriormente, dada la naturaleza del algoritmo, creemos que llega un punto en el que el programa ya no puede ser más paralelizable.

Hernández González Ricardo Omar

La parte más complicada de este proyecto fue hacer la parte paralela, ya que al principio el programa lo teníamos en funciones, pero para paralelizar no sabíamos como hacerlo con ellas, entonces pasamos todo a serial. La realización de este proyecto consideramos fue exitosa.

Torres Caballero Bruno

Este proyecto tuvo algunas dificultades al momento de paralelizar el programa, costó un poco de trabajo el lograr hacer que el sistema dividiera bien la carga entre los hilos, y no se ejecutara nada más es uno.

Referencias

Guerrero Martínez, D, Rodríguez Lumley, S. (2010). *Programación Paralela*. Consultado el 20 Noviembre 2019, de Universidad de Granada Sitio web: https://lsi.ugr.es/jmantas/ppr/ayuda/mpi_ayuda.php

---- (2000). *Introducción a MPI*. de Informática.es. Consultado el 18 de noviembre de: http://informatica.uv.es/iiguia/ALP/materiales2005/2_2_introMPI.htm

Ledesma, S. (2000). *PPM Image*. de Wintempla. Consultado el 15 de noviembre: http://sintesis.ugto.mx/WintemplaWeb/02Wintempla/10Images/07PPM%20Image/index.htm

Vazquez, L.. (2005). *Tutorial para el uso de imágenes en Octave*. de fing.edu Consultado el 20 de noviembre de 2019 de https://iie.fing.edu.uy/investigacion/grupos/gti/timag/papers/octima-tutor.pdf