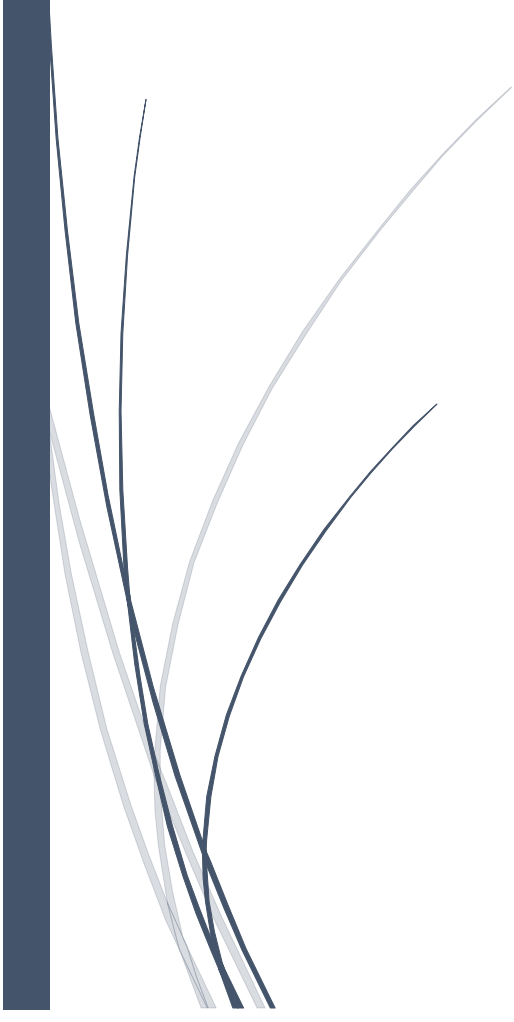
4-5-2019

Práctica 2: MPI2:

Sistema de Distribución de Rende rizado de Gráficos



ANDRÉS GUTIÉRREZ CEPEDA
INFORMATES SOFTWARE DEVELOPMENT

Tabla de contenido

Enunciado del problema:

Utilizaremos las primitivas pertinentes MPI2 como acceso paralelo a disco y gestión de procesos dinámico:
Inicialmente el usuario lanzará un solo proceso mediante mpirun -np 1 ./pract2. Con ello MPI lanza un primer proceso que será el que tiene acceso a la pantalla de gráficos, pero no a disco. Él mismo será el encargado de levantar N procesos (con N definido en tiempo de compilación como una constante) que tendrán acceso a disco, pero no a gráficos directamente.
Los nuevos procesos lanzados se encargarán de leer de forma paralela los datos del archivo foto.dat. Después, se encargarán de ir enviando los pixels al primer elemento de proceso para que éste se encargue de representarlo en pantalla.
Usaremos la plantilla pract2.c para comenzar a desarrollar la práctica.
En ella debemos completar el código que ejecuta el proceso con acceso a la ventana de gráficos (rank 0 inicial) y la de los procesos “trabajadores”.
Se proporciona el archivo foto.dat. La estructura interna de este archivo es 400 filas por 400 columnas de puntos. Cada punto está formado por una tripleta de tres “unsigned char” correspondiendo al valor R,G y B de cada uno de los colores primarios. Estos valores se pueden usar para la función dibujaPunto.

Planteamiento de la solución:

Para el desarrollo de esta practica deberemos de estudiar como tratar el fichero de datos sobre el que tenemos que trabajar. Deberemos hacer dos tipos de particiones, la distribución para las view y la distribución de líneas de lectura, los dos serán muy similares, pero con alguna diferencia.
Lo primero es calcular el tamaño de una línea para poder dividir el fichero de manera correcta, el fichero representa una imagen de 400x400 donde se localizan los valores de rgb que identifican cada pixel, todos ellos como unsigned char, por una línea tendrá un tamaño de (400x3xsizeof(unsigned_char)).
Una vez sabemos cuanto ocupa una línea, para identificar el tamaño de la view simplemente habrá que multiplicar el tamaño de una línea por la función suelo de 400 entre el numero de procesos ejecutados en la lectura de la imagen.
El numero de líneas que deberá leer cada fichero será igual que lo ultimo que he mencionado, será 400 líneas que tiene el fichero dividido por el numero de procesos. En caso de que el numero de procesos no sea par, el ultimo proceso se hará cargo de las ultimas líneas que quedan sin repartir.

Diseño del problema:

```
12   #define np 7
```
Declaramos el numero de procesos que leerán el fichero de datos.

```
23   long particion=(400*3*sizeof(unsigned char))*floor(400/np);
```
Declaramos como explicamos en la sección anterior el tamaño de la view.

```
69  int main (int argc, char *argv[]) {
70
71    int rank,size;
72    MPI_Comm commPadre, intercomm;
73    int tag;
74    MPI_Status status;
75    int buf[5];
76    unsigned char bufr[3];
77    int errcodes[np];
78    int lineas;
79    int media;
80    int tr;
81    int tg;
82    int tb;
83
84    MPI_File fh;
85    MPI_Offset offset;
```

Declaramos todo lo necesario para la realización de la practica, como los comunicadores, etiquetas, buffers, etc, así como líneas que será la partición del fichero para cada proceso, media que se utilizará en el filtro blanco y negro, y tr, tg y tb que se utilizará para el filtro sepia.

```
88  MPI_Init(&argc, &argv);
89  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
90  MPI_Comm_size(MPI_COMM_WORLD, &size);
91  MPI_Comm_get_parent( &commPadre );
92  if ( (commPadre==MPI_COMM_NULL)
93       && (rank==0) )  {
94
95      initX();
96
97      MPI_Comm_spawn( "exec/pract2", &argv[1], np, MPI_INFO_NULL, 0, MPI_COMM_WORLD, &intercomm, errcodes );
98      /* Codigo del maestro */
99
100
101
102     for(int i=0;i<400*400;i++){
103           MPI_Recv(&buf,5,MPI_INT,MPI_ANY_SOURCE,0,intercomm,&status);
104           dibujaPunto(buf[0],buf[1],buf[2],buf[3],buf[4]);
105     }
106
107     sleep(5);
```

Lo primero que haremos será comprobar que el proceso de Rank 0 y creado originalmente por el lanzamiento de la ejecución de la practica, si esto es así, lo primero que haremos será lanzar el resto de los procesos, los cuales tendrán por argumento el tipo de filtro, a su vez este proceso al tener el acceso a imagen será el que se encargará de recibir los datos del resto de procesos y dibujar los pixel en las posiciones correspondientes.

```
116 else {
117
118
119   /* Codigo de todos los trabajadores */
120   /* El archivo sobre el que debemos trabajar es foto.dat */
121     MPI_File_open(MPI_COMM_WORLD,"dirs/foto.dat",MPI_MODE_RDONLY,MPI_INFO_NULL,&fh);
122     MPI_File_set_view(fh, particion*rank, MPI_UNSIGNED_CHAR, MPI_UNSIGNED_CHAR, "native", MPI_INFO_NULL);
123     if(rank==np-1){
124           lineas=floor(400/np)+(400-(floor(400/np)*np));
125     }else{
126           lineas=floor(400/np);
127     }
128     for(int i=0;i<lineas;i++){
129           for(int j=0;j<400;j++){
130           MPI_File_read(fh, bufr, 3, MPI_UNSIGNED_CHAR, &status);
131           //printf("%d %d %d\n",(int)bufr[0],(int)bufr[1],(int)bufr[2]);
132           buf[0]=j;//para mandar las cordenadas correctas
133           buf[1]=i+(rank*(ceil(400/np)));
134           if(*argv[1]=='n'){
135                 buf[2]=(int)bufr[0];
136                 buf[3]=(int)bufr[1];
137                 buf[4]=(int)bufr[2];
```

En caso de ser los procesos lanzados por el proceso principal, lo que se hará será distribuir el fichero de manera equitativa, salvo para el ultimo proceso que cuando el numero de procesos sea impar recibirá una parte mayor, lo primero que se hare es abrir el fichero y establecer el view en función de la partición que se definió al principio, una vez todo esto se ha realizado se comenzara a leer tantas líneas como tenga asignado el proceso y enviara tanto las coordenadas como los valores de rgb.

```
198               MPI_Bsend(&buf,5,MPI_INT,0,0,commPadre);
199               }
200         }
201         MPI_File_close(&fh);
```

Algunos ejemplos de filtros serian los siguientes:

```
138               }else if(*argv[1]=='b'){
139                     media=((int)bufr[0]+(int)bufr[1]+(int)bufr[2])/3;
140                     buf[2]=media;
141                     buf[3]=media;
142                     buf[4]=media;
```

Para realizar un tratamiento de blanco y negro a la imagen es sencillo, una vez

se tienen los valores de rgb del pixel, se realiza una media entre ellos y dicha media sustituirá a los valores de rgb del pixel.

```
143             }else if(*argv[1]=='s'){
144                   tr = (0.393*(int)bufr[0]) + (0.769*(int)bufr[1]) + (0.189*(int)bufr[2]);
145                   tg = (0.349*(int)bufr[0]) + (0.686*(int)bufr[1]) + (0.168*(int)bufr[2]);
146                   tb = (0.272*(int)bufr[0]) + (0.534*(int)bufr[1]) + (0.131*(int)bufr[2]);
147
148                   if(tr > 255){
149                         buf[2] = 255;
150                   }else{
151                         buf[2] = tr;
152                   }
153
154                   if(tg > 255){
155                         buf[3] = 255;
156                   }else{
157                         buf[3] = tg;
158                   }
159
160                   if(tb > 255){
161                         buf[4] = 255;
162                   }else{
163                         buf[4] = tb;
164                   }
```

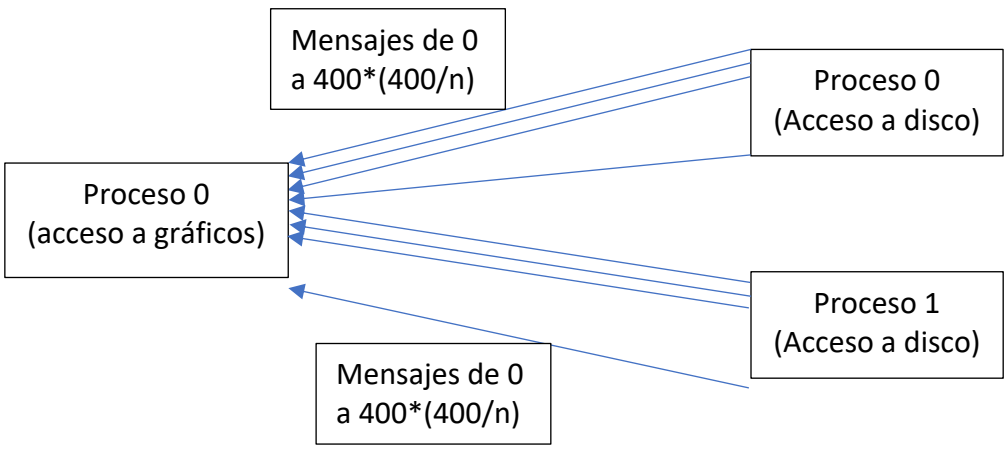
En el caso del sepia es mas complejo, dado que hay que aplicar una función especial de valores para cada valor de rgb, teniendo en cuenta también los propios valores de rgb. En caso de que el valor de rgb supere 250 se debe controlar que se mantenga en ese valor y no lo supere.

```
165             }else if(*argv[1]=='A'){
166                   media=((int)bufr[0]+(int)bufr[1]+(int)bufr[2])/3;
167                   buf[2]=0;
168                   buf[3]=0;
169                   buf[4]=media;
170             }
171             else if(*argv[1]=='V'){
172                   media=((int)bufr[0]+(int)bufr[1]+(int)bufr[2])/3;
173                   buf[2]=0;
174                   buf[3]=media;
175                   buf[4]=0;
176             }else if(*argv[1]=='R'){
177                   media=((int)bufr[0]+(int)bufr[1]+(int)bufr[2])/3;
178                   buf[2]=media;
179                   buf[3]=0;
180                   buf[4]=0;
181             }else if(*argv[1]=='a'){
182                   media=((int)bufr[0]+(int)bufr[1]+(int)bufr[2])/3;
183                   buf[2]=media;
184                   buf[3]=media;
185                   buf[4]=0;
```

Otros ejemplos mas sencillos serian los siguientes, en los cuales se aplica un filtro de color con los 6 colores básicos que se pueden obtener con rgb, que son el rojo, verde, azul, amarillo, magenta y cian. Para aplicarlos simplemente hay que aplicar una media de todos los valores de rgb a los valores de rgb que son necesarios para obtener el color deseado.

Explicación del flujo de datos:

Explicare el flujo de datos para un caso sencillo, en el cual el proceso lanzado originalmente solo lanza un total de 2 procesos mas, esos procesos se dividen el archivo y mandaran los datos al proceso original que los ira imprimiendo según le lleguen esos datos.



Instrucciones de como compilar y ejecutar:

Se dispone de un archivo makefile para facilitar el manejo del programa, para la compilación se realizará el comando make all. Y para la ejecución de los distintos test se colocará el comando make test<identificador_test>.
Los identificadores son los siguientes:

- Base
- BlancoYNegro
- Sepia
- Azul
- Rojo
- Verde
- Amarillo
- Violeta
- Cyan

Conclusiones:

Como conclusiones de este trabajo podemos destacar la facilidad que nos da mpi para dividir el acceso que tienen los distintos procesos a las funciones del equipo, en este caso funciones de acceso a disco y funciones de acceso a gráficos, sin perder la facilidad de comunicación que existe entre estos procesos.