APLICACIÓN PARA LA DIGITALIZACIÓN DE LIBROS

INFORME TÉCNICO

Arquitectura de Computadores, 21/12/2018

Ander Gil
marq04 *Aitor Gonzalez*
Daniel Ruskov

ÍNDICE

ÍNDICE **1**

INTRODUCCIÓN **2**

TÉCNICAS DE TRABAJO **2**

DESARROLLO **2**

Versión serie 3

Versión paralela 5

RESULTADOS **7**

CONCLUSIONES **9**

INFORMACIÓN ADICIONAL **10**

BIBLIOGRAFÍA **10**

INTRODUCCIÓN

En este proyecto se plantea el problema de digitalización de libros antiguos de forma eficiente. El proceso de digitalización consiste en suavizar el ruido de una imagen (libro escaneado) para una mejor visualización del mismo. Nuestro objetivo es completar el código necesario para que, dada una imagen, esta pase por un tratamiento de suavizado o filtrado y después debe ser encriptada (codificada mediante una matriz clave 2x2). Inicialmente el código se ejecutará en serie (secuencialmente) por lo que tomará un largo tiempo poder tratar muchos libros.

Nuestro segundo objetivo es paralelizar el código con las técnicas de sincronización y planificación, de forma que se traten varias partes de la imagen al mismo tiempo (paralelamente) logrando minimizar lo máximo posible el tiempo de tratado de cada imagen.

Finalmente haremos un estudio de los tiempos de ejecución según la dimensión en píxeles de diferentes libros (imágenes) en serie y paralelo (de 2 a 32 hilos) para poder sacar conclusiones de cuál sería la más eficiente para cada libro.

TÉCNICAS DE TRABAJO

La aplicación será programada en lenguaje C utilizando PuTTY (cliente SSH) y el servidor Xming para poder visualizar el resultado de los libros tratados. Inicialmente dispondremos de los siguientes ficheros:

```
encript.c  filter.h    Libro3.pgm  pixmap.o           recuperar
encript.h  Libro1.pgm  pagina_s.c  preparar_subida.c  upload.h
filter.c   Libro2.pgm  pixmap.h    preparar_subida.h  upload.o
```

siendo los ficheros.c los que debemos editar para completar el código, los libros 1, 2 y 3 (.pgm) las imágenes de prueba que debemos tratar, y el archivo recuperar el cual dada la imagen cifrada, la imagen filtrada y la clave de descifrado, descifra la imagen y la compara con la cifrada y así poder saber si el proceso de cifrado es correcto. Los ficheros.h son los ficheros de cabecera para hacer las llamadas a otras funciones y los ficheros.o de tipo objeto, ya que trataremos imágenes.

DESARROLLO

Nota: todos los archivos para la versión en serie han sido nombrados *nombre*_serie.c para ser diferenciados con los ficheros de la versión paralelo *nombre*_parallel.c.

❖ Versión serie

Inicialmente hemos completado el código de los ficheros filter.c, encript.c y preparar_subida.c para que, dado un libro, la aplicación cree una imagen suavizada y otra cifrada.

filter_serie.c: se ha completado la función filtrar_pagina, la cual recorre los píxeles de la foto por filas y por cada pixel calcula la media de radio 1 de todos los píxeles vecinos con la ayuda de la función añadida get_px guardandolo en una copia del libro inicial para crear el libro filtrado. La función filtrar_pagina crea la nueva imagen modificando la copia y además devuelve el valor de el valor medio de todos los píxeles de la imágen(avg) menos el valor mínimo entre todos los píxeles (min). De esta forma obtenemos un límite de filtrado.

Los límites para cada libro son: Libro1=177; Libro2=177.5; Libro3=125.

```
/* Aplicar filtro a una pagina                                          */
/************************************************************************/
int get_px(pagina *in,int i,int j){
int resul=0;
        for (int x=i-1;x<i+2;x++){
                for(int y=j-1;y<j+2;y++){
                        resul+= in->im[x][y];
                }
        }
return resul/9;
}//get_px

double filtrar_pagina(pagina *in_page, pagina *out_page){
int min=400;
double  avg=0;
long tot=0;
int px=0;
for(int i=1;i<in_page->h-1;i++){
        for(int j=1;j<in_page->w-1;j++){
                px=get_px(in_page,i,j);
                out_page->im[i][j]=px;
                if(px<min)min=px;
                tot+=px;
        }
}
avg=(double)tot/((in_page->h-2)*(in_page->w-2));
printf("\n Minimo: %d  Average: %.2f   Total: %ld\n", min, avg, tot);
return(avg-min);
}//filtrar_pagina
```

encript_serie.c: se ha completado la función generar_pagina_encriptada de forma que cogiendo pixeles de dos en dos por filas, los multiplique por una matriz clave de encriptado 2x2 y de esta forma obtener dos píxeles cifrados mediante la

llamada a la función añadida encriptar_pixeles. Estos píxeles cifrados se guardan en una copia de la imagen de entrada.

```
void encriptar_pixeles (unsigned char *v1, unsigned char *v2){
////////////////////////////////////////////////////////////////////////////
/* POR HACER: codigo para el encriptado de 2 pixeles                      */
////////////////////////////////////////////////////////////////////////////
int matriz[2][2] ={{21,35},{18,79}};
int aux=matriz[0][0]*(*v1)+matriz[0][1]*(*v2);
int aux2= matriz[1][0]*(*v1)+matriz[1][1]*(*v2);
*v1=(unsigned char)(aux%256);
*v2=(unsigned char) (aux2%256);
}
void generar_pagina_encriptada(pagina in_page, pagina *out_page){
  generar_pagina(out_page,in_page.h,in_page.w,NEGRO);
 //Copiar la pagina en out_page
  for(int i=0; i<in_page.h; i++){
   for(int j=0; j<in_page.w; j++){
    out_page->im[i][j]= in_page.im[i][j];
   }
  }
////////////////////////////////////////////////////////////////////////////
/* POR HACER: codigo para generar la pagina encriptada     */
////////////////////////////////////////////////////////////////////////////
 for (int i=0; i<in_page.h; i=i+1){
  for (int j=0; j<in_page.w; j=j+2){
   encriptar_pixeles(&(out_page->im[i][j]),&(out_page->im[i][j+1]));
  }
 }
}
```

preparar_subida_serie.c: genera la imagen pixel por pixel.

```
/* Prepara la subida de la pagina a la nube                               */
/************************************************************************/
void preparar_subida(pagina in_page){
for(int i=0; i<in_page.h; i++){
 for (int j=0; j<in_page.w; j++){
  upload(in_page.im[i][j], i, j, in_page.h, in_page.w);
 }
}
}
```

pagina_s.c: en la versión serie no se ha hecho ninguna modificación. En este fichero se encuentra el programa principal encargado de llamar a las funciones para tratar una imagen. Para su ejecución es necesario pasarle el LibroX.pgm y el valor límite de filtrado como argumentos.

Finalmente, en la versión serie se ha creado un ejecutable, el cual compila todos los ficheros necesarios para la creación del ejecutable de la aplicación en versión serie. Código del ejecutable de compilación:

gcc filter_serie.c encript_serie.c preparar_subida_serie.c pagina_s.c pixmap.o upload.o -o ej_serie -lm

❖ Versión paralela

Una vez completados todos los ficheros.c para la versión serie, se han creado copias para poder ser modificadas adecuadamente para una versión en paralelo, creando secciones paralelas en el código.

filter_parallel.c: se han paralelizado los bucles de las funciones copiar_pagina y filtrar_pagina.

Como se puede ver en la imagen a continuación, en copiar_pagina se crea una sección paralela (#pragma omp…). Hemos elegido una planificación estática porque el tiempo de ejecución es el mismo en todas las iteraciones, y así nos ahorramos ese tiempo extra que se necesitaría para sincronizar todos los hilos en una planificación dinámica. Hemos optado por repartir el trabajo en bloques de tamaño 2, pero elegir otro diferente tampoco supondría una diferencia significativa. Para que los índices " i " y " j " no sufran modificaciones hasta que ese hilo termine de usarlo optamos por privatizar (explicado más en profundidad en el proximo parrafo) estas variables. Hemos seguido esta misma planificación en todas las funciones auxiliares para copiar las páginas.

```
void copiar_pagina(pagina *in_page, pagina *out_page){
int i,j;
#pragma omp parallel for private(i,j) schedule(static,2)
        for (i=0;i<in_page->h;i++){
                for (j=0;j<in_page->w;j++) out_page->im[i][j]=in_page->im[i][j];
        }//for
}
```

Igualmente en filtrar_pagina se crea una sección paralela con las características de una planificación igual a la de copiar_pagina, con la diferencia de que esta vez los bloques son de tamaño 4. Al paralelizar la imagen, la repartimos entre los diferentes hilos. Con el uso de la cláusula private y reduction, solventamos el problema de que se puedan sobrescribir los datos e índices. Es decir, en el caso de por ejemplo dos hilos escriban en min 3 y otro 5 a la vez, podría suceder que el dato guardado sea el 5 borrando así el 3 y dando un resultado no deseado. Lo que hacen las clausulas private y reduction para solventar esto es privatizar las variables. Esto consiste en que cada hilo tiene su propia variable (*min, tot, i, j* y *px*), independiente de la del resto de hilos, donde cada hilo almacena el píxel mínimo de se trozo de la imagen, la suma de su trozo, etc. En el caso de min lo que queremos es obtener el píxel con el valor más pequeño de toda la imagen, y en el caso de tot la suma de los pixeles de toda la imagen. Para conseguir esto reduction se encarga de al final del for (en este caso) sumar las sumas de cada trozo de la imagen, en

nuestra variable tot. Y almacenar en nuestra variable min el valor más pequeño de los trozos, obteniendo así los resultados que buscábamos.

```
double filtrar_pagina(pagina *in_page, pagina *out_page){
int min=400;
double avg=0;
long tot=0;
int px=0;
int i,j;
#pragma omp parallel for private(i,j,px) reduction(min:min) reduction(+:tot) schedule(static,4)
   for(i=1;i<in_page->h-1;i++){
        for(j=1;j<in_page->w-1;j++){
                px=get_px(in_page,i,j);
                out_page->im[i][j]=px;
                if(px<min)min=px;
                tot+=px;
        }//for
  }//for
avg=(double)tot/((in_page->h-2)*(in_page->w-2));
// printf("Average = %.2f, Tot= %ld and Min = %d\n",avg,tot,min);
return avg-min;
}//filtrar_pagina
```

encript_parallel.c: se ha paralelizado la función generar_pagina_encriptada según las necesidades de los bucles. Aquí, seguimos aplicando una planificación estática por las mismas razones que antes. Pero en este caso no se trata de un reparto entrelazado sino consecutivo.

```
void generar_pagina_encriptada(pagina in_page, pagina *out_page)}
  generar_pagina(out_page,in_page.h,in_page.w,NEGRO);
 //Copiar la pagina en out_page
#pragma omp parallel
{
 int nth= omp_get_num_threads();
 int i,j;
#pragma omp for private(i,j) schedule(static,in_page.h/nth)
  for(i=0; i<in_page.h; i++)
   for(j=0; j<in_page.w; j++)
    out_page->im[i][j]= in_page.im[i][j];
#pragma omp for private(i,j) schedule(static,in_page.h/nth)
 for (i=0; i<in_page.h; i=i+1)
  for (j=0; j<in_page.w; j=j+2)
   encriptar_pixeles(&(out_page->im[i][j]),&(out_page->im[i][j+1]));
 }//pragma omp parallel
}//generar_pagina
```

preparar_subida_parallel.c: se ha paralelizado la función preparar_subida de forma dinámica porque ahora sí que el tiempo de ejecución de cada iteración varía dependiendo del píxel. Cada iteración tendrá un tiempo de ejecución diferente y por eso es mejor optar por una planificación dinámica, para repartir el trabajo de manera más equitativa.

```
void preparar_subida(pagina in_page){
#pragma omp parallel
{
int i,j;
#pragma omp for private(i,j) schedule(dynamic,1)
for(i=0; i<in_page.h; i++){
 for (j=0; j<in_page.w; j++){
  upload(in_page.im[i][j], i, j, in_page.h, in_page.w);
 }
}
}//omp parallel
}
```

pagina_p.c: únicamente se ha modificado el texto a la hora de imprimir por pantalla, de manera que en vez de imprimirse “serie” ahora se imprime “paralelo”. También hemos modificado el nombre de los libros creados para que acaben en “_par.pgm” en vez de “_ser.pgm”.

Finalmente, para la compilación y creación del ejecutable versión paralelo, se ha creado un archivo ejecutable. Código del ejecutable:

gcc -fopenmp filter_parallel.c encript_parallel.c preparar_subida_parallel.c pagina_p.c pixmap.o upload.o -o ej_parallel -lm

RESULTADOS

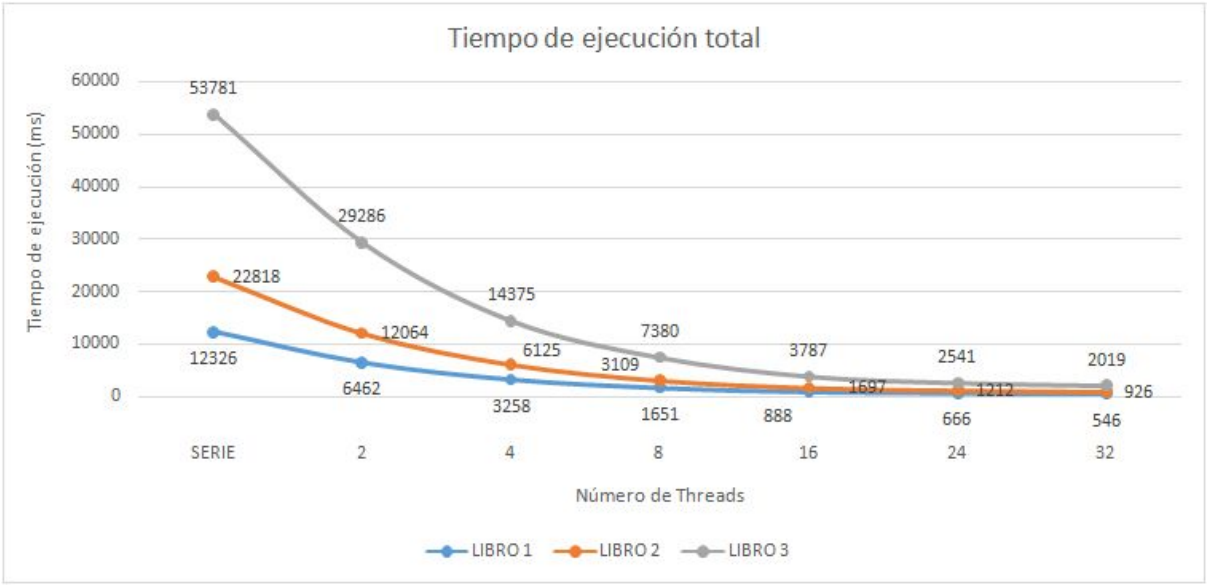
Una vez compilado y ejecutado todo lo anterior, hemos obtenido los siguientes tiempos de ejecución mostrados a continuación. Los resultados son diferenciados en tres tablas, una para cada libro, y en cada tabla se muestra el tiempo (filtrado, cifrado, subida y total) de ejecución en serie y en paralelo para 2, 4, 8, 16, 24 y 32 hilos. Para visualizar y descargar el excel con todos los resultados, click aquí.

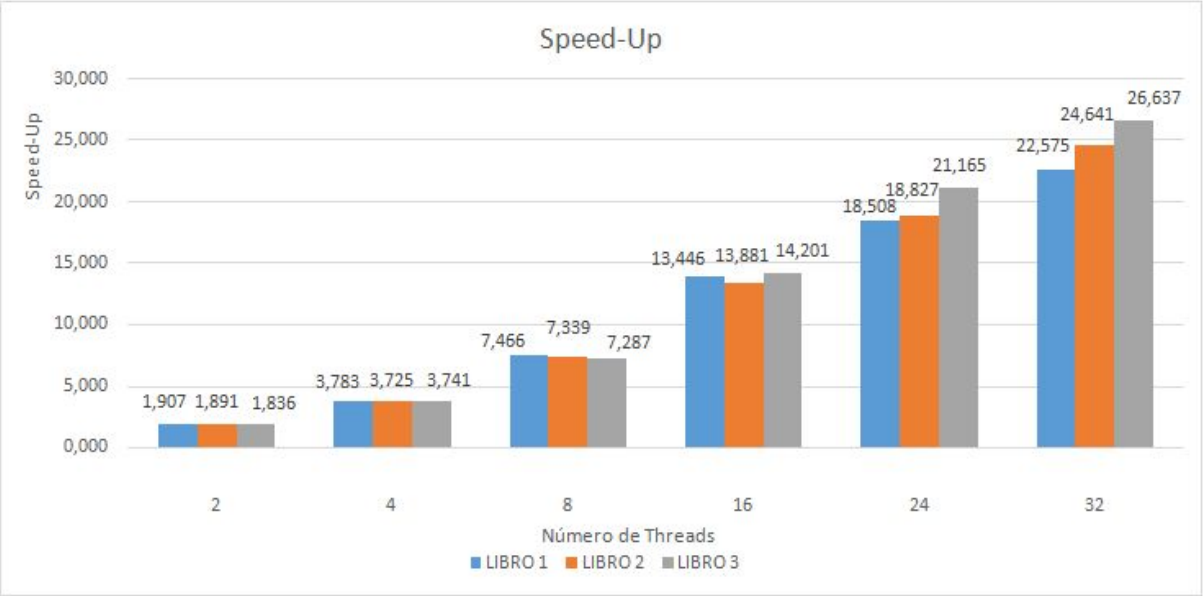
		SERIE	PARALELO					
LIBRO 1			2 hilos	4 hilos	8 hilos	16 hilos	24 hilos	32 hilos
Tiempo (ms)	Filtrar	11380	5930	2990	1510	815	612	500
	Cifrar	237	152	78	43	25	21	17
	Subir Pagina	709	380	190	98	48	33	29
	TOTAL (main)	12326	6462	3258	1651	888	666	546

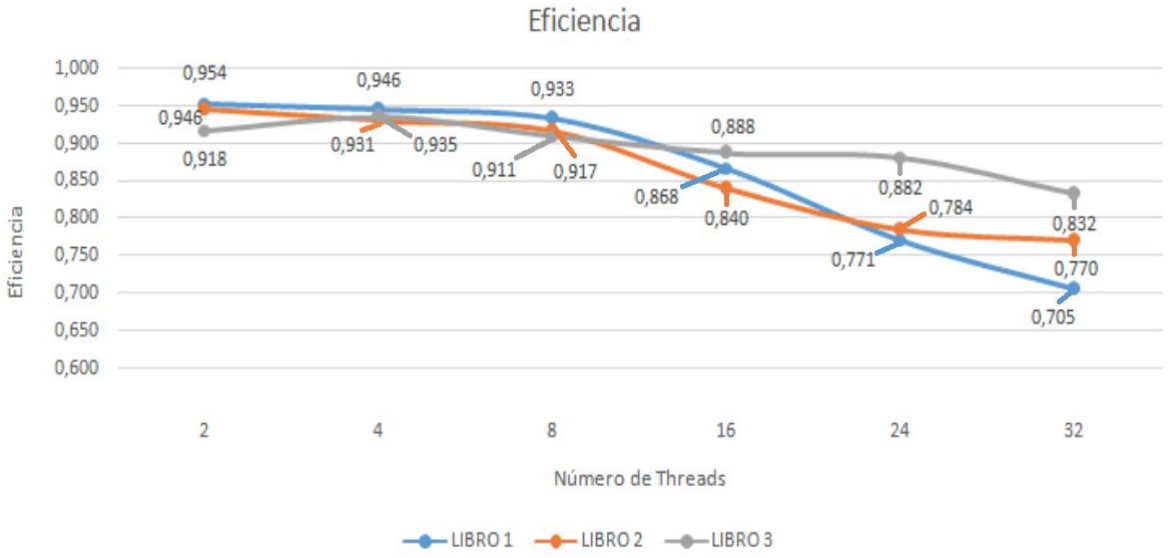
		SERIE	PARALELO					
LIBRO 2			2 hilos	4 hilos	8 hilos	16 hilos	24 hilos	32 hilos
Tiempo (ms)	Filtrar	21082	11086	5630	2858	1564	1115	850
	Cifrar	445	281	148	77	45	38	28
	Subir Pagina	1291	697	347	174	88	59	48
	TOTAL (main)	22818	12064	6125	3109	1697	1212	926

		SERIE	PARALELO					
LIBRO 3			2 hilos	4 hilos	8 hilos	16 hilos	24 hilos	32 hilos
Tiempo (ms)	Filtrar	49311	26769	13084	6732	3446	2304	1830
	Cifrar	1138	718	370	198	113	87	74
	Subir Pagina	3332	1799	921	450	228	150	115
	TOTAL (main)	53781	29286	14375	7380	3787	2541	2019

Partiendo de los datos anteriores, se muestran tres gráficos (Tiempo de ejecución, speed-up y eficiencia) para las ejecuciones en paralelo. En el primer gráfico podemos observar la disminución del tiempo de ejecución (ms) con el aumento de número de hilos. En el segundo podemos observar el factor aceleración conseguido al paralelizar la aplicación. Finalmente, en el tercer gráfico, se puede observar la eficiencia obtenida según el número de threads en tanto por ciento (%), desde la cual podemos obtener conclusiones para el siguiente apartado.







CONCLUSIONES

Observando los gráficos, podemos concluir que con una ejecución en paralelo, siempre obtendremos una ganancia de tiempo. La cuestión es, en este caso, cuál sería el número de hilos más adecuado para una resultado satisfactorio. Podremos lograr un buen resultado con una ejecución en 8 hilos, debido a que es una eficiencia aceptable antes de su caída con el aumento de hilos y un tiempo de ejecución equivalente alrededor de ¼ del tiempo de ejecución en serie para los libros 1 y 2, y alrededor de $^{14}/_{100}$ para el libro 3. En ningún caso sugeriríamos la ejecución paralela en menos de 4 y más de 16 hilos.

INFORMACIÓN ADICIONAL

Para el cifrado utilizamos una matriz clave [21, 35; 18, 79] y su inversa [79, -35; -18, 21] para el descifrado. Toda la información se encuentra en la bibliografía a continuación.

BIBLIOGRAFÍA

- Cifrado de Hill
- Filtrado de imágenes