MEMORIA P2 CPA

ISMAEL FERNÁNDEZ HERRERUELA

EJERCICIO 1

En este primer ejercicio se nos solicitaba realizar la paralelización de la parte 1 y la parte 3 ya que la 2 no era posible paralelizarla.

```
101  // Part 1. Find optimal offset of each line with respect to the previous line
102
103  for ( y = 1 ; y < h ; y++ ) {
104
105    // Find offset of line y that produces the minimum distance between lines y and y-1
106    dmin = distance( w, &a[3*(y-1)*w], &a[3*y*w], INT_MAX ); // offset=0
107    bestoff = 0;
108    #pragma omp parallel for private (d)schedule(runtime)
109    for ( off = 1 ; off < w ; off++ ) {
110      d  = distance( w-off, &a[3*(y-1)*w], &a[3*(y*w+off)], dmin );
111      d += distance( off, &a[3*(y*w-off)], &a[3*y*w], dmin-d );
112      // Update minimum distance and corresponding best offset
113      #pragma omp critical
114      if ( d < dmin ) { dmin = d; bestoff = off; }
115    }
116    voff[y] = bestoff;
117  }
```

Al realizar la paralelización de la primera parte, utilizamos la d como variable privada ya que como con cada hilo se modificaría, no queremos que eso ocurra.

```
128    // Part 3. Shift each line to its place, using auxiliary buffer v
129    #pragma omp parallel private(v)
130    {
131    v = malloc( 3 * max * sizeof(Byte) );
132    if ( v == NULL )
133      fprintf(stderr,"ERROR: Not enough memory for v\n");
134    else {
135      #pragma omp for schedule(runtime)
136      for ( y = 1 ; y < h ; y++ ) {
137        cyclic_shift( w, &a[3*y*w], voff[y], v );
138      }
139
140      free(v);
141    }}
142
143    free(voff);
144  }
```

En la parte 3 ocurre lo mismo, queremos que la variable v se utilice como privada para que no se modifique con cada hilo. Y dentro del else utilizamos un pragma omp for para cada iteración del bucle.

EJERCICIO 2

En el segundo ejercicio se nos pedía realizar la paralelización mediante una única región paralela, paralelizar el bucle externo de la primera parte y el bucle de la tercera parte.

```
101   #pragma omp parallel private(dmin, d, off, bestoff, v)
102   {
103   #pragma omp for schedule(runtime)
104   // Part 1. Find optimal offset of each line with respect to the previous line
105   for ( y = 1 ; y < h ; y++ ) {
106
107     // Find offset of line y that produces the minimum distance between lines y and y-1
108     dmin = distance( w, &a[3*(y-1)*w], &a[3*y*w], INT_MAX ); // offset=0
109     bestoff = 0;
110     for ( off = 1 ; off < w ; off++ ) {
111       d  = distance( w-off, &a[3*(y-1)*w], &a[3*(y*w+off)], dmin );
112       d += distance( off, &a[3*(y*w-off)], &a[3*y*w], dmin-d );
113       // Update minimum distance and corresponding best offset
114       //#pragma omp critical
115       if ( d < dmin ) { dmin = d; bestoff = off; }
116     }
117     voff[y] = bestoff;
118   }
119   #pragma omp single
120   {
121   // Part 2. Convert offsets from relative to absolute and find maximum offset of any line
122   max = 0;
123   voff[0] = 0;
124
125
126   for ( y = 1 ; y < h ; y++ ) {
127     voff[y] = ( voff[y-1] + voff[y] ) % w;
128     d = voff[y] <= w / 2 ? voff[y] : w - voff[y];
129     if ( d > max ) max = d;
130   }
131   }
132
133   // Part 3. Shift each line to its place, using auxiliary buffer v
134
135   v = malloc( 3 * max * sizeof(Byte) );
136
137   if ( v == NULL )
138     fprintf(stderr,"ERROR: Not enough memory for v\n");
139   else {
140   #pragma omp for schedule(runtime)
141     for ( y = 1 ; y < h ; y++ ) {
142       cyclic_shift( w, &a[3*y*w], voff[y], v );
143     }
144     }
145     free(v);
146   }
147
148   free(voff);
149
150 }
```

En primer lugar, privatizamos las variables dmin, d, off, bestoff y v para que cada hilo obtenga su copia local. Acto seguido paralelizamos el bucle for externo de la primera parte. A continuación, utilizamos un pragma omp single porque debido a que la segunda parte no se puede paralelizar, la ejecutamos una sola vez. Y en ultimo lugar paralelizamos el bucle for de la parte 3.

EJERCICIO 3

La planificación Dynamic debería de ofrecer mejores resultados. Esto se debe a como realiza la asignación de hilos en comparación con la static. Si por ejemplo realizamos una planificación static con chunk 4, el primer hilo haría las 4 primeras iteraciones, el segundo hilo haría las 4 siguientes y así sucesivamente, lo cual puede llevarnos a unos tiempos de ejecución mas elevados. Mientras que por otra parte la planificación Dynamic va asignando hilos según se van necesitando, de manera que mientras que una ejecución dura 5 segundos, hay otras 3 ejecuciones que duran 1 segundo cada una, entonces al distribuir los hilos de manera dinámica, si tuviésemos 4 hilos, en un hilo realizaríamos la de 5 y con los otros 3 hariamos las 3 ejecuciones de 1 segundo, por lo que el tiempo total de ejecución seria de 5 segundos ya que es el hilo que mas ha tardado en ejecutarse. En el caso de static, tuviésemos un hilo por cada 4 ejecuciones, tardaría 8 segundos ya que primero haría la de 5 segundos, después una de 1 segundo y así hasta acabar.

EJERCICIO 4

Dynamic default

```
iferher@alumno.upv.es@kahan:~/prac2$ sbatch realign.sh
Submitted batch job 45388
iferher@alumno.upv.es@kahan:~/prac2$ cat slurm-45388.out
Tiempo: 0.118996
```

Static default

```
iferher@alumno.upv.es@kahan:~/prac2$ sbatch realign.sh
Submitted batch job 45392
iferher@alumno.upv.es@kahan:~/prac2$ cat slurm-45392.out
Tiempo: 0.124232
```

Static Chunk 1

```
iferher@alumno.upv.es@kahan:~/prac2$ sbatch realign.sh
Submitted batch job 45394
iferher@alumno.upv.es@kahan:~/prac2$ cat slurm-45394.out
Tiempo: 0.122920
```

Como podemos observar, la planificación Dynamic es significativamente mas rápida que las demás, esto es debido a que realiza la asignación de hilos en tiempo de ejecución y va asignando según se vayan liberando los hilos de manera automática. Por lo tanto reduce el tiempo de ejecución ya que en lo que acaba la ejecución de un hilo que tarda mas tiempo, se habrán realizado ejecuciones de otros hilos que tarden menos tiempo, a

diferencia del static que asigna los hilos sin tener en cuenta el tiempo y esto resulta en un tiempo mayor de ejecución.

EJERCICIO 5

REALIGN 2

Tiempos con planificación dinámica:

```
Tiempo: 2.319331
Tiempo: 1.169998
Tiempo: 0.592403
Tiempo: 0.301482
Tiempo: 0.182180
Tiempo: 0.109495
Tiempo: 0.085973
Tiempo: 0.087185
Tiempo: 0.092736
```

Tiempos con planificación estática:

```
Tiempo: 2.319420
Tiempo: 1.169671
Tiempo: 0.591139
Tiempo: 0.303771
Tiempo: 0.186317
Tiempo: 0.117588
Tiempo: 0.087288
Tiempo: 0.084566
Tiempo: 0.092803
```

Tiempos con planificación estática 1 chunk:

```
Tiempo: 2.321531
Tiempo: 1.170019
Tiempo: 0.591260
Tiempo: 0.302651
Tiempo: 0.186862
Tiempo: 0.107479
Tiempo: 0.083285
Tiempo: 0.085123
Tiempo: 0.090745
```

REALIGN 1

Tiempos con planificación estática:

```
Tiempo: 2.336784
Tiempo: 1.218594
Tiempo: 0.622388
Tiempo: 0.325196
Tiempo: 0.203075
Tiempo: 0.193146
Tiempo: 0.311348
Tiempo: 0.441337
Tiempo: 0.640543
```

Tiempos con planificación estática 1 chunk:

```
Tiempo: 2.337146
Tiempo: 1.218076
Tiempo: 0.620112
Tiempo: 0.324602
Tiempo: 0.202252
Tiempo: 0.190725
Tiempo: 0.316490
Tiempo: 0.432702
Tiempo: 0.604299
```

Tiempos con planificación dinámica:

```
Tiempo: 2.337237
Tiempo: 1.217714
Tiempo: 0.622673
Tiempo: 0.325206
Tiempo: 0.204294
Tiempo: 0.189843
Tiempo: 0.316267
Tiempo: 0.442018
Tiempo: 0.644545
```

Utilizando en potencias de 2, desde 1 hasta 256 hilos. Como podemos observar, a partir de los 64-128 hilos vuelve a incrementar el tiempo y esto es debido a que se realizan tantas creaciones de hilos y se distribuyen tanto las tareas que aumenta el tiempo. No he realizado pruebas con mas hilos debido a que ya se comprueba que a partir de los 64-128 vuelve a subir el tiempo.

Para calcular el speedup y la eficiencia obtuve el tiempo que tarda sin paralelizar el cual es el siguiente: `Tiempo: 3.642433`.

REALIGN1

TIEMPO SIN PARALELIZAR =	3,642433			
REALIGN 1 ESTATICO				
HILOS	SPEEDUP	EFICIENCIA		
1	1,558738	1		
2	2,989045572	1,49452279		
4	5,852350945	1,46308774		
8	11,20073125	1,40009141		
16	17,93639296	1,12102456		
32	18,85844387	0,58932637		
64	11,69853673	0,18278964		
128	8,25317841	0,06447796		
256	5,686476942	0,0222128		
REALIGN 1 ESTATICO 1 CHUNK				
	SPEEDUP	EFICIENCIA		
1	1,558496	1		
2	2,990317678	1,49515884		
4	5,873830856	1,46845771		
8	11,22122784	1,40265348		
16	18,00937939	1,12558621		
32	19,09782671	0,59680708		
64	11,50884072	0,17982564		
128	8,417878817	0,06576468		
256	6,027534383	0,02354506		
REALIGN 1 DINAMICO				
	SPEEDUP	EFICIENCIA		
1	1,558435	1		
2	2,991205653	1,49560283		
4	5,8496723	1,46241807		
8	11,20038683	1,40004835		
16	17,82936846	1,11433553		
32	19,18655415	0,59957982		
64	11,51695561	0,17995243		
128	8,240463058	0,06437862		
256	5,65116943	0,02207488		

REALIGN2

TIEMPO SIN PARALELIZAR =	3,642433			
REALIGN 2 ESTATICO				
HILOS	SPEEDUP	EFICIENCIA		
1	1,570407	1		
2	3,114066263	1,55703313		
4	6,161720002	1,54043		
8	11,99071998	1,49884		
16	19,54965462	1,22185341		
32	30,97623057	0,96800721		
64	41,7289089	0,6520142		
128	43,07207388	0,33650058		
256	39,24908678	0,15331675		
REALIGN 2 ESTATICO 1 CHUNK				
	SPEEDUP	EFICIENCIA		
1	1,568979	1		
2	3,113140043	1,55657002		
4	6,16045902	1,54011475		
8	12,03509323	1,50438665		
16	19,49263628	1,21828977		
32	33,88971799	1,05905369		
64	43,73456205	0,68335253		
128	42,79023296	0,33429869		
256	40,13921428	0,15679381		
REALIGN 2 DINAMICO				
	SPEEDUP	EFICIENCIA		
1	1,570467	1		
2	3,11319592	1,55659796		
4	6,148572847	1,53714321		
8	12,08175944	1,51021993		
16	19,99359425	1,24959964		
32	33,26574729	1,0395546		
64	42,36717341	0,66198708		
128	41,77820726	0,32639224		

En conclusión, observando todos los resultados, speedups y eficiencias, podemos confirmar que la segunda versión es mas rápida y mas eficiente.

Para lanzarlo en el cluster he utilizado un archivo .sh con el siguiente contenido:

```
#!/bin/bash

#SBATCH --nodes=1

#SBATCH --time=5:00

#SBATCH --partition=cpa
OMP_SCHEDULE=dynamic
OMP_NUM_THREADS=1 ./realign1 small.ppm small1.ppm
OMP_NUM_THREADS=2 ./realign1 small.ppm small1.ppm
OMP_NUM_THREADS=4 ./realign1 small.ppm small1.ppm
OMP_NUM_THREADS=8 ./realign1 small.ppm small1.ppm
OMP_NUM_THREADS=16 ./realign1 small.ppm small1.ppm
OMP_NUM_THREADS=32 ./realign1 small.ppm small1.ppm
OMP_NUM_THREADS=64 ./realign1 small.ppm small1.ppm
OMP_NUM_THREADS=128 ./realign1 small.ppm small1.ppm
OMP_NUM_THREADS=256 ./realign1 small.ppm small1.ppm
```

Esa es solo la primera versión de la paralelización pero para la segunda versión simplemente habría que cambiar ./realign1 por ./realign2 y para cambiar la planificación, en la sentencia de OMP_SCHEDULE por dynamic, static o static,1.