Práctica 1

Sesión 2

Objetivos

- Implementación paralela del filtrado de una imagen mediante OpenMP
- Paralelización de varios bucles anidados
  - Determinación del ámbito de las variables que aparecen en la región paralela:
    - Compartidas
    - Privadas
    - Reducciones
- Dependencia entre iteraciones

Programa de filtrado de imágenes

- Recuerda que en la 1ª sesión de la 1ª práctica almacenaste en una carpeta (material) de la unidad **W** todos los programas y ficheros en C necesarios para realizar la 1ª práctica

- En imagenes.c tienes la implementación secuencial del programa de filtrado.
- Los ficheros peppers.ppm y peppers-1k.ppm contienen la misma imagen, la segunda con una mayor resolución que la primera. Esta segunda imagen la usarás cuando quieras hacer pruebas con la máquina paralela de Kahan (lanzar trabajos a la cola de **Kahan**)
- PPM es un formato basado en texto, el cual puede ser visualizado por diferentes programas, como irfanview1 o display.
- Ejemplo:

```
P3        <- Cadena constante que indica el formato (ppm, color RGB)
512 512   <- Dimensiones de la imagen (número de columnas y número de filas)
255       <- Mayor nivel de intensidad
224 137 125 225 135 ... <- 512x512x3 valores. Cada punto son tres valores consecutivos (R,G,B)
```

Programa de filtrado de imágenes

Las primeras líneas del fichero imagenes.c son las siguientes:

```
1   #include <stdio.h>
2   #include <string.h>
3   #include <stdlib.h>
4
5   #define max(a, b) ((a) > (b) ? (a) : (b))
6   #define min(a, b) ((a) < (b) ? (a) : (b))
7   #define MAXCAD 100
8
9   #define NUM_PASOS 5
10  #define DIST_RADIO 8
11  #define IMAGEN_ENTRADA "peppers.ppm"
12  #define IMAGEN_SALIDA "peppers-fil.ppm"
```

- Puedes cambiar los nombres de los ficheros imagen de entrada y salida en las líneas 11 y 12.
- El **número de pasos** y el **radio** que se utilizarán en el procesamiento de la imagen se encuentran definidos en las líneas 9 y 10. Los puedes modificar para que sean más costosas las ejecuciones
- La reserva de memoria la realiza la función de lectura de la imagen lee_ppm, garantizando que todos los pixeles de la imagen se encuentran consecutivos

Programa de filtrado de imágenes

- Para comparar los tiempos de ejecución de los códigos secuenciales y paralelos es conveniente utilizar la misma función para la toma de tiempos; utilizando para ello la función **omp_get_wtime** vista ya en la sesión 1 de esta práctica
- Modifica el código de **imagenes.c**, añadiendo las líneas que aparecen a continuación:

```
156    double t=omp_get_wtime();
157    rc = Filtro(NUM_PASOS, DIST_RADIO, ImgOrg, ImgDst, n, m);
158    t=omp_get_wtime()-t;
159    printf("Tiempo de ejecución código secuencial %f segundos\n",t);
```

- Recuerda añadir al principio del código del fichero **imagenes.c**:

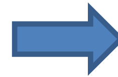
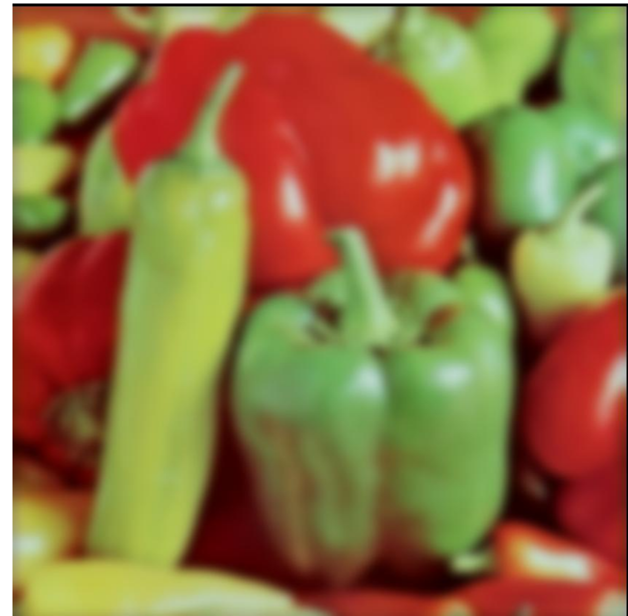
  #include <omp.h>

- Compílalo y ejecútalo tal como se muestra a continuación:

```
jjibanez@upvnet.upv.es@ldsic-vdi08:~/prac_cpa/prac1/s2/codigos$  gcc -fopenmp -o imagenes imagenes.c
jjibanez@upvnet.upv.es@ldsic-vdi08:~/prac_cpa/prac1/s2/codigos$ ./imagenes
Abierta una imagen de n:512, m:512
Tiempo de ejecucion secuencial 3.544237 segundos
```

Programa de filtrado de imágenes

- Al ejecutar el programa, obtendrás la imagen filtrada:

- La imagen filtrada se almacenará en el fichero **peppers-fil.ppm**. Este fichero se utilizará para comprobar el correcto funcionamiento de los códigos paralelos que se van a desarrollar.
- Como los ficheros de imagen en formato PPM son de texto plano, se pueden visualizar con los comandos head, more o less.

Ámbito correcto de las variables en una región paralela (RP)

- Privadas:
  - La variable iteradora del bucle **for** que se paraleliza es **privada** de facto
  - Normalmente las variables que aparezcan a la izquierda de una asignación o varíen su valor en la RP son **privadas**
- Compartidas
  - Cuando una variable debe ser actualizada en la RP por todos los hilos, será compartida. En la 4ª sesión de prácticas usaremos las directivas **critical** o **atomic** para evitar la condición de carrera
  - Si se trata de un array y los hilos van a modificar diferentes componentes del array, entonces el array debe será **compartido** (no hay condición de carrera)
- Reducciones en bucles paralelos for
  - Una variable será una reducción en un **bucle for** si la variable se inicializa antes del **bucle for** y dicha variable se actualiza dentro del bucle como suma, producto, máximo o mínimo de términos que dependen de la variable iteradora del bucle paralelo **for**

$$s = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} A_{ij}$$

```
int i, j;
double s=0.0, A[N][N];
for (i=0; i<N; i++)
    for (j=0; j<N; j++)
        s=s+A[i][j];
```

Los dos bucles se pueden paralelizar, pero solo uno a la vez:

Bucle i:

#pragma parallel for reduction (+: s) private(i,j)

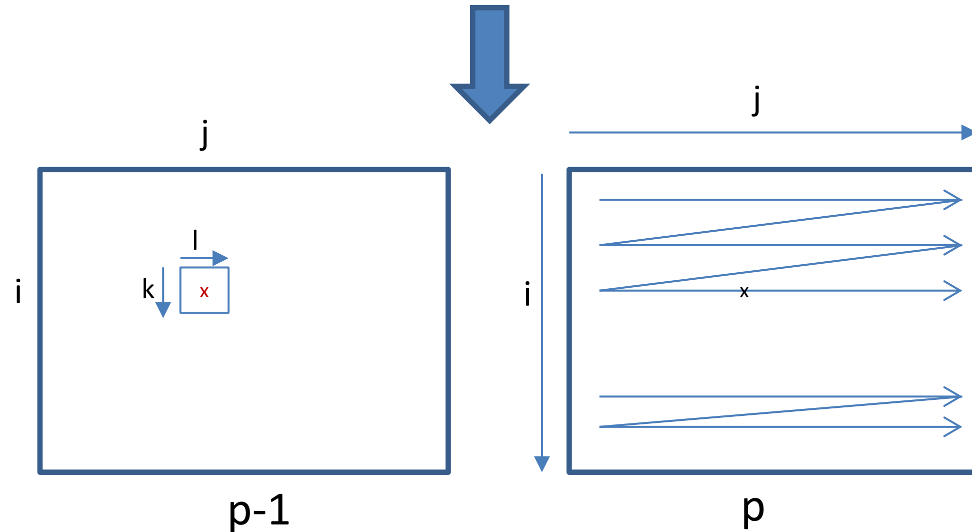
Bucle j:

#pragma parallel for reduction (+: s) private(j)

Programa de filtrado de imágenes

```
for (p = 0; p < pasos; p++) {
  for (i = 0; i < n; i++) {
   for (j = 0; j < m; j++) {
    resultado.r = 0;
    resultado.g = 0;
    resultado.b = 0;
    tot = 0;
    for (k = max(0, i - radio); k <= min(n - 1, i + radio); k++) {
     for (l = max(0, j - radio); l <= min(m - 1, j + radio); l++) {
      v = ppdBloque[k - i + radio][l - j + radio];
      resultado.r += ppsImagenOrg[k][l].r * v;
      resultado.g += ppsImagenOrg[k][l].g * v;
      resultado.b += ppsImagenOrg[k][l].b * v;
      tot += v;
     }
    }
    resultado.r /= tot;
    resultado.g /= tot;
    resultado.b /= tot;
    ppsImagenDst[i][j].r = resultado.r;
    ppsImagenDst[i][j].g = resultado.g;
    ppsImagenDst[i][j].b = resultado.b;
   }
  }
}
```

- El bucle anidado i-j se utiliza para calcular la matriz de colores en la etapa **p**, a partir de los pixeles de los colores vecinos de x (DIST_RADIO) en la iteración anterior **p-1**:



El procesamiento de la imagen se realiza paso a paso mediante el bucle externo, de manera que la imagen en la iteración **p** se obtiene a patir de la imagen en la iteración **p-1**

$$imagen_0 \rightarrow imagen_1 \rightarrow \ldots\ldots \rightarrow imagen_{p-1} \rightarrow imagen_p \ldots\ldots \rightarrow imagen_{pasos-1}$$

Programa de Filtrado de imágenes

- El filtrado se basa en la media ponderada:
  - El color de cada pixel viene dado por tres enteros comprendidos entre 0 y 255 (uno para cada color básico)
  - El color de un pixel de una imagen se obtiene como promedio de los valores del mismo color correspondientes a los pixels vecinos de la imagen anterior, con un cierto radio (constante DIST_RADIO). Este cálculo se realiza en los bucles **k** y **l**

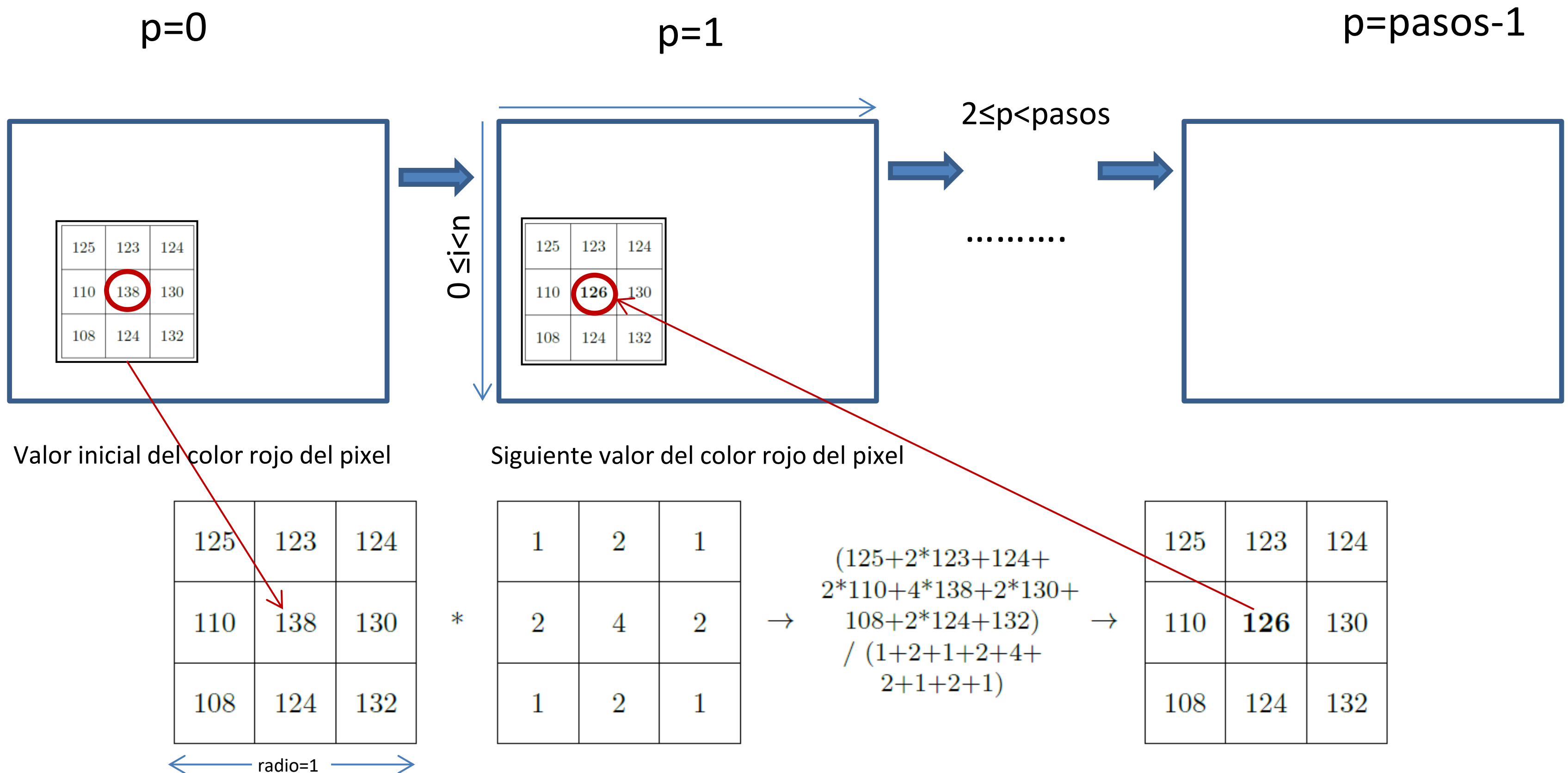


Figura 3: Modelo de aplicación de una media ponderada en el filtrado de imágenes.

Códigos paralelos

- Para cada bucle **for** que sea paralelizable, deberás obtener una implementación paralela (no paralelices a la vez dos bucles for anidados), utilizando la directiva parallel for y las clausulas necesarias para el correcto funcionamiento (private, reduction, etc….)

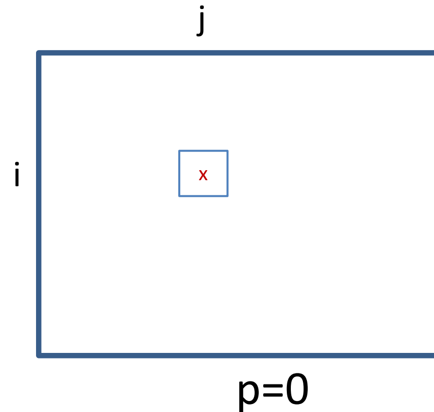
```
109  for (p = 0; p < pasos; p++) {
110    for (i = 0; i < n; i++) {
111      for (j = 0; j < m; j++) {
112        resultado.r = 0;
113        resultado.g = 0;
114        resultado.b = 0;
115        tot = 0;
116        for (k = max(0, i - radio); k <= min(n - 1, i + radio); k++) {
117          for (l = max(0, j - radio); l <= min(m - 1, j + radio); l++) {
118            v = ppdBloque[k - i + radio][l - j + radio];
119            resultado.r += ppsImagenOrg[k][l].r * v;
120            resultado.g += ppsImagenOrg[k][l].g * v;
121            resultado.b += ppsImagenOrg[k][l].b * v;
122            tot += v;
123          }
124        }
125        resultado.r /= tot;
126        resultado.g /= tot;
127        resultado.b /= tot;
128        ppsImagenDst[i][j].r = resultado.r;
129        ppsImagenDst[i][j].g = resultado.g;
130        ppsImagenDst[i][j].b = resultado.b;
131      }
132    }
```

- Para cada posible paralelización realizarás una copia del fichero imagenes.c llamándolo, por ejemplo, pimagenes-*.c, donde * es sustituido por el nombre de la variable del bucle a paralelizar

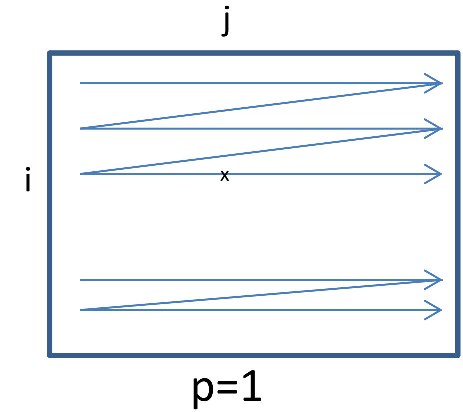
Códigos paralelos: paralelización bucle p

¿Es posible paralelizar el primer bucle for (bucle con variable p)? Justifica la respuesta

```
for (p = 0; p < pasos; p++) {
  for (i = 0; i < n; i++) {
   for (j = 0; j < m; j++) {
    resultado.r = 0;
    resultado.g = 0;
    resultado.b = 0;
    tot = 0;
    for (k = max(0, i - radio); k <= min(n - 1, i + radio); k++) {
     for (l = max(0, j - radio); l <= min(m - 1, j + radio); l++) {
      v = ppdBloque[k - i + radio][l - j + radio];
      resultado.r += ppsImagenOrg[k][l].r * v;
      resultado.g += ppsImagenOrg[k][l].g * v;
      resultado.b += ppsImagenOrg[k][l].b * v;
      tot += v;
     }
    }
    resultado.r /= tot;
    resultado.g /= tot;
    resultado.b /= tot;
    ppsImagenDst[i][j].r = resultado.r;
    ppsImagenDst[i][j].g = resultado.g;
    ppsImagenDst[i][j].b = resultado.b;
   }
  }
```



p=0



p=1

Los colores del pixel x en la iteración p=1 se obtienen a partir de los pixeles de los colores vecinos de x en la iteración anterior p=0

En general: los colores del pixel x en la iteración **p** se obtienen a partir de los pixeles de los colores vecinos de x en la iteración anterior **p-1**

Código paralelo: ¿es posible paralelizar bucle i?

```
109   for (p = 0; p < pasos; p++) {
110     for (i = 0; i < n; i++) {
111       for (j = 0; j < m; j++) {
112         resultado.r = 0;
113         resultado.g = 0;
114         resultado.b = 0;
115         tot = 0;
116         for (k = max(0, i - radio); k <= min(n - 1, i + radio); k++) {
117           for (l = max(0, j - radio); l <= min(m - 1, j + radio); l++) {
118             v = ppdBloque[k - i + radio][l - j + radio];
119             resultado.r += ppsImagenOrg[k][l].r * v;
120             resultado.g += ppsImagenOrg[k][l].g * v;
121             resultado.b += ppsImagenOrg[k][l].b * v;
122             tot += v;
123           }
124         }
125         resultado.r /= tot;
126         resultado.g /= tot;
127         resultado.b /= tot;
128         ppsImagenDst[i][j].r = resultado.r;
129         ppsImagenDst[i][j].g = resultado.g;
130         ppsImagenDst[i][j].b = resultado.b;
131       }
132     }
```

#pragma omp parallel for ....

i=0,1,...,n-1

Ho
H1
H2

Para una iteración del primer bucle (variable p), el cálculo de los colores RGB de las n filas de las matrices de colores (rgb) de los pixels se reparten entre los hilos (por defecto, cada hilo calcula los colores de los pixeles de su bloque de filas). Para cada valor de p, se activará una región paralela

Códigos paralelos: paralelización bucle i

- Copia el fichero **imagenes.c**, llamándolo, por ejemplo, **pimagenes-i.c**
- Cambia en pimagenes-i.c la línea

  #define IMAGEN_SALIDA "peppers-fil.ppm"

  por

  #define IMAGEN_SALIDA "peppers-fil-i.ppm"

  de esta manera podrás ver posteriormente que la paralelización está bien hecha, comparando los ficheros peppers-fil.ppm y peppers-fil-i.ppm mediante, por ejemplo, los comando **cmp** o **diff**
- En el código de la función **pimagenes-i.c** debes calcular el tiempo de ejecución mediante **omp_get_wtime** y mostrar en pantalla ese tiempo junto con el número de hilos, usando para ello la función **omp_get_num_threads:**

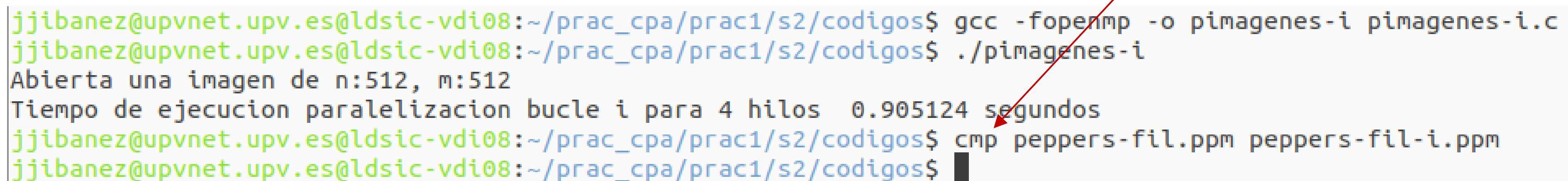
```
int nh;
#pragma omp parallel
{
  nh=omp_get_num_threads();
}
double t=omp_get_wtime();
rc = Filtro(NUM_PASOS, DIST_RADIO, ImgOrg, ImgDst, n, m);
t = omp_get_wtime()-t;
printf("Tiempo de ejecución paralelización bucle i para %d hilos  %f segundos\n", nh, t);
```

Códigos paralelos: paralelización bucle i

- En el código del fichero **pimagenes-i.c** paraleliza el bucle **for** con variable iteradora **i,** mediante la directiva

  #pragma omp parallel for

  poniendo las cláusulas necesarias (private, reduction)
- Compila_y ejecuta el código
- Una manera sencilla de comprobar que ha funcionado correctamente el programa paralelo es usar el comando **cmp** o el comando **diff**, los cuales comparan dos ficheros mostrando las diferencias; si no las hubiese no saldrá nada

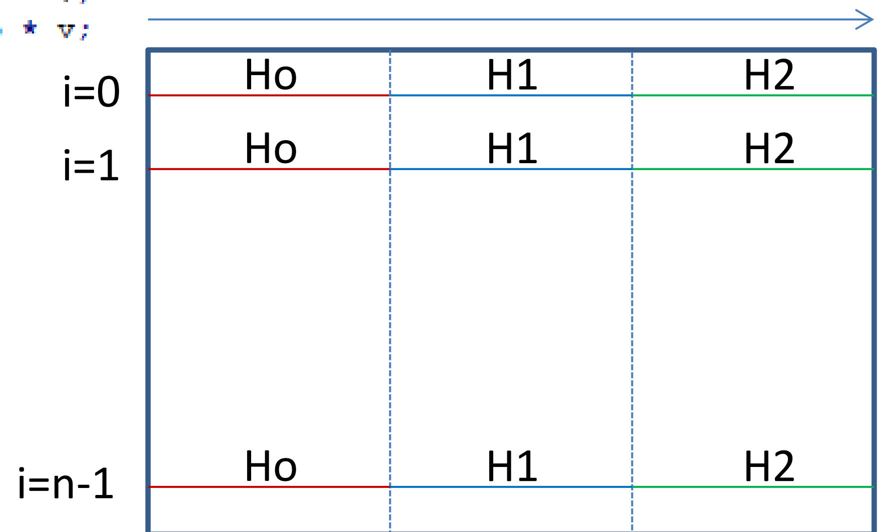


```
jjibanez@upvnet.upv.es@ldsic-vdi08:~/prac_cpa/prac1/s2/codigos$ gcc -fopenmp -o pimagenes-i pimagenes-i.c
jjibanez@upvnet.upv.es@ldsic-vdi08:~/prac_cpa/prac1/s2/codigos$ ./pimagenes-i
Abierta una imagen de n:512, m:512
Tiempo de ejecucion paralelizacion bucle i para 4 hilos  0.905124 segundos
jjibanez@upvnet.upv.es@ldsic-vdi08:~/prac_cpa/prac1/s2/codigos$ cmp peppers-fil.ppm peppers-fil-i.ppm
jjibanez@upvnet.upv.es@ldsic-vdi08:~/prac_cpa/prac1/s2/codigos$
```

Códigos paralelos: paralelización bucle j

```
109  for (p = 0; p < pasos; p++) {
110    for (i = 0; i < n; i++) {
111      for (j = 0; j < m; j++) {
112        resultado.r = 0;
113        resultado.g = 0;
114        resultado.b = 0;
115        tot = 0;
116        for (k = max(0, i - radio); k <= min(n - 1, i + radio); k++) {
117          for (l = max(0, j - radio); l <= min(m - 1, j + radio); l++) {
118            v = ppdBloque[k - i + radio][l - j + radio];
119            resultado.r += ppsImagenOrg[k][l].r * v;
120            resultado.g += ppsImagenOrg[k][l].g * v;
121            resultado.b += ppsImagenOrg[k][l].b * v;
122            tot += v;
123          }
124        }
125        resultado.r /= tot;
126        resultado.g /= tot;
127        resultado.b /= tot;
128        ppsImagenDst[i][j].r = resultado.r;
129        ppsImagenDst[i][j].g = resultado.g;
130        ppsImagenDst[i][j].b = resultado.b;
131      }
132    }
```

#pragma omp parallel for ....



Para cada **i**, se crea una región paralela en donde se reparten las iteraciones de **j** entre los hilos, de manera que cada hilo calcula los pixeles de su parte de fila. Para cada valor de **p** se tendrán **n** regiones paralelas

Códigos paralelos: paralelización bucle j

- Para facilitar la implementación de esta paralelización, puedes copiar el fichero que has creado, pimagenes-i.c, llamándolo ahora pimagenes-j.c
- Elimina o comenta la línea #**pragma omp parallel for** que hayas colocado antes
- Añade la línea **#pragma omp parallel for** antes del bucle **for** con variable iteradora **j**; añadiendo las cláusulas necesarias para su correcto funcionamiento
- Modifica las líneas en las que se muestra el número de hilos para saber que se trata de la implementación paralela del bucle j

```
167    printf("Tiempo de ejecucion paralelizacion bucle j para %d hilos  %f segundos\n",nh,t);
```

- Compila, ejecuta el código y comprueba que ha funcionado correctamente el programa paralelo al usar el comando **cmp** o el comando **diff**

```
jjibanez@upvnet.upv.es@ldsic-vdi08:~/prac_cpa/prac1/s2/codigos$ gcc -fopenmp -o pimagenes-j pimagenes-j.c
jjibanez@upvnet.upv.es@ldsic-vdi08:~/prac_cpa/prac1/s2/codigos$ ./pimagenes-j
Abierta una imagen de n:512, m:512
Tiempo de ejecucion paralelizacion bucle j para 4 hilos  0.988640 segundos
jjibanez@upvnet.upv.es@ldsic-vdi08:~/prac_cpa/prac1/s2/codigos$ cmp peppers-fil.ppm peppers-fil-j.ppm
jjibanez@upvnet.upv.es@ldsic-vdi08:~/prac_cpa/prac1/s2/codigos$
```

Códigos paralelos: paralelización bucle k

```
109  for (p = 0; p < pasos; p++) {
110    for (i = 0; i < n; i++) {
111      for (j = 0; j < m; j++) {
112        resultado.r = 0;
113        resultado.g = 0;
114        resultado.b = 0;
115        tot = 0;
116        for (k = max(0, i - radio); k <= min(n - 1, i + radio); k++) {
117          for (l = max(0, j - radio); l <= min(m - 1, j + radio); l++) {
118            v = ppdBloque[k - i + radio][l - j + radio];
119            resultado.r += ppsImagenOrg[k][l].r * v;
120            resultado.g += ppsImagenOrg[k][l].g * v;
121            resultado.b += ppsImagenOrg[k][l].b * v;
122            tot += v;
123          }
124        }
125        resultado.r /= tot;
126        resultado.g /= tot;
127        resultado.b /= tot;
128        ppsImagenDst[i][j].r = resultado.r;
129        ppsImagenDst[i][j].g = resultado.g;
130        ppsImagenDst[i][j].b = resultado.b;
131      }
132    }
```

#pragma omp parallel for ....

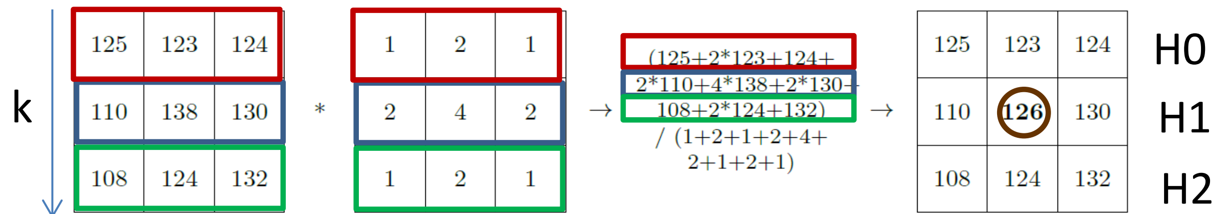


Figura 3: Modelo de aplicación de una media ponderada en el filtrado de imágenes.

Para cada pixel de componentes (i, j), se crea una región paralela en donde se reparten las iteraciones de k entre los hilos, determinando el color del pixel. Total mxn regiones paralelas. Cada hilo calcula las sumas de su parte

Códigos paralelos: paralelización bucle k

- Como la variable **resultado** es de tipo *struct*:

  struct { int r, g, b; } resultado;

  no la puedas utilizar en las cláusulas de tipo **reduction**

  En ese caso, habría que definir tres nuevas variables, por ejemplo, red, green y blue, y usar estas nuevas variables en lugar de la variable resultado (variable de tipo estructura)

```
114    int red = 0;//resultado.r = 0;
115    int green=0;//resultado.g = 0;
116    int blue=0;//resultado.b = 0;
```

  y substituir esas nuevas variables en las apariciones de resultado.r, resultado.g y resultado.b

Códigos paralelos: paralelización bucle k

- Para facilitar la implementación de esta paralelización, puedes copiar el fichero que has creado, pimagenes-j.c, llamándolo ahora pimagenes-k.c
- Elimina o comenta la línea #**pragma omp parallel for** que hayas colocado antes
- Añade la línea **#pragma omp parallel for** antes del bucle for con variable iteradora **k**; añadiendo las cláusulas necesarias para su correcto funcionamiento
- Modifica las líneas en las que se muestra el número de hilos para saber que se trata de la implementación paralela del bucle k

```
166     printf("Tiempo de ejecucion paralelizacion bucle k para %d hilos  %f segundos\n",nh,t);
```

- Compila, ejecuta el código y comprueba que ha funcionado correctamente el programa paralelo al usar el comando **cmp** o el comando **diff**

```
jjibanez@upvnet.upv.es@ldsic-vdi08:~/prac_cpa/prac1/s2/codigos$ gcc -fopenmp -o pimagenes-k pimagenes-k.c
jjibanez@upvnet.upv.es@ldsic-vdi08:~/prac_cpa/prac1/s2/codigos$ ./pimagenes-k
Abierta una imagen de n:512, m:512
Tiempo de ejecución paralelización bucle k para 4 hilos  4.500095 segundos
jjibanez@upvnet.upv.es@ldsic-vdi08:~/prac_cpa/prac1/s2/codigos$ cmp peppers-fil.ppm peppers-fil-k.ppm
jjibanez@upvnet.upv.es@ldsic-vdi08:~/prac_cpa/prac1/s2/codigos$
```

Códigos paralelos: paralelización bucle l

```
109  for (p = 0; p < pasos; p++) {
110    for (i = 0; i < n; i++) {
111      for (j = 0; j < m; j++) {
112        resultado.r = 0;
113        resultado.g = 0;
114        resultado.b = 0;
115        tot = 0;
116        for (k = max(0, i - radio); k <= min(n - 1, i + radio); k++) {
117          for (l = max(0, j - radio); l <= min(m - 1, j + radio); l++) {
118            v = ppdBloque[k - i + radio][l - j + radio];
119            resultado.r += ppsImagenOrg[k][l].r * v;
120            resultado.g += ppsImagenOrg[k][l].g * v;
121            resultado.b += ppsImagenOrg[k][l].b * v;
122            tot += v;
123          }
124        }
125        resultado.r /= tot;
126        resultado.g /= tot;
127        resultado.b /= tot;
128
129
130
131
132    }
```

#pragma omp parallel for ....

l=1

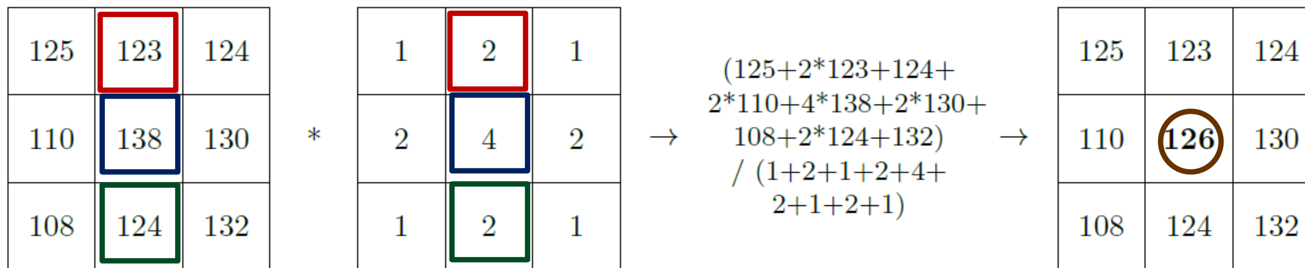


Figura 3: Modelo de aplicación de una media ponderada en el filtrado de imágenes.

Para cada trío de valores (i, j,k) se crea una región paralela en donde se reparten las iteraciones de l entre los hilos. Total ≈ m*n*radio regiones paralelas. En este ejemplo, en la iteración l=1 cada hilo solo calcularía un producto H0(123*2), H1(138*4), H2(138*2). En general, en la iteración l cada hilo realizaría las sumas y productos que le tocan de la columna l

Códigos paralelos: paralelización bucle l

- Para facilitar la implementación de esta paralelización puedes copiar el fichero pimagenes-k.c, llamándolo ahora pimagenes-l.c.
- Elimina o comenta la línea **#pragma omp parallel for** que hayas colocado antes.
- Añade la línea **#pragma omp parallel for** antes del bucle for con variable iteradora **l**; añadiendo las claúsulas necesarias para su correcto funcionamiento
- Modifica las líneas en las que se muestra el número de hilos para saber que se trata de la implementación paralela del bucle l
- Compila, ejecuta el código. Comprobarás que tarda mucho tiempo. Aborta la ejecución y más adelante cuando lances la ejecución en el sistema de colas de Kahn, comprueba que funciona bien

```
jjibanez@upvnet.upv.es@ldsic-vdi08:~/prac_cpa/prac1/s2/codigos$ gcc -fopenmp -o pimagenes-l pimagenes-l.c
jjibanez@upvnet.upv.es@ldsic-vdi08:~/prac_cpa/prac1/s2/codigos$ ./pimagenes-l
Abierta una imagen de n:512, m:512
^C
jjibanez@upvnet.upv.es@ldsic-vdi08:~/prac_cpa/prac1/s2/codigos$
```

Trabajo en el cluster Kahan

- Modifica todos los códigos fuente de manera que el fichero de entrada sea peppers-1k.ppm

```
12    #define IMAGEN_ENTRADA "peppers-1k.ppm"
13    #define IMAGEN_SALIDA "peppers-fil.ppm"
```

- Conéctate desde Polilabs o desde el OL al **front-end** de kahan:

  ssh -l login@alumno.upv.es kahan.dsic.upv.es

- Crea en el front-end el directorio **prac1** y copia el fichero **peppers-1k.ppm** :

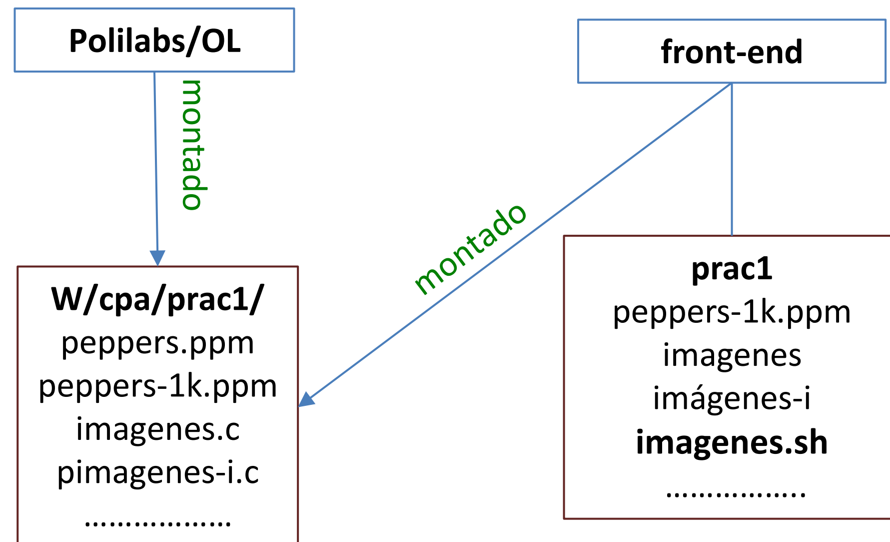
  cp ~/W/cpa/prac1/peppers-1k.ppm ~/prac1/

- Sitúate en el anterior directorio y compila los ficheros

  gcc -Wall -fopenmp -o imagenes ~/W/cpa/prac1/imagenes.c

  gcc -Wall -fopenmp -o pimagenes-i ~/W/cpa/prac1/pimagenes-i.c

- Crea el script imagenes.sh, el cual permitirá lanzar una tarea con una o varias ejecuciones de los códigos generados y cópialo en la carpeta prac1 del front-end (puedes editarlo en W y después copiarlo en el directorio del **front-end**)



Pruebas a realizar sobre el cluster

- Para lanzar los trabajos, podrías crear el siguiente script:

```
#!/bin/bash
#SBATCH --output=res_s2.txt
#SBATCH --nodes=1
#SBATCH --time=5:00
#SBATCH --partition=cpa
./imagenes
./pimagenes-i
./pimagenes-j
./pimagenes-k
./pimagenes-l
```

- Lanza el trabajo escribiendo en la línea de comandos:

```
sbatch imagen.sh
```

- ¡Recuerda no dejar espacios en blanco innecesarios!
- ¿Cuál de las implementaciones paralelas funciona mejor? Justifica la respuesta

Resultados comparativa

```
Abierta una imagen de n:1024, m:1024
Tiempo de ejecucion secuencial 12.714654 segundos
Abierta una imagen de n:1024, m:1024
Tiempo de ejecucion paralelizacion bucle i para 64 hilos
0.491871 segundos
Abierta una imagen de n:1024, m:1024
Tiempo de ejecucion paralelizacion bucle j para 64 hilos
0.651218 segundos
Abierta una imagen de n:1024, m:1024
Tiempo de ejecucion paralelizacion bucle k para 64 hilos
417.466378 segundos
slurmstepd-kahan01: error: *** JOB 1366 ON kahan01
CANCELLED AT 2023-09-29T18:15:01 DUE TO TIME LIMIT ***
```

Pruebas a realizar sobre el cluster

- Modifica el script para obtener los tiempos considerando 2, 8, 32 y 64 hilos (basta hacerlo para la implementación en la que se paraleliza el bucle de variable iteradora i)

Muchas gracias

y

¡ a programar !