PRÀCTICA DE PP: *Kmeans* amb OpenMP

Temps del codi paral·lelitzat amb 2 Threads						
Versió	**K=2**	**K=4**	**K=8**	**K=16**	**K=32**	**K=64**
openMP-Ofast	0.71	2.39	9.72	17.62	35.49	267.15
Iteracions	9	22	51	47	49	196
Checksum	6405336	12689895	22329779	42559141	73490886	124269810
Temps del codi paral·lelitzat amb 6 Threads						
Versió	**K=2**	**K=4**	**K=8**	**K=16**	**K=32**	**K=64**
openMP-Ofast	0.33	0.91	3.43	6.15	12.02	89.85
Temps del codi paral·lelitzat amb 12 Threads						
Versió	**K=2**	**K=4**	**K=8**	**K=16**	**K=32**	**K=64**
openMP-Ofast	0.29	0.53	1.81	3.28	6.25	45.49

Ha estat executat amb WILMA.

Les iteracions i el Checksum coincideix amb diferents threads.

Compilació i execució

El nostre codi es compila fàcilment amb el `job.sub` que ja era proporcionat pel professorat. Conté la comanda `gcc`, la declaració de threads i la comanda `perf stat` per diferents versions. Considerem que aquesta és la manera més fàcil de compilar-lo.

Una vegada s'ha compilat, obtindrem un arxiu *.out* que haurem d'executar amb el `cat`.

Estratègia de paral·lelització

L'estratègia de OpenMP implementada distribueix la carrega de treball dels **bucles** entre diferents threads. Hem centrat la paral·lelització en el bucle principal de la *funció main kmeans* (el tercer pas), ja que és on es realitzen la majoria de operacions.

Els bucles que poden ser paral·lelitzats són tots els bucles que formen part del bucle principal, ja que cada iteració de cadascun dels bucles és independent.

El coll d'ampolla de la versió seqüencial era la crida al mètode `find_closest_centroid()` que era el mètode que consumia més temps, degut a que es cridava en cada iteració i es feia per cada píxel de la imatge.

Canvis realitzats per obtenir la solució

Un requisit previ abans de modificar res del codi, és importar la llibreria `#include <omp.h>` per poder utilitzar OpenMP.

Hem realitzat canvis importants en el procés de trobar el clúster més proper a cada píxel. Per aconseguir-ho, hem creat unes **variables auxiliars en forma de vector** de 32 bits, que ens permeten fer la reducció de les sumes parcials dels valors dels píxels (assignats a cada clúster). En comptes de fer les sumes directament sobre els centroides, les fem sobre aquests vectors. Aquesta estratègia fa possible realitzar la reducció, on cada thread farà les seves sumes i després combinaran els resultats. A més, aquesta tècnica ens permet utilitzar menys memòria en comparació amb l'estructura separada que teníem per cada centroide.

Les línies a continuació mostren les modificacions resultants:

```
#pragma omp parallel for reduction(+: red[:k], green[:k], blue[:k], points[:k])
for(j = 0; j < num_pixels; j++)
{
        closest = find_closest_centroid(&pixels[j], centroides, k);
        red[closest] += pixels[j].r;
```

```
        green[closest] += pixels[j].g;
        blue[closest] += pixels[j].b;
        points[closest] += 1;
}
```

Ara ens centrem en el coll d'ampolla que s'ha mencionat anteriorment i que per tant hem d'optimitzar, en comptes d'utilitzar un vector `dis[num_clusters]` que emmagatzemava les distàncies entre el punt i cada centroide, s'ha utilitzat una variable auxiliar. El fet de tenir un vector feia que s'utilitzés molta memòria innecessària, ja que només volíem saber el centroide més proper (el que té distància mínima). Al principi pensàvem que es podria paral·lelitzar aquest bucle, però quan ho intentàvem, el temps augmentava, això es degut a que hi ha una forta dependència de dades entre iteracions (cada iteració del bucle depèn de l'anterior). El codi resultant ha sigut:

```
uint8_t find_closest_centroid(rgb* p, cluster* centroids, uint8_t num_clusters){
        uint32_t min = UINT32_MAX;
        uint8_t closest = 0, j;
        int16_t diffR, diffG, diffB;
        uint32_t aux; // !!!!!!!!!!!!!!!!!

        for(j = 0; j < num_clusters; j++)
        {
                diffR = centroids[j].r - p->r;
                diffG = centroids[j].g - p->g;
                diffB = centroids[j].b - p->b;
                // No sqrt required.
                aux = diffR*diffR + diffG*diffG + diffB*diffB;

                if(aux<min)
                {
                        min = aux;
                        closest = j;
                }
        }
        return closest;
```

En el bucle que actualitza els centroides també s'han realitzat diverses modificacions per millorar-ne l'eficiència, especialment en relació amb la manipulació dels vectors de dades. Amb l'ús d'aquests vectors, el càlcul de la mitjana de cada component de colors de píxels es pot fer de manera més òptima, ja que podrem accedir més ràpidament a caché i millorar el rendiment.

A més hem aplicat la directiva `#pragma omp parallel for reduction(||: condition)` que permet fer una operació de reducció sobre la condició OR de la variable `condition`, al final de la regió paral·lela totes còpies de cada thread es combinaran i per obtenir el resultat final de la variable.

El fet d'afegir aquest pragma ha disminuït considerablement el temps d'execució. Les línies inferiors mostren el codi actualitzat de la regió on s'actualitzen els valors dels centroides:

```
condition = 0;
#pragma omp parallel for reduction(||: condition)
for(j = 0; j < k; j++)
{
        if(points[j]> 0)
        {
        centroides[j].media_r = red[j] / points[j];
        centroides[j].media_g = green[j] / points[j];
        centroides[j].media_b = blue[j] / points[j];
        changed = centroides[j].media_r != centroides[j].r || centroides[j].media_g !=
                  centroides[j].g || centroides[j].media_b != centroides[j].b;
        condition = condition || changed;
        centroides[j].r = centroides[j].media_r;
        centroides[j].g = centroides[j].media_g;
```

```
        centroides[j].b = centroides[j].media_b;
        }
}

i++;
```

A continuació es mostra el primer bucle que realitza el reset dels centroides. Hem hagut d'inicialitzar a més a més dels valors de `media` i `num_puntos` de cada centroide, els valors dels vectors `red, green, blue,` utilitzats per calcular la suma acumulada de cada component de color, i `points`. La directiva `#pragma omp parallel for` indica a OpenMP que s'ha d'executar en paral·lel i dividirà el treball entre els threads disponibles, així permetem que els múltiples fils inicialitzen els centroides de forma simultània. Afegir aquest pragma ha fet que el temps disminueixi però tampoc gaire, tenint en compte les altres.

Tal com hem mencionat anteriorment, cada iteració del bucle és independent de la resta (cada thread treballa en un índex de centroide diferent) per tant es perfectament paral·lelitzable, ja que no hi ha dependències de dades entre les iteracions.

```
#pragma omp parallel for
for(j = 0; j < k; j++)
{
        centroides[j].media_r = 0;
        centroides[j].media_g = 0;
        centroides[j].media_b = 0;
        centroides[j].num_puntos = 0;

        red[j] = 0;
        green[j] = 0;
        blue[j] = 0;
        points[j] = 0;
}
```

Mètriques de rendiment

Recordem que, el codi seqüencial trigava:

Versió	**K=2**	**K=4**	**K=8**	**K=16**	**K=32**	**K=64**
seq+Ofast	1.30	4.38	19.61	34.71	70.56	531.86

Hem decidit observar com el temps canvia al repartir la càrrega de treball entre 2, 6 i 12 threads. Les taules inferiors mostren el Speedup i eficiència obtinguts:

Temps del codi paral·lelitzat amb 2 Threads						
Versió	**K=2**	**K=4**	**K=8**	**K=16**	**K=32**	**K=64**
openMP-Ofast	0.71	2.39	9.72	17.62	35.49	267.15
Speedup	1.83x	1.83x	2.02x	1.97x	1.99x	1.99x
Eficiència	0.91	0.91	1.01	0.98	0.99	0.99
Temps del codi paral·lelitzat amb 6 Threads						
Versió	**K=2**	**K=4**	**K=8**	**K=16**	**K=32**	**K=64**
openMP-Ofast	0.33	0.91	3.43	6.15	12.02	89.85
Speedup	3.93x	4.81x	5.72x	5.64x	5.87x	5.91x
Eficiència	0.65	0.8	0.95	0.94	0.97	0.98
Temps del codi paral·lelitzat amb 12 Threads						
Versió	**K=2**	**K=4**	**K=8**	**K=16**	**K=32**	**K=64**
openMP-Ofast	0.29	0.53	1.81	3.28	6.25	45.49
Speedup	4.48x	8.26x	10.83x	10.58x	11.29x	11.69x
Eficiència	0.37	0.68	0.9	0.88	0.94	0.97

Anàlisi de rendiment

Una observació clau per analitzar el rendiment del nostre programa paral·lelitzat és l'speedup, és a dir, la relació entre el temps de la versió seqüencial i el temps de l'execució de la versió paral·lelitzada. Quant més gran sigui l'speedup millor serà l'eficiència de la paral·lelització, per tant, observem que augmenta amb el numero de threads, això indica que la paral·lelització està sent efectiva.

L'eficiència es altra mesura d'anàlisis que ens indica com de bé s'està utilitzant la potencia de cada fil. Es calcula com l'speedup dividit entre el numero de threads. Una eficiència propera a 1 significa que estem optimitzant els recursos, és òptim. En aquest cas, l'eficiència disminueix una mica a mesura que augmenten el numero de threads: amb k=2 paral·lelitzat amb 2 threads=0.91 i 6 threads=0.65. Tot i això, es manté bastant alta molt a prop de 1.

En general, observem que a mesura que augmenta el numero de clústers (k), l'speedup i l'eficiència milloren. Això passa perquè hi ha més recursos de processament disponibles per dividir la carrega de treball.

En resum, podem afirmar que la paral·lelització de l'algorisme k-means utilitzant OpenMP ha estat efectiva en la millora del rendiment del programa.

Problemes trobats

Al principi de tot ens vam trobar amb un codi bastant extens que no sabíem ben bé per on començar, després de llegir-ho tot i analitzar-ho detingudament vam aplicar diversos pragmas i modificacions. Però sí que és veritat, que en comptes de disminuir el temps, l'augmentaven (obtenint 1.40 – 1.60s com a temps d'execució), altres vegades disminuíem el temps però no obteníem el Checksum correcte degut a que les operacions no es feien de la manera corresponent i d'altres només aconseguíem fer una única iteració.

GitHub

https://github.com/annagarciav/kmeans.git

ANNEX:

```
[pp-32@clus-login OpenMPLAB]$ sbatch job.sub
Submitted batch job 224006
[pp-32@clus-login OpenMPLAB]$ cat slurm-224006.out
TEST = 1

WIDTH : 3840
HEIGHT: 2160
LENGHT: 8294400

STEP 1: K = 2
STEP 2: Init centroids
STEP 3: Updating centroids

Number of K-Means iterations: 9

Centroide 0 : R[68]     G[56]   B[45]
Centroide 1 : R[196]    G[186]  B[171]

Time to Kmeans is 0 seconds and 567522 microseconds
Checksum value = 6405336

 Performance counter stats for './kmeans_OpenMP.exe test 2 imagen.bmp':

         1.258,38 msec task-clock:u                     #    1,770 CPUs utilized
                0      context-switches:u               #    0,000 /sec
                0      cpu-migrations:u                 #    0,000 /sec
              525      page-faults:u                    #  417,204 /sec
    3.198.198.084      cycles:u                         #    2,542 GHz                         (66,55%)
       18.763.336      stalled-cycles-frontend:u        #    0,59% frontend cycles idle        (66,56%)
      761.529.456      stalled-cycles-backend:u         #   23,81% backend cycles idle         (66,69%)
    6.010.715.218      instructions:u                   #    1,88  insn per cycle
                                                  #    0,13  stalled cycles per insn     (66,80%)
      611.753.187      branches:u                       #  486,145 M/sec                       (66,76%)
        2.244.193      branch-misses:u                  #    0,37% of all branches             (66,64%)

       0,711142902 seconds time elapsed

       1,232836000 seconds user
       0,015810000 seconds sys
```

Output a l'executar la versió paral·lela amb K=2 i 2 threads.

```
[pp-32@clus-login OpenMPLAB]$ cat slurm-224008.out
TEST = 1

WIDTH : 3840
HEIGHT: 2160
LENGHT: 8294400

STEP 1: K = 4
STEP 2: Init centroids
STEP 3: Updating centroids

Number of K-Means iterations: 22

Centroide 0 : R[37]     G[30]   B[25]
Centroide 1 : R[160]    G[144]  B[123]
Centroide 2 : R[218]    G[213]  B[201]
Centroide 3 : R[97]     G[81]   B[63]

Time to Kmeans is 2 seconds and 247334 microseconds
Checksum value = 12689895

 Performance counter stats for './kmeans_OpenMP.exe test 4 imagen.bmp':

         4.618,94 msec task-clock:u                     #    1,930 CPUs utilized
                0      context-switches:u               #    0,000 /sec
                0      cpu-migrations:u                 #    0,000 /sec
            2.022      page-faults:u                    #  437,763 /sec
   11.857.297.737      cycles:u                         #    2,567 GHz                         (66,67%)
      157.999.063      stalled-cycles-frontend:u        #    1,33% frontend cycles idle        (66,72%)
    2.606.568.151      stalled-cycles-backend:u         #   21,98% backend cycles idle         (66,68%)
   21.437.627.553      instructions:u                   #    1,81  insn per cycle
                                                  #    0,12  stalled cycles per insn     (66,68%)
    2.002.852.072      branches:u                       #  433,618 M/sec                       (66,65%)
       20.826.956      branch-misses:u                  #    1,04% of all branches             (66,61%)

       2,393372594 seconds time elapsed

       4,565272000 seconds user
       0,018809000 seconds sys
```

Output a l'executar la versió paral·lela amb K=4 i 2 threads.

```
[pp-32@clus-login OpenMPLAB]$ cat slurm-224009.out
TEST = 1

WIDTH : 3840
HEIGHT: 2160
LENGHT: 8294400

STEP 1: K = 8
STEP 2: Init centroids
STEP 3: Updating centroids

Number of K-Means iterations: 51

Centroide 0 : R[49]     G[46]   B[41]
Centroide 1 : R[90]     G[66]   B[48]
Centroide 2 : R[194]    G[186]  B[169]
Centroide 3 : R[98]     G[102]  B[92]
Centroide 4 : R[155]    G[105]  B[61]
Centroide 5 : R[26]     G[19]   B[15]
Centroide 6 : R[229]    G[226]  B[216]
Centroide 7 : R[154]    G[142]  B[126]

Time to Kmeans is 9 seconds and 578534 microseconds
Checksum value = 22329779

 Performance counter stats for './kmeans_OpenMP.exe test 8 imagen.bmp':

         19.146,14 msec task-clock:u                     #    1,969 CPUs utilized
                 0      context-switches:u               #    0,000 /sec
                 0      cpu-migrations:u                 #    0,000 /sec
             2.848      page-faults:u                    #  148,751 /sec
    49.309.522.253      cycles:u                         #    2,575 GHz                         (66,67%)
     1.528.743.106      stalled-cycles-frontend:u        #    3,10% frontend cycles idle        (66,66%)
    10.178.620.318      stalled-cycles-backend:u         #   20,64% backend cycles idle         (66,66%)
    83.432.172.428      instructions:u                   #    1,69  insn per cycle
                                                  #    0,12  stalled cycles per insn     (66,67%)
     7.801.397.605      branches:u                       #  407,466 M/sec                       (66,68%)
       207.520.245      branch-misses:u                  #    2,66% of all branches             (66,67%)

       9,722283806 seconds time elapsed

      18,975628000 seconds user
       0,026748000 seconds sys
```

Output a l'executar la versió paral·lela amb K=8 i 2 threads.

```
STEP 1: K = 16
STEP 2: Init centroids
STEP 3: Updating centroids

Number of K-Means iterations: 47

Centroide 0 : R[34]     G[28]   B[23]
Centroide 1 : R[82]     G[47]   B[29]
Centroide 2 : R[171]    G[171]  B[165]
Centroide 3 : R[13]     G[77]   B[99]
Centroide 4 : R[116]    G[77]   B[48]
Centroide 5 : R[20]     G[13]   B[10]
Centroide 6 : R[208]    G[207]  B[198]
Centroide 7 : R[140]    G[135]  B[125]
Centroide 8 : R[147]    G[111]  B[74]
Centroide 9 : R[193]    G[107]  B[41]
Centroide 10 : R[225]   G[194]  B[151]
Centroide 11 : R[104]   G[102]  B[92]
Centroide 12 : R[50]    G[46]   B[39]
Centroide 13 : R[240]   G[237]  B[228]
Centroide 14 : R[73]    G[71]   B[63]
Centroide 15 : R[193]   G[151]  B[109]

Time to Kmeans is 17 seconds and 484525 microseconds
Checksum value = 42559141

 Performance counter stats for './kmeans_OpenMP.exe test 16 imagen.bmp':

         34.935,19 msec task-clock:u                     #    1,982 CPUs utilized
                 0      context-switches:u               #    0,000 /sec
                 0      cpu-migrations:u                 #    0,000 /sec
             3.310      page-faults:u                    #   94,747 /sec
    90.019.472.085      cycles:u                         #    2,577 GHz                         (66,66%)
     4.533.409.083      stalled-cycles-frontend:u        #    5,04% frontend cycles idle        (66,67%)
    18.364.123.470      stalled-cycles-backend:u         #   20,40% backend cycles idle         (66,67%)
   139.742.650.547      instructions:u                   #    1,55  insn per cycle
                                                  #    0,13  stalled cycles per insn     (66,67%)
    13.444.882.450      branches:u                       #  384,852 M/sec                       (66,67%)
       626.758.799      branch-misses:u                  #    4,66% of all branches             (66,67%)

      17,628308070 seconds time elapsed

      34,631102000 seconds user
       0,025764000 seconds sys
```

Output a l'executar la versió paral·lela amb K=16 i 2 threads.

```
STEP 1: K = 32
STEP 2: Init centroids
STEP 3: Updating centroids

Number of K-Means iterations: 49

Centroide 0 : R[23]     G[20]   B[17]
Centroide 1 : R[77]     G[30]   B[14]
Centroide 2 : R[196]    G[194]  B[185]
Centroide 3 : R[29]     G[37]   B[36]
Centroide 4 : R[130]    G[54]   B[23]
Centroide 5 : R[13]     G[10]   B[8]
Centroide 6 : R[156]    G[199]  B[210]
Centroide 7 : R[156]    G[146]  B[134]
Centroide 8 : R[133]    G[126]  B[114]
Centroide 9 : R[194]    G[101]  B[34]
Centroide 10 : R[174]   G[169]  B[161]
Centroide 11 : R[102]   G[107]  B[102]
Centroide 12 : R[39]    G[28]   B[19]
Centroide 13 : R[220]   G[216]  B[206]
Centroide 14 : R[68]    G[69]   B[63]
Centroide 15 : R[224]   G[175]  B[74]
Centroide 16 : R[125]   G[86]   B[54]
Centroide 17 : R[205]   G[164]  B[130]
Centroide 18 : R[120]   G[155]  B[161]
Centroide 19 : R[152]   G[109]  B[67]
Centroide 20 : R[8]     G[52]   B[85]
Centroide 21 : R[178]   G[132]  B[95]
Centroide 22 : R[98]    G[69]   B[46]
Centroide 23 : R[24]    G[119]  B[121]
Centroide 24 : R[77]    G[53]   B[36]
Centroide 25 : R[244]   G[241]  B[234]
Centroide 26 : R[121]   G[105]  B[84]
Centroide 27 : R[54]    G[40]   B[28]
Centroide 28 : R[89]    G[87]   B[78]
Centroide 29 : R[52]    G[52]   B[47]
Centroide 30 : R[231]   G[202]  B[160]
Centroide 31 : R[35]    G[9]    B[6]

Time to Kmeans is 35 seconds and 352031 microseconds
Checksum value = 73490886

 Performance counter stats for './kmeans_OpenMP.exe test 32 imagen.bmp':

         69.914,88 msec task-clock:u                     #    1,970 CPUs utilized
                 0      context-switches:u               #    0,000 /sec
                 0      cpu-migrations:u                 #    0,000 /sec
             9.815      page-faults:u                    #  140,385 /sec
   180.362.473.871      cycles:u                         #    2,580 GHz                         (66,67%)
     8.234.577.948      stalled-cycles-frontend:u        #    4,57% frontend cycles idle        (66,67%)
    38.285.664.563      stalled-cycles-backend:u         #   21,23% backend cycles idle         (66,66%)
   276.316.351.052      instructions:u                   #    1,53  insn per cycle
                                                  #    0,14  stalled cycles per insn     (66,67%)
    27.015.793.327      branches:u                       #  386,410 M/sec                       (66,67%)
     1.147.936.920      branch-misses:u                  #    4,25% of all branches             (66,66%)

      35,497862208 seconds time elapsed

      69,390007000 seconds user
       0,025784000 seconds sys
```

Output a l'executar la versió paral·lela amb K=32 i 2 threads.

```
Centroide 33 : R[206]   G[163]  B[128]
Centroide 34 : R[24]    G[28]   B[5]
Centroide 35 : R[118]   G[109]  B[97]
Centroide 36 : R[225]   G[179]  B[75]
Centroide 37 : R[195]   G[194]  B[185]
Centroide 38 : R[214]   G[211]  B[203]
Centroide 39 : R[247]   G[246]  B[240]
Centroide 40 : R[45]    G[50]   B[13]
Centroide 41 : R[15]    G[6]    B[4]
Centroide 42 : R[86]    G[58]   B[35]
Centroide 43 : R[232]   G[227]  B[217]
Centroide 44 : R[7]     G[66]   B[125]
Centroide 45 : R[95]    G[111]  B[108]
Centroide 46 : R[10]    G[13]   B[11]
Centroide 47 : R[149]   G[120]  B[94]
Centroide 48 : R[89]    G[39]   B[23]
Centroide 49 : R[179]   G[173]  B[162]
Centroide 50 : R[49]    G[13]   B[7]
Centroide 51 : R[6]     G[38]   B[83]
Centroide 52 : R[147]   G[138]  B[126]
Centroide 53 : R[240]   G[219]  B[181]
Centroide 54 : R[42]    G[28]   B[19]
Centroide 55 : R[126]   G[83]   B[51]
Centroide 56 : R[94]    G[206]  B[197]
Centroide 57 : R[153]   G[101]  B[64]
Centroide 58 : R[142]   G[28]   B[21]
Centroide 59 : R[19]    G[13]   B[10]
Centroide 60 : R[191]   G[120]  B[84]
Centroide 61 : R[80]    G[87]   B[82]
Centroide 62 : R[39]    G[40]   B[39]
Centroide 63 : R[71]    G[25]   B[11]

Time to Kmeans is 267 seconds and 10412 microseconds
Checksum value = 124269810

 Performance counter stats for './kmeans_OpenMP.exe test 64 imagen.bmp':

         528.945,23 msec task-clock:u                     #    1,980 CPUs utilized
                 0      context-switches:u               #    0,000 /sec
                 0      cpu-migrations:u                 #    0,000 /sec
             4.077      page-faults:u                    #    7,708 /sec
 1.365.547.383.815      cycles:u                         #    2,582 GHz                         (66,67%)
    42.005.174.133      stalled-cycles-frontend:u        #    3,08% frontend cycles idle        (66,67%)
   306.226.326.193      stalled-cycles-backend:u         #   22,43% backend cycles idle         (66,67%)
 2.146.755.707.399      instructions:u                   #    1,57  insn per cycle
                                                  #    0,14  stalled cycles per insn     (66,67%)
   211.617.995.331      branches:u                       #  400,075 M/sec                       (66,67%)
     5.883.560.332      branch-misses:u                  #    2,78% of all branches             (66,67%)

      267,154263841 seconds time elapsed

      525,337579000 seconds user
        0,033715000 seconds sys
```

Output a l'executar la versió paral·lela amb K=64 i 2 threads.

```
STEP 1: K = 16
STEP 2: Init centroids
STEP 3: Updating centroids

Number of K-Means iterations: 47

Centroide 0 : R[34]     G[28]   B[23]
Centroide 1 : R[82]     G[47]   B[29]
Centroide 2 : R[171]    G[171]  B[165]
Centroide 3 : R[13]     G[77]   B[99]
Centroide 4 : R[116]    G[77]   B[48]
Centroide 5 : R[20]     G[13]   B[10]
Centroide 6 : R[208]    G[207]  B[198]
Centroide 7 : R[140]    G[135]  B[125]
Centroide 8 : R[147]    G[111]  B[74]
Centroide 9 : R[193]    G[107]  B[41]
Centroide 10 : R[225]   G[194]  B[151]
Centroide 11 : R[104]   G[102]  B[92]
Centroide 12 : R[50]    G[46]   B[39]
Centroide 13 : R[240]   G[237]  B[228]
Centroide 14 : R[73]    G[71]   B[63]
Centroide 15 : R[193]   G[151]  B[109]

Time to Kmeans is 6 seconds and 8256 microseconds
Checksum value = 42559141

 Performance counter stats for './kmeans_OpenMP.exe test 16 imagen.bmp':

         35.020,77 msec task-clock:u                     #    5,692 CPUs utilized
                 0      context-switches:u               #    0,000 /sec
                 0      cpu-migrations:u                 #    0,000 /sec
             1.501      page-faults:u                    #   42,860 /sec
    90.197.403.241      cycles:u                         #    2,576 GHz                         (66,66%)
     4.487.895.717      stalled-cycles-frontend:u        #    4,98% frontend cycles idle        (66,67%)
    18.417.577.402      stalled-cycles-backend:u         #   20,42% backend cycles idle         (66,67%)
   140.107.607.667      instructions:u                   #    1,55  insn per cycle
                                                  #    0,13  stalled cycles per insn     (66,69%)
    13.560.514.468      branches:u                       #  387,214 M/sec                       (66,68%)
       613.207.608      branch-misses:u                  #    4,52% of all branches             (66,65%)

       6,152916539 seconds time elapsed

      34,709139000 seconds user
       0,029703000 seconds sys
```

Output a l'executar la versió paral·lela amb K=16 i 6 threads.