Facultad de Informática
UPV/EHU

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea

Grado en Ingeniería Informática

Arquitectura de Computadores

GII – 2º curso

Proyecto: sistemas multiprocesador

Paralelización de una aplicación utilizando **OpenMP**

En el tercer tema de la asignatura estamos analizando las características principales de los sistemas multiprocesador y, para programar aplicaciones paralelas en estos sistemas, estamos utilizando OpenMP. Tras los ejercicios iniciales realizados en el laboratorio, tienes que completar y paralelizar una aplicación.

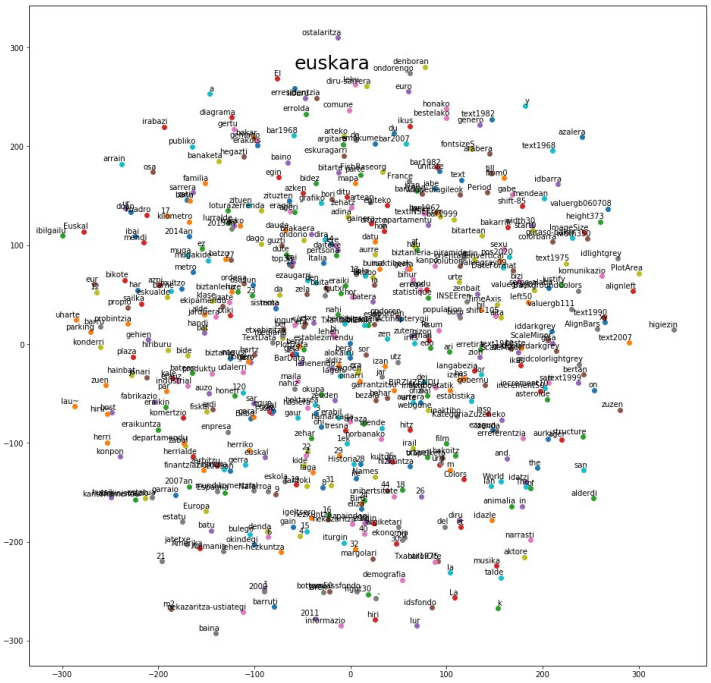
Se trata de una simplificación de una aplicación real, del ámbito del **procesamiento del lenguaje natural (NLP: Natural Language Processing)** y **aprendizaje automático (ML: Machine Learning)**. El objetivo es aplicar todos los conocimientos vistos en clase y trabajados en los laboratorios para desarrollar una aplicación paralela correcta y lo más eficiente posible. Utiliza todas las técnicas aprendidas para paralelizar la aplicación, a pesar de que, en algunas partes del código, los tiempos de ejecución son muy pequeños y su paralelización quizá no sea muy eficiente. Muestra y explica luego, en el informe final, todas las parte paralelizadas.

El trabajo está pensado para que se realice en grupos de tres personas, de forma colaborativa (no cada uno una parte). Antes de empezar a programar, es importante leer con atención esta documentación y analizar el código de la aplicación: estructuras de datos, programa principal, objetivo de las funciones que se deben programar….

Cercanía de conceptos

El NLP es un campo de conocimiento que está teniendo muchos avances en los últimos años. Estos avances han posibilitado crear, por ejemplo, aplicaciones de traducción automática o los modelos generativos de lenguaje. Tal ha sido su impacto que han influido en nuestro modo de trabajar. A lo largo de los años, las técnicas de NLP han ido cambiando y perfeccionando. Sin embargo, en los últimos años, uno de los factores que ha posibilitado esta mejora tan significativa ha sido el uso de redes neuronales profundas para mapear las palabras y frases a un espacio vectorial (nos centraremos en palabras).

La incrustación de palabra (*word embedding*), los vectores de distribución para representar palabras, son vectores numéricos (suelen ser de 300 dimensiones) que se utilizan en el campo de NLP. Estos vectores se inducen a partir de un gran *corpus* de texto y cada palabra del *corpus* se codifica en un espacio vectorial, asignándole así, una representación abstracta de su significado. Esta representación se basan en la idea de "El significado de una palabra lo determinan sus palabras vecinas!". Esto posibilita aplicar operaciones matemáticas sobre estos vectores, lo cual facilita medir cuantitativamente las similitudes y relaciones entre las palabras.



Representación en dos dimensiones de 500 palabras de euskera.

Han digitalizado todos los libros encontrados en la biblioteca Imperial de Constantinopla (ver URL1). Ahora, se quiere realizar un análisis automático de los temas científicos y tecnológicos que se tratan en estos libros. Para ello, en una primera fase, se han generado las *word embeddings* para muchas de las palabras recolectadas, y para la segunda fase, nos han pasado dos ficheros grandes. En un fichero se encuentran las representaciones vectoriales de dichas palabras, y en la otra, la distancia que tiene cada una de esta palabra a cada campo de investigación definido por la UNESCO (ver URL2). La tarea que nos han encomendado dentro de este proyecto es procesar dichos ficheros de datos, agrupar palabras y analizar cómo se relaciona cada agrupamiento con los campos UNESCO. Todo esto, lo más rápido posible.

[URL1] https://es.wikipedia.org/wiki/Biblioteca_Imperial_de_Constantinopla

[URL2] https://es.wikipedia.org/wiki/Clasificaci%C3%B3n_Unesco

En la primera fase, hay que procesar el fichero **`vecpal.dat`** que contiene 211.640 representaciones vectoriales de palabras --40 dimensiones por palabra, normalizados entre -1 y 1--, con el fin de agruparlos en diferentes grupos conceptuales (clústeres), para ello se utilizará la cercanía entre los significados de las palabras.

**** K-means ****

Para clasificar los vectores (palabras) en su grupo conceptual se utilizará el algoritmo de *clustering* **K-means** (Lloyd 1982). La técnica de clustering o agrupamiento es un método no-supervisado de clasificación de patrones, el cual particiona el espacio de entrada en diferentes clústeres. El objetivo de los algoritmos de clustering es crear particiones de clústeres de tal manera que los ejemplos que pertenecen al mismo clúster sean similares y los ejemplos que pertenecen a clústeres distintos, diferentes. K-means es uno de los algoritmos más utilizados en la literatura científica y su principio principal es minimizar la suma de errores al cuadrado (ver algoritmo) entre los ejemplos del mismo grupo. El algoritmo es el siguiente:

Begin
  Training data: $X = \{X_i,\ 1 \le i \le N\}$
  Select the number of clusters: $K \le N$
  Randomly select K centroids: $C = \{C_j,\ 1 \le j \le K\}$;
  Repeat
    Assign the instances to the closest cluster centroids:
      for i in 1 to N
        closest $C(X_i) = C_i \mid \min_{1 \le j \le K} d(X_i, C_j)$
      end for;
    Update the K cluster centroids C:
      for j in 1 to K
        C_j = mean ($X_i \mid$ closest $C(X_i) = j$);
      end for;
  Until cluster centroids stop changing or maximum number of iterations
End

Cada agrupación o clúster se representa por el punto céntrico que se calcula haciendo la media de todos los elementos del clúster. Este punto céntrico es conocido como “centroide”.

El algoritmo K-means genera particiones de, como mucho, K clústeres/ agrupaciones, pero K es un hiper-parámetro que debe de ser dado al algoritmo, pero el usuario no sabe cual es el número de clústeres idóneo. Por tanto, una vez que el algoritmo de clustering haya procesado la base de datos y obtenido una partición de clústeres que particione los datos de entrada en K grupos, emerge una cuestión relevante: ¿Cómo de bien se ajusta la partición a los datos de entrada? ¿Ha sido el K seleccionado una buena elección?

Para ello, la metodología que se sigue en un proceso de clustering es el siguiente: se computan diferentes particiones de clústeres y se selecciona la partición que mejor se ajusta a los datos de entrada. Como esta información (K) no es previamente conocida, tendremos que ejecutar el algoritmo varias veces con diferentes valores de K. En nuestro caso, el K inicial lo fijaremos a 35 y lo incrementaremos por 10 en cada iteración, obteniendo particiones de 35, 45, 55... clústeres. En este proceso evaluaremos todas las particiones y seleccionaremos una de las mejores en ajustarse a los datos de entrada.

El proceso de estimar cómo de bien se ajusta una partición de datos a la estructura que subyace en los datos se conoce como la validación de clústeres (Cluster Validation). Las técnicas más comunes validan una partición utilizando los mismos datos particionados. Para ello miden o aproximan la compactitud intra-clúster y la separación inter-clúster. Estos indicadores se llaman Índices de Validación de Clústeres (CVI: Cluster Validity Indexes). En este proyecto vamos a utilizar uno de estos índices para validar y seleccionar las particiones que generemos.

**** Índices de Validación de Clústeres (Cluster Validity Indexes / CVI) ****

Nuestro CVI, por un lado, calculará la compactitud de cada clúster o grupo (cuan compactos están los elementos del clúster) como el promedio de las distancias entre pares de los elementos del grupo (término a). Por otro lado, la separación de un clúster se calculará como el promedio de las distancias entre el centroide del clúster en cuestión y los otros centroides (término b). A continuación se describirá el índice que se va ha utilizar:

- **El término a(k) aproxima la distancia intra-clúster del clúster k**, y para ello calcula la media de todas las distancias entre elementos que pertenecen al clúster k. Nos da la idea de cuan cohesionados/ compactos están los elementos dentro del clúster k o qué densidad tiene el clúster k. Por el bien de la simplicidad y la comprensión esta fórmula está sin optimizar.

$$a(k) = \begin{cases} \frac{1}{|c_k|^2} \sum_{x_i \in c_k} \sum_{x_j \in c_k} d(x_i, x_j) & |c_k| > 1, \\ 0 & |c_k| \leq 1. \end{cases}$$

  donde c_k es el centroide del clúster k, x_i y x_j son los elementos i y j que están asociados al centroide c_k o al clúster k, d() es la distancia euclidiana entre dos ejemplos y $|c_k|$ es el número de ejemplos que pertenecen al clúster k.

- **El término b(k) aproxima la distancia inter-clúster del clúster k**, y para ello calcula la media de todas las distancias entre el centroide k y los otros centroides. Nos da la idea de cuan separado está el clúster k de otras.

$$b(k) = \frac{1}{|C| - 1} \sum_{c_p \in C} d(c_k, c_p)$$

  donde C es la colección de centroides, |C| el número de centroides o el número de clústeres que tiene la partición de clústeres y c_k y c_p son los centroides k y p.

- **El término s(k) es un ratio entre el termino a(k) y b(k)** que indica la calidad del clúster k. En cuanto a este termino, lo deseable, es que la distancia intra-clúster (término a) sea pequeña (cohesión) y la distancia inter-clúster (término b) grande (separación).

$$s(k) = \frac{b(k) - a(k)}{max\{a(k), b(k)\}}$$

- **S es la media de todos los términos s(k)** e indica la calidad de la partición de clústeres.

$$S = \frac{1}{|C|} \sum_{c_k \in C} s(k)$$

  Definido así, el rango de este índice es [-1, 1], donde -1 indica una mala partición y el 1 una buena partición.

En la segunda fase, después de agrupar las palabras acorde a la cercanía semántica, se analizará la relación que tiene cada campo de ciencia y tecnología definidas por UNESCO con cada uno de los grupos (**`campo.dat`**). Para ello, habrá que calcular para cada campo qué clúster le queda más cerca y cual más lejos, utilizando la mediana de las distancias entre las palabras de un grupo, según su cercanía a los campos UNESCO.

**** Análisis de grupos conceptuales ****

Como se ha comentado, la base de datos contiene más de 211.000 vectores de 40 dimensiones (valores normalizados entre -1 y 1). El objetivo es agrupar todas estos vectores en K grupos conceptuales (*clústeres*) diferentes, en función de la cercanía entre sus 40 dimensiones.

Tras la generación de los K clústeres, se realizará un análisis de la relación con los 23 campos UNESCO (**`campo.dat`**) en los vectores (palabras) encuadradas en cada grupo, obteniendo cuál es el mayor y el menor porcentaje de presencia de cada campo entre todos los grupos y los grupos que tienen esos valores (máximo y mínimo). La relación con un campo UNESCO en un grupo se calculará como la mediana de las probabilidades correspondientes a ese campo en el grupo en cuestión.

Se utilizará una versión simplificada de la mediana: se ordenarán los valores del conjunto de menor a mayor y se devolverá el valor que está en la posición N/2 (división entre enteros) tanto si N es impar como par.

Resumen

La aplicación funciona de esta forma:

1ª fase

1. Al comienzo se leen dos ficheros. Por una parte, el banco de vectores (**`vecpal.dat`**), que se almacena en la matriz **`pal`**. Cada fila de la matriz tiene la representación vectorial de una palabra (40 valores en 40 columnas). Por otra parte, la vinculación de dicho vector con los 23 campos UNESCO (**`campo.dat`**), que se almacena en la matriz **`campo`** (cada fila tiene 23 valores, valores entre 0 y 1 que indican la probabilidad de que la palabra a pertenecer a un campo UNESCO).
2. A continuación empieza un proceso iterativo (desde el paso 3 a 8) en el que se genera la partición y se evalúa.
3. Se genera de forma aleatoria un *centroide* inicial para cada uno de los K grupos (clústeres). En este caso como el K inicial es 35, se generarán 35 centroides en un espacio de 40 dimensiones que corresponde al espacio de características a la cual se han mapeado las palabra. Estos centroides serán los representantes del clúster. Este cálculo se realiza en la función: **`inicializar_centroides (...)`**
4. Partiendo de esa situación, se calcula la distancia entre todas las palabras (representación vectorial) del fichero y los 35 *centroides*, utilizando la función **`distpal()`**. La distancia entre dos palabras (representación vectorial) es simplemente la distancia euclídea entre los 40 valores de sus características.

   $$\text{dis}(p, q) = \text{raiz\_cuadrada}\ [\ (p_1 - q_1)^2 + (p_2 - q_2)^2 + \ldots + (p_n - q_n)^2\ ]$$

   donde p_i y q_i son los elementos i de la representación vectorial de la palabra p y q.

   Las distancias calculadas se utilizan para hacer los grupos: se asigna a cada palabra (representación vectorial) del fichero su grupo conceptual más cercano, el de menor distancia, mediante la función **`grupo_cercano (...)`**
5. Una vez clasificados las palabras (representación vectorial) en K grupos, se calcula un nuevo *centroide* para cada grupo, esto es, se calculan los valores medios para las 40 características de todos los miembros de ese grupo.
6. Se calcula la distancia entre el nuevo *centroide* y el *centroide* anterior.
7. Se repite el proceso hasta que se cumpla una de estas dos condiciones: (a) el movimiento de los valores de los *centroides* es inferior a un determinado umbral (DELTA1), (b) se supera un número máximo de iteraciones (MAXIT).
8. Una vez que la partición está generado será evaluado utilizando el CVI. Para ello, se comparará la bondad de la partición (valor del CVI) actual con la de la anterior y si la bondad actual es mejor el número de clústeres (K) se incrementará en 10 y el proceso se repetirá. Si la calidad o bondad de la partición actual (según el CVI) no mejora

la de la partición anterior, la búsqueda del K se para y se usa esta última partición en la fase siguiente. Esta estrategia de búsqueda es conocida como "best-first".

2ª fase

1. El proceso de clusterización particiona las palabras de la base de datos en diferentes grupos. Las palabras que se encuentren en un mismo grupo estarán semánticamente cerca. Una vez finalizado este proceso se realiza el análisis de cuan cerca están estos grupos con los campos UNESCO de ciencia y tecnología, para ello se hará uso de la función: **`grupo_cercano (...).`** Analizando cada grupo hay que calcular, para campo UNESCO qué grupo está más cerca (mínimo) y qué grupo más lejos (máximo). Para ello se utilizará como la cercanía representativa de un grupo hacia un campo concreto, la mediana de las cercanías que tiene un grupo al campo UNESCO en cuestión. Para ello, se utilizará la función **analisis_campos `(...)`**

   Tened en cuenta que para calcular la mediana hay que ordenar los valores de cercanía a un campo UNESCO del grupo en cuestión y hay que quedarse con el valor medio (N/2 tanto para N impar como par, por ejemplo, si hay 80 o 81 valores ordenados de menor al mayor, se tomará como el valor medio la posición 40). Ordena las los valores de cercanía utilizando un algoritmo simple de ordenar, como el de la burbuja.

Material proporcionado

Tienes todo el material necesario para desarrollar la aplicación en el directorio **ARQ/ppalabras** de tu cuenta de trabajo. Copia el material a un nuevo directorio de trabajo en tu cuenta. **NOTA: no copies los ficheros de entrada (`vecpal.dat` y `campo.dat`); son demasiados grandes, por lo que utilizaremos directamente los que están en el directorio ARQ/pwembed**

- `grupopal_s.c` Programa principal de la aplicación (versión serie).
- `fun_s.c` Contiene las funciones que utiliza el programa principal: **`distpal(), grupo_cercano (), silhouette_simple(), y analisis_campos()`**.
  Tienes que completar esas funciones para conseguir la versión **serie** del programa.
- `defineg.h` Definiciones que se utilizan en los ficheros `grupo_pal_s.c` y `fun_s.c`.
- `fun.h` Cabeceras (declaraciones) de las funciones (`extern`).
- `vecpal.dat` Fichero de entrada que contiene los representaciones vectoriales de palabras. El valor de la primera línea indica el número de vectores y los valores del resto de líneas son vectores con 40 valores, los cuales situan cada palabra en un espacio de características de 40 dimensiones.
- `campos.dat` Fichero de entrada que contiene 23 valores entre 0 y 1 (23 campos) por cada palabra. Estos valores indican la cercanía (distancia) de cada palabra a cada campo UNESCO de ciencia y tecnología.
- `res.out` Fichero que contiene los resultados que debes conseguir: centroides y densidad de los grupos, y análisis de campos. De esta forma, puedes comparar tus resultados (en el fichero `vecpal_s.`out para la versión serie del programa) con los que deberías obtener.

Para compilar y ejecutar el programa

- Para **compilar** todo el programa:

  ```
  gcc -O2 -o grupopal_s grupopal_s.c fun_s.c -lm
  ```

- Para **ejecutar** el programa (suponiendo que tienes el material en un directorio creado desde el directorio raíz de tu cuenta) [puedes crear un comando para lanzar la ejecución del programa]:

  ```
  grupopal_s ../ARQ/pwembed/vecpal.dat ../ARQ/pwembed/campos.dat [ num ]
  ```

  Hay dos opciones para ejecutar el programa. Si se indica el tercer parámetro —`num`—, se procesará sólo el número de vectores indicado. Es la opción que puedes utilizar para realizar pruebas, utilizando un número

pequeño de vectores para que la ejecución sea rápida. Si no se indica el tercer parámetro, se procesa todo el fichero de entrada. Esta opción es la que utilizarás, tras comprobar que el programa funciona correctamente, para obtener los resultados finales.

RECUERDA que trabajaremos en grupo, pero el trabajo de cada grupo es individual.

Para realizar este trabajo práctico sigue los siguientes pasos.

A. Crea la versión serie de la aplicación

A1. Analiza la estructura del programa en serie y completa el código (en el fichero `fun_s.c`).

A2. Crea un comando (*script*) para compilar toda la aplicación. Para ello, edita un fichero de texto con los comandos que quieres ejecutar (por ejemplo, `gcc -O2 -o grupopal_s grupopal_s.c fun_s.c -lm`). Cambia los permisos del fichero para convertirlo en un fichero ejecutable: `chmod 700 nombre_fichero`

De esta forma, será suficiente ejecutar ese comando cada vez que quieras compilar la aplicación, en lugar de tener que escribir cada vez el comando completo (largo) en la línea de comandos de la terminal. No olvides que debes finalizar el comando de compilación con la opción `-lm`, para poder incorporar la librería que incluye las funciones matemáticas que utilizamos.

A3. **Comprueba el correcto funcionamiento de la versión serie del programa**. Compara tus resultados (en el fichero `vecpal_s.out`) con los resultados del fichero `res.out`. Para ello, puedes visualizar algunas líneas de ambos ficheros o comparar ambos ficheros utilizando el siguiente comando Linux:

```
diff res.out dbgen_s.out
```

Para hacer las pruebas iniciales, dispones también de un fichero de resultados reducido, para 1000 vectores (`res1000.out`).

B. Crea la versión paralela de la aplicación

B1. Tras crear la versión serie y comprobar su correcto funcionamiento, crea la versión paralela de la aplicación utilizando OpenMP. Crea una copia de todos los módulos de la versión serie y modifícala para programar la versión paralela (por ejemplo, `grupopal_p.c`, y así para el resto de módulos). De esta forma, siempre tendrás disponible la versión serie completa del programa.

Crea otro script para poder compilar la versión paralela de la aplicación.

Comprueba el correcto funcionamiento de la versión paralela de estas dos formas: (a) en serie y paralelo (por ejemplo, utilizando 3 hilos) comprobando que los resultados son idénticos; y (b) en paralelo con diferentes hilos (por ejemplo, con 2 y 8) para comprobar que los resultados también son idénticos independientemente del número de hilos que se utilicen.

B2. A continuación, **analiza el rendimiento obtenido** en función del número de hilos. Intenta optimizar el código paralelo (estrategias de planificación y funciones de sincronización) para obtener una solución eficiente.

Una vez obtenida la "mejor" versión paralela, realiza la experimentación correspondiente: ejecuta la aplicación para **2, 4, 8, 16, 24, 32, 48** y **64** hilos; calcula los tiempos de ejecución serie y paralelo para ese número de procesos, los factores de aceleración y eficiencias conseguidas.

Obviamente, realiza todas las pruebas y experimentos que te parezcan oportunos.

C. Escribe un informe técnico

Finalmente, tienes que escribir un informe técnico: programas desarrollados, correctamente comentados y explicados, resultados obtenidos (datos y gráficas realizadas), conclusiones, etc. El informe es reflejo de todo el trabajo realizado, tómatelo con tiempo y seriedad. Ten en cuenta las recomendaciones que te hemos indicado en el documento correspondiente (ver documento en eGela).

>> Tiempos de trabajo estimados (grupos de dos personas): 40 horas

- Analizar la aplicación, entenderla y crear la versión serie: 08 - 10 horas
- Crear la versión paralela, comprobaciones, pruebas, resultados: 18 - 22 horas
- Escritura del informe técnico: 08 - 10 horas

>> **Plazo de entrega** (informe y código en eGela): **29 de diciembre**

gengrupos_s.c

```
/*****************************************************************************
    grupopal_s   SERIE

    Entrada: vecpal.dat    fichero con las representaciones vectoriales de palabras
             campos.dat    fichero con la carcania de cada palabra a cada campo UNESCO
    Salida:  vecpal_s.out  centroides, densidad, analisis

    compilar con el modulo fun_s.c y la opcion -lm
*****************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include "defineg.h"
#include "fun.h"

float  mvec[MAXV][NDIM];                // vectores (palabras) a procesar
struct lista_grupos listag[MAX_GRUPOS]; // lista de vectores de los grupos
float  mcam[MAXV][NCAM];                // campos UNESCO asociados a los vectores (palabras)
struct  analisis info_cam[NCAM];        // analisis de los campos UNESCO
int ngrupos = 35;

// programa principal
// ==================
int main (int argc, char *argv[]) {
        float   a[MAX_GRUPOS]; // densidad de cada cluster
        int     popul[MAXV]; // grupo de cada vector
        float   cent[MAX_GRUPOS][NDIM]; // centroides
        int     i, j, nvec, grupo, num, ind;
        int     fin = 0, num_ite = 0;
        int     convergencia_cont;
        double  sil, sil_old, diff;

        FILE   *fd;
        struct timespec t1, t2, t11, t12, t17, t20, t21;
        double texe, t_lec, t_clust, t_camp, t_escri;

        if ((argc < 3)  || (argc > 4)) {
          printf ("[!] ERROR: %s bd_vectores bd_campos [num_vec]\n", argv[0]);
                exit (-1);
        }

        setbuf(stdout, NULL);
        printf ("\n*** Ejecucion serie ***\n\n");
        clock_gettime (CLOCK_REALTIME, &t1);

        // lectura de datos (palabras): mvec[i][j]
        // =======================================
        clock_gettime (CLOCK_REALTIME, &t11);
        fd = fopen (argv[1], "r");
        if (fd == NULL) {
                printf ("[!] Error al abrir el fichero %s\n", argv[1]);
                exit (-1);
        }

        fscanf (fd, "%d", &nvec);
        if (argc == 4) nvec = atoi(argv[3]); // 4. parametro: numero de vectores
```

```
for (i=0; i<nvec; i++)
        for (j=0; j<NDIM; j++)
                fscanf (fd, "%f", &(mvec[i][j]));

fclose (fd);


// lectura de datos (campos): mcam[i][j]
// =====================================
fd = fopen (argv[2], "r");
if (fd == NULL) {
        printf ("[!] Error al abrir el fichero %s\n", argv[2]);
        exit (-1);
}

for (i=0; i<nvec; i++) {
        for (j=0; j<NCAM; j++)
                fscanf (fd, "%f", &(mcam[i][j]));
}
fclose (fd);
clock_gettime (CLOCK_REALTIME, &t12);
t_lec = (t12.tv_sec-t11.tv_sec) + (t12.tv_nsec-t11.tv_nsec)/(double)1e9;

convergencia_cont = 0;
sil_old = -1;
while(ngrupos<MAX_GRUPOS && convergencia_cont<1){
        // generacion de los primeros centroides de forma aleatoria
        // ========================================================
        inicializar_centroides(cent);

        // A: agrupar los vectores y calcular los nuevos centroides
        // ========================================================
        num_ite = 0;
        fin = 0;
        while ((fin == 0) && (num_ite < MAXIT)) {
                // calcular el grupo mas cercano
                grupo_cercano(nvec, mvec, cent, popul);
                // calcular los nuevos centroides de los grupos
                fin = nuevos_centroides(mvec, cent, popul, nvec);
                num_ite++;
        }

        // B: Calcular la "calidad" del agrupamiento
        // =========================================
        // lista de clusters: numero de elementos y su clasificacion
        for (i=0; i<ngrupos; i++) listag[i].nvecg = 0;
        for (i=0; i<nvec; i++){
                grupo = popul[i];
                num=listag[grupo].nvecg;
                listag[grupo].vecg[num] = i; // vectores de cada grupo (cluster)
                listag[grupo].nvecg++;
        }

        // silhouette: calidad de la particion de clusters
        sil = silhouette_simple(mvec, listag, cent, a);
```

```
                // calcular la diferencia: medida de la estabilidad
                diff = sil - sil_old;
                if(diff < DELTA2) convergencia_cont++;
                else convergencia_cont = 0;
                sil_old = sil;

                ngrupos=ngrupos+10;
        }
        ngrupos=ngrupos-10;
        clock_gettime (CLOCK_REALTIME, &t17);
        t_clust = (t17.tv_sec-t12.tv_sec) + (t17.tv_nsec-t12.tv_nsec)/(double)1e9;

        // 2. fase: numero de vectores de cada grupo; analisis campos UNESCO
        // ================================================================
        clock_gettime (CLOCK_REALTIME, &t20);
        analisis_campos(listag, mcam, info_cam);
        clock_gettime (CLOCK_REALTIME, &t21);
        t_camp = (t21.tv_sec-t20.tv_sec) + (t21.tv_nsec-t20.tv_nsec)/(double)1e9;

        // escritura de resultados en el fichero de salida
        // ===============================================
        fd = fopen ("vecpal_s.out", "w");
        if (fd == NULL) {
                printf ("Error al abrir el fichero dbgen_out.s\n");
                exit (-1);
        }

        fprintf (fd,">> Centroides de los clusters\n\n");
        for (i=0; i<ngrupos; i++) {
                for (j=0; j<NDIM; j++) fprintf (fd, "%7.3f", cent[i][j]);
                fprintf (fd,"\n");
        }

        fprintf (fd,"\n\n>> Numero de clusteres: %d. Numero de vectores en cada cluster:\n\n", ngrupos);
        ind = 0;
        for (i=0; i<ngrupos/10; i++) {
                for (j=0; j<10; j++){
                        fprintf(fd, "%6d", listag[ind].nvecg);
                        ind++;
                }
                fprintf(fd, "\n");
        }
        for(i=ind; i<ngrupos; i++) fprintf(fd, "%6d", listag[i].nvecg);
        fprintf(fd, "\n");

        fprintf (fd,"\n\n>> Densidad de los clusters: a[i]\n\n");
        ind = 0;
        for (i=0; i<ngrupos/10; i++) {
                for (j=0; j<10; j++){
                        fprintf (fd, "%9.2f", a[ind]);
                        ind++;
                }
                fprintf (fd, "\n");
        }
        for(i=ind; i<ngrupos; i++) fprintf(fd, "%9.2f", a[i]);
        fprintf(fd, "\n");

        fprintf (fd,"\n\n>> Analisis de campos (medianas) en los grupos\n\n");
```

```
        for (i=0; i<NCAM; i++)
                fprintf (fd,"Campo: %2d - mmax: %4.2f (grupo %2d) - mmin: %4.2f (grupo %2d)\n",
                                i, info_cam[i].mmax, info_cam[i].gmax, info_cam[i].mmin, info_cam[i].gmin);

        fprintf (fd,"\n\n");
        fclose (fd);

        clock_gettime (CLOCK_REALTIME, &t2);
t_escri = (t2.tv_sec-t21.tv_sec) + (t2.tv_nsec-t21.tv_nsec)/(double)1e9;
        texe = (t2.tv_sec-t1.tv_sec) + (t2.tv_nsec-t1.tv_nsec)/(double)1e9;

        // mostrar por pantalla algunos resultados
        // =======================================
        printf ("\n>> Centroides 0, 20, 40...\n");
        for (i=0; i<ngrupos; i+=20) {
                printf ("\n  cent%2d -- ", i);
                for (j=0; j<NDIM; j++) printf ("%5.1f", cent[i][j]);
                printf("\n");
        }

        printf ("\n>> Numero de clusteres: %d. Tamanno de los grupos:\n\n", ngrupos);
        ind = 0;
        for (i=0; i<ngrupos/10; i++) {
                for (j=0; j<10; j++){
                        printf ("%6d", listag[ind].nvecg);
                        ind++;
                }
                printf ("\n");
        }
        for(i=ind; i<ngrupos; i++) printf("%6d", listag[i].nvecg);
        printf("\n");

        printf("\n>> Densidad de los clusters: a[i]\n\n");
        ind = 0;
        for (i=0; i<ngrupos/10; i++) {
                for (j=0; j<10; j++){
                        printf("%9.2f", a[ind]);
                        ind++;
                }
                printf("\n");
        }
        for(i=ind; i<ngrupos; i++) printf("%9.2f", a[i]);
        printf("\n");

        printf ("\n>> Analisis de campos en los grupos\n\n");
        for (i=0; i<NCAM; i++)
                printf ("Campo: %2d - max: %4.2f (grupo %2d) - min: %4.2f (grupo %2d)\n",
                                i, info_cam[i].mmax, info_cam[i].gmax, info_cam[i].mmin, info_cam[i].gmin);

        printf("\n");
        printf("t_lec,%f\n", t_lec);
        printf("t_clust,%f\n", t_clust);
        printf("t_camp,%f\n", t_camp);
        printf("Texe_s,%f\n", texe);
}
```

fun_s.c

```
/*************************************************
 AC - OpenMP -- SERIE
 fun_s.c
 rutinas que se utilizan en el modulo grupopal_s.c
*************************************************/
#include <math.h>
#include <stdlib.h>
#include "defineg.h" // definiciones

/*********************************************************************
 1 - Funcion para calcular la distancia euclidea entre dos vectores
 Entrada: 2 elementos con NDIM caracteristicas (por referencia)
 Salida:  distancia (double)
*********************************************************************/
double gendist (float *vec1, float *vec2){
        // PARA COMPLETAR
        // calcular la distancia euclidea entre dos vectores
        return 0.0;
}

/*****************************************************************************
 2 - Funcion para calcular el grupo (cluster) mas cercano (centroide mas cercano)
 Entrada: nvec  numero de vectores, int
          mvec  vectores, una matriz de tamanno MAXV x NDIM, por referencia
          cent  centroides, una matriz de tamanno ngrupos x NDIM, por referencia
 Salida:  popul grupo mas cercano a cada elemento, vector de tamanno MAXV, por ref.
*****************************************************************************/
void grupo_cercano (int nvec, float mvec[][NDIM], float cent[][NDIM],
                int *popul){
        // PARA COMPLETAR
        // popul: grupo mas cercano a cada elemento
}

/*****************************************************************************
 3 - Funcion para calcular la calidad de la particion de clusteres.
     Ratio entre a y b.
     El termino a corresponde a la distancia intra-cluster.
     El termino b corresponde a la distancia inter-cluster.
 Entrada: mvec    vectores, una matriz de tamanno MAXV x NDIM, por referencia
          listag  vector de ngrupos structs (informacion de grupos generados), por ref.
          cent    centroides, una matriz de tamanno ngrupos x NDIM, por referencia
 Salida:  valor del CVI (double): calidad/bondad de la particion de clusters
*****************************************************************************/
double silhouette_simple(float mvec[][NDIM], struct lista_grupos *listag, float cent[][NDIM],
                float a[]){
    //float b[ngrupos];

    // PARA COMPLETAR

    // aproximar a[i] de cada cluster: calcular la densidad de los grupos;
    //           media de las distancia entre todos los elementos del grupo;
    //           si el numero de elementos del grupo es 0 o 1, densidad = 0
    // ...

    // aproximar b[i] de cada cluster
    // ...
```

```
    // calcular el ratio s[i] de cada cluster
    // ...

    // promedio y devolver
    // ...
    return 0.0;
}

/*****************************************************************************************
 4 - Funcion para relizar el analisis de campos UNESCO
 Entrada:  listag   vector de ngrupos structs (informacion de grupos generados), por ref.
           mcam     campos, una matriz de tamaño MAXV x NCAM, por referencia
 Salida:   info_cam vector de NCAM structs (informacion del analisis realizado), por ref.
*****************************************************************************************/
void analisis_campos(struct lista_grupos *listag, float mcam[][NCAM],
                struct analisis *info_cam){
        // PARA COMPLETAR
        // Realizar el analisis de campos UNESCO en los grupos:
        //    mediana maxima y el grupo en el que se da este maximo (para cada campo)
        //    mediana minima y su grupo en el que se da este minimo (para cada campo)
}

/************************************
   OTRAS FUNCIONES DE LA APLICACION
************************************/
void inicializar_centroides(float cent[][NDIM]){
        int i, j;
        float rand_val;
        srand (147);
        for (i=0; i<ngrupos; i++)
                for (j=0; j<NDIM/2; j++){
                        rand_val = ((rand() % 10000) / 10000.0)*2-1;
                        cent[i][j] = rand_val;
                        cent[i][j+(NDIM/2)] = cent[i][j];
                }
}

int nuevos_centroides(float mvec[][NDIM], float cent[][NDIM], int popul[], int nvec){
        int i, j, fin;
        double discent;
        double additions[ngrupos][NDIM+1];
        float newcent[ngrupos][NDIM];

        for (i=0; i<ngrupos; i++)
                for (j=0; j<NDIM+1; j++)
                        additions[i][j] = 0.0;

        // acumular los valores de cada caracteristica; numero de elementos al final
        for (i=0; i<nvec; i++){
                for (j=0; j<NDIM; j++) additions[popul[i]][j] += mvec[i][j];
                additions[popul[i]][NDIM]++;
        }

        // calcular los nuevos centroides y decidir si el proceso ha finalizado o no (en funcion de DELTA)
        fin = 1;
        for (i=0; i<ngrupos; i++){
                if (additions[i][NDIM] > 0) { // ese grupo (cluster) no esta vacio
                        // media de cada caracteristica
```

```
                        for (j=0; j<NDIM; j++)
                                newcent[i][j] = (float)(additions[i][j] / additions[i][NDIM]);

                        // decidir si el proceso ha finalizado
                        discent = gendist (&newcent[i][0], &cent[i][0]);
                        if (discent > DELTA1){
                                fin = 0;  // en alguna centroide hay cambios; continuar
                        }

                        // copiar los nuevos centroides
                        for (j=0; j<NDIM; j++)
                                cent[i][j] = newcent[i][j];
                }
        }
        return fin;
}
```

defineg.h

```
/*********************************************************
    defineg.h
    definiciones utilizadas en los modulos de la aplicacion
*********************************************************/

#define MAXV 211640     // numero de vectores (palabras)
#define MAX_GRUPOS 100  // numero de clusters
#define NDIM 40         // dimensiones de cada vector
#define NCAM 23         // numero de campos UNESCO

#define DELTA1 0.01
#define DELTA2 0.01
#define MAXIT 10000    // convergencia: numero de iteraciones maximo

extern int ngrupos;

// estructuras de datos

struct lista_grupos // informacion de los clusters
{
 int vecg[MAXV];    // indices de los vectores
 int nvecg;         // numero de vectores en el grupo
};

struct analisis     // resultados del analisis de grupos
{
 float mmax, mmin;  // maximo y minimo de las medianas de las probabilidad de cada campo
 int gmax, gmin;    // grupos con los valores maximo y minimo de las medianas
};
```

fun.h

```
/*************************************************************
    fun.h
    cabeceras de las funciones utilizadas en el modulo gengrupos
*************************************************************/

extern double gendist (float *vec1, float *vec2);
extern void grupo_cercano (int nvec, float mvec[][NDIM], float cent[][NDIM], int *popul);
extern double silhouette_simple(float mvec[][NDIM], struct lista_grupos *listag, float cent[][NDIM], float a[]);
extern void analisis_campos(struct lista_grupos *listag, float mcam[][NCAM], struct analisis *info_cam);

extern void inicializar_centroides(float cent[][NDIM]);
extern int nuevos_centroides(float mvec[][NDIM], float cent[][NDIM], int popul[], int nvec);
```