Introducción

En este informe voy a tratar de explicar el funcionamiento del programa, además de los objetivos de aprendizaje que se intentan poner en práctica con este trabajo. Para ello utilizare gráficas con el rendimiento en distintas condiciones y algunos fragmentos de código para que sea más cómodo entenderlo.

El principal objetivo del trabajo es aprender cómo paralelizar un programa escrito en C con la librería OpenMP para este lenguaje, además de mejorar nuestra capacidad de resolver errores y buscar soluciones sin preguntar todo a los profesores. Y para ello utilizar más google u otro buscador y leer también las páginas oficiales de las librerías y lenguajes que utilizamos para acostumbrarnos a leer ese tipo de instrucciones con más facilidad. Para hacer el proyecto yo lo que hice fue primero descargarme toda la carpeta del proyecto “pgenetica” que se encontraba en la maquina que la universidad nos proporciono (ya que si había mucha gente conectada iba muy lenta) y con la carpeta en mi ordenador personal trabaje en él usando el editor neovim y la terminal kitty para ejecutarlo. Mi portatil cuenta con un i5 de 11 generación de 12 hilos a un maximo de 5GHz, no es tanto como la máquina proporcionada por la universidad la cual cuenta con 64 hilos pero para trabajar en la versión en serie y las primeras versiones paralelas era suficiente y trabajaba más cómodo en mi entorno, y mi entorno es una instalación de Arcolinux con Hypr como gestor de ventanas y el código lo ejecuto dentro de la terminal con el compilador gcc y con un script muy simple para automatizar la compilación y la ejecución el cual recibe un número y ejecuta el programa con ese número de muestras y si no le proporcionas ningún número lo ejecuta entero.

```
#!/bin/zsh
gcc -O2 -o gengrupos_s gengrupos_s.c fun_s.c -lm -fopenmp
export OMP_NUM_THREADS=12
echo "el numero para ejecutar el programa es: " $@
./gengrupos_s dbgen.dat dbenf.dat $@
```

Fundamentos para la correcta elaboración del proyecto

Para elaborar el proyecto hay que tener un mínimo de experiencias en programación aunque esta no debe ser concretamente en C, ya que casi no se utilizan punteros ni otras funciones que tiene C, si que ayuda ya que tendrías una mejor base. Además hace falta cierto nivel en matemáticas ya que hay algunas funciones del programa en las que hay que implementar funciones con sumadores y otros símbolos matemáticos para la que necesitas esa experiencia.
https://www.cprogramming.com/

Pero al tratarse de un proyecto en el que se trata de poner en práctica la paralelización de un programa hay que tener también una base para saber que problemas te puedes encontrar en el proceso. Y conocer las funciones de la biblioteca que usamos para paralelizarlo (OpenMP) la cual tiene una página web con ejemplos y guías básicas para empezar a utilizarla.
https://www.openmp.org/resources/tutorials-articles/

Aplicación

El programa es una simplificación de una programa real en el que se nos dan dos archivos con datos, el primero es “dbgen.dat” el cual contiene 211000 muestras genéticas de elementos patógenos con 40 características genéticas y el otro archivo es “dbenf.dat” y contiene filas de 18 familias de posibles enfermedades que puede ocasionar cada muestra anteriormente mencionada. Es decir cada línea del archivo “dbgen.dat” es una muestra con 40 características y el mismo número de línea del archivo “dbenf.dat” contiene las enfermedades que puede producir esa muestra. Nuestro trabajo será procesar ese banco de datos para crear grupos genéticos y encuadrar cada muestra en el que corresponda, esto depende de la cercanía de sus características genéticas (cuanto más similares sean estarán en el mismo grupo).
A los alumnos nos ha sido entregado el proyecto pero sin paralelizar y con un archivo llamado “fun_s.c” en el cual tenemos que completar 4 funciones, el resto nos ha sido entregado ya hecho pero tenemos que paralelizar también lo que nos ha sido dado.

Las funciones que nos han dejado en blanco para que las completemos nosotros son:

- Gendist: toma dos elementos con NCAR características y nos devuelve la distancia euclidiana entre ambos.Con esta función queremos ver lo parecidos o distintos que son estos dos elementos. Para ello basta con implementar esta fórmula en el programa:
  dis (p, q) = raiz_cuadrada [(p1 - q1) 2 + (p2 - q2) 2 + … + (pn - qn) 2]
  En mi caso lo he implementado de la siguiente forma:
  ‘’’
  ```
  double termino = 0;
    for(int i = 0; i < NCAR; i++){
      termino += pow((elem1[i] - elem2[i]), 2);
    }

    return sqrt(termino);
  ```
  ‘’’
  Siendo elem1 y elem2 p y q respectivamente.
  Esta función no la he paralelizado ya que es un for bastante pequeño y las veces que la uso se encuentra dentro de una región paralela.

- Grupo_cercano: toma el número de elementos el cual es el número de elementos que contiene la matriz elem que se nos pasa además toma una matriz llamada cent y un

array llamado popul. En esta función tenemos que encontrar el grupo más cercano a cada centroide. Para ello llamo a la función gendist con un elemento y con todos los centroides y elijo el índice menor, y esto lo hago con todos los elementos. La función la he metido en una región paralela y dentro de esta le he especificado que hay un for para que lo haga en paralelo, con la configuración estándar funciona bien así que no he visto la necesidad de especificar las variables privadas o compartidas.

- Silhouette_simple: toma una matriz de elementos, una lista de grupos la cual tiene dos atributos siendo uno el número de elementos del grupo y otro la lista de elementos del grupo, una matriz de centroides y una lista de números float.
  Esta función tiene 4 apartados:
  1. Término a[]: este término aproxima la distancia intra-cluster de cada cluster, es decir, la distancia que hay entre cada elemento del cluster, esto para todos los clusters/grupos. Para esto utiliza esta función:

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

     Y la he implementado con tres loops for, uno para recorrer cada grupo dentro de “listag” (el elemento que contiene la lista de grupos) y los otros dos para recorrer la matriz de elementos con cada grupo y cada elemento del grupo para calcular su distancia euclídea y así calcular su media y guardarla en a[].

  2. Término b[]: este término aproxima la distancia inter-cluster de cada cluster utilizando esta función:

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

     Aquí calcula la distancia entre el centroide k y los demás centroides.
  3. Término s[]: este es un ratio entre a y b y nos indica la calidad del cluster, cuanto menor sea a y mayor sea b el cluster tendrá más calidad. Este término se calcula con esta función:

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

  4. Término S: es la media de todos los términos s, y nos indica la calidad de la partición de clusters. Para calcularlo se hace con esta función:

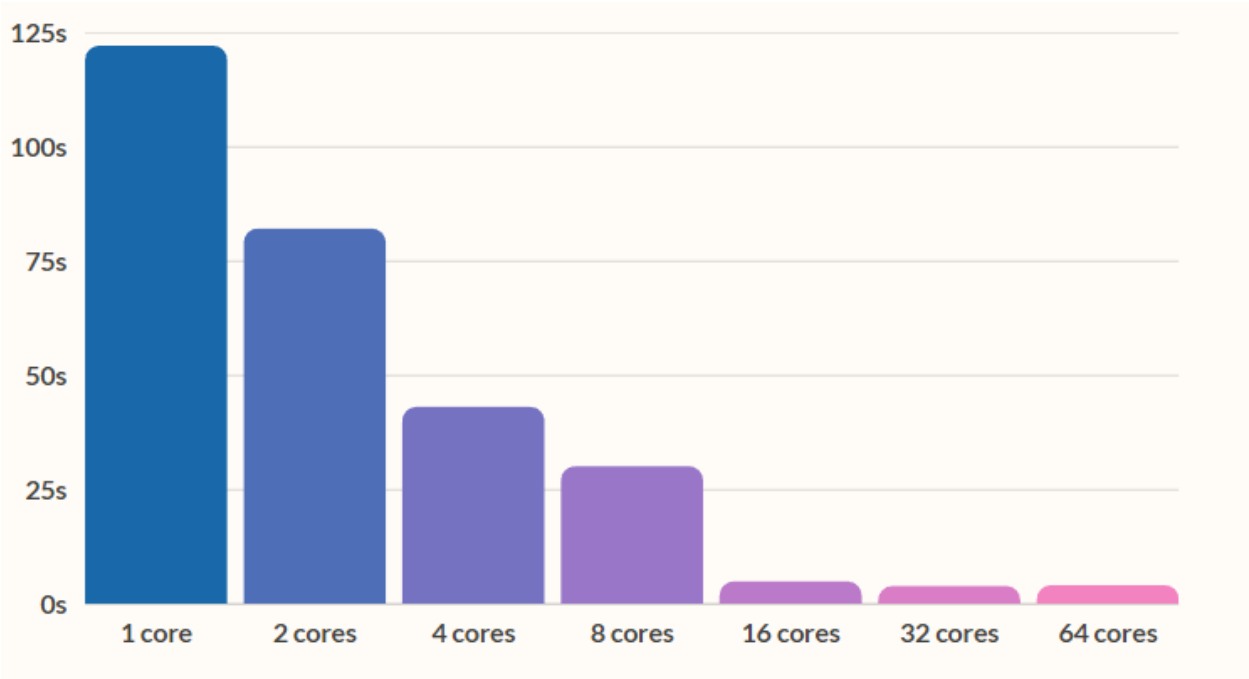
     $$S = \frac{1}{|C|} \sum_{c_k \in C} s(k)$$

     Para paralelizar esta función lo he metido en una región paralela y he

especificado cuando había un for, en las primeras fases intente poner nowait en los for pero no se puede ya que tienen valores que dependen de los anteriores loops.

- Analisis_enfermedades: toma una lista de grupos (listag), una matriz de enfermedades y una estructura de análisis para las enfermedades la cual tiene 4 parámetros, 2 son la mediana máxima y la mínima y otros dos son el grupo en el que se encuentra la mediana máxima y en el que se encuentra la mediana mínima.
  Para esta función primero hay que ordenar todos los elementos de cada grupo para así sacar la mediana más fácilmente de cada grupo. Para ordenarlos simplemente los recorro y con el algoritmo de la burbuja y un elemento auxiliar cambio los valores para que estén en orden. Una vez ordenados coges el valor que se encuentre en el medio sea el número de elementos par o impar.

Rendimiento con distinto número de hilos



Numero de hilos	Tiempo (segundos)
1 hilo	122s
2 hilos	82s
4 hilos	43s
8 hilos	30s
16 hilos	4.5s
32 hilos	3.8s
64 hilos	4.1s

(Estos resultados varían dependiendo de la carga de la máquina en el momento de las pruebas.)

Conclusión

En general el proyecto me ha parecido bastante entretenido al trabajar con una máquina de 64 hilos y al ser en el lenguaje C ya que yo empecé a programar en c++ y me gusta la sintaxis de estos lenguajes. Además el hecho de que tengamos que integrar matemáticas al programa también me parece interesante ya que te hace ver como se compenetran esas dos disciplinas.

En algunos puntos me ha parecido un poco confuso el enunciado del proyecto y no tenía muy claro que debía hacer o qué estructura me estaba pidiendo usar en alguna parte (en la parte de analisis_enfermedades no entendía lo de la mediana máxima y mínima y en la de silhouette_simple la b no sabía si queríais que lo comparase con los centroides o no) pero también tengo que decir que solo he podido asistir a dos clases en las que trabajamos este proyecto así que puede que lo explicaseis mas a fondo después pero no me fue posible asistir así que lo trabaje en mi portatil y eso hizo bastante más confuso esas partes. Los resultados no me dan igual que los que aparecen en los archivos de resultado dados por los profesores por esas confusiones con el enunciado pero ya tenia que pasar a paralelizar porque si no no me daba tiempo y ya que el proyecto se centraba en la paralelización no lo vi del todo mal.

El proyecto en general me ha ayudado a entender un poco mejor cómo se hace un programa más del ámbito científico con fórmulas y algoritmos y me ha parecido bastante entretenido aunque si que me hubiese gustado algo más de libertad ya que lo que es de programa solo hemos hecho 4 funciones y paralelizar todo. Se que no está todo lo bien que debería pero la verdad es que hacerlo solo se me ha complicado un poco más de lo que pensaba pero creo que sí que he aprendido bastante así que estoy razonablemente contento de lo que he avanzado el proyecto por mi cuenta.