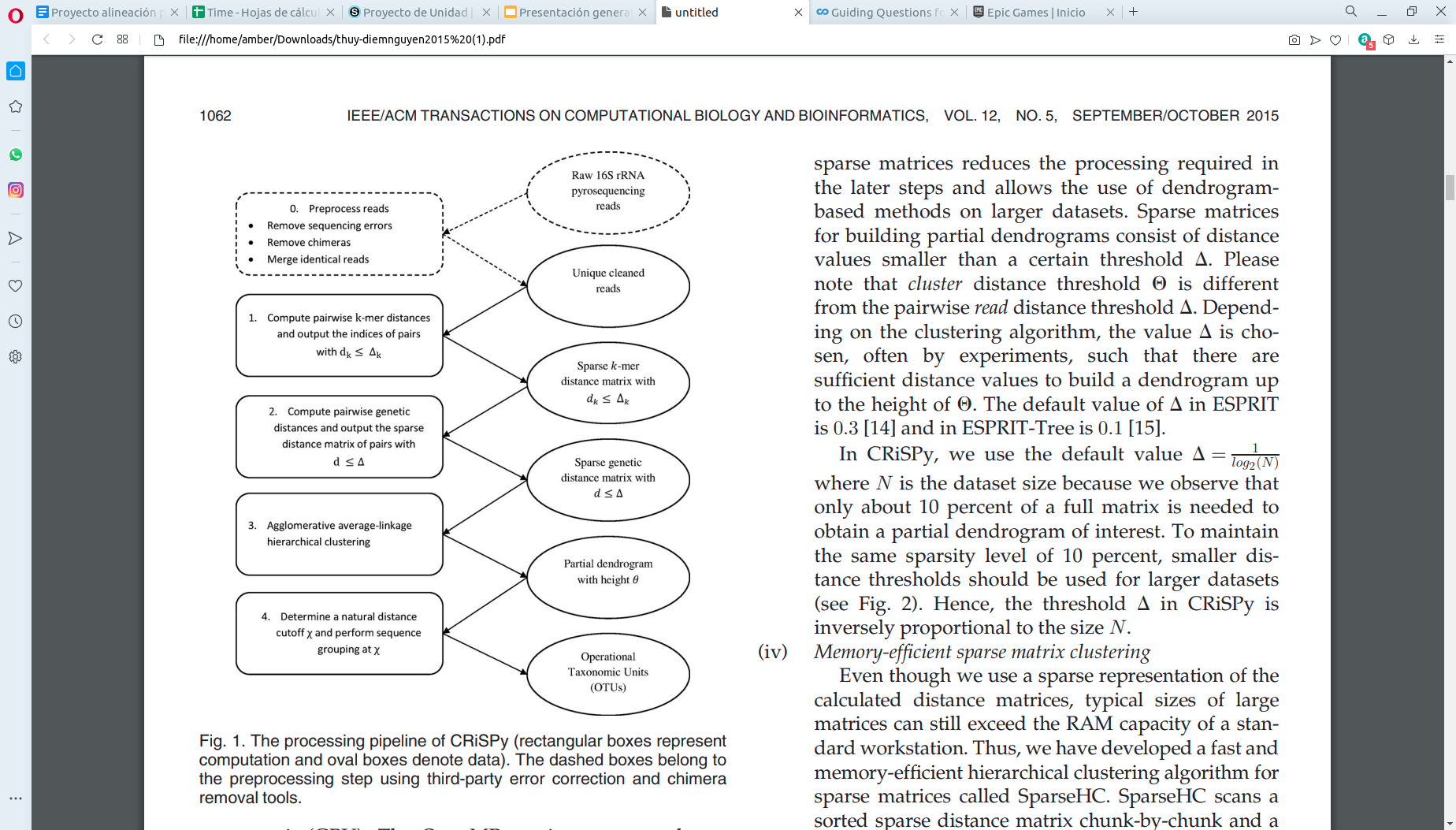
Algoritmos para el alineamiento de secuencias de ADN y proteínas

**Introducción**

El taxonomic profiling está diseñado para determinar qué organismos están dentro de una muestra y cuán abundantes son, la muestra puede ser secuenciada mediante una técnica llamada pyrosequencing que provee grandes cantidades de datos en muy poco tiempo, por esta razón es necesario utilizar métodos que puedan procesar esta información de manera rápida.

Una de los opciones para realizar esto está detallada en el paper “Efficient and Accurate OTU Clustering with GPU-Based Sequence Alignment and Dynamic Dendrogram Cutting”, donde se realizan comparaciones entre secuenciaciones para conocer su distancia genética con pasos intermedios como se muestra en la figura 1.



Nosotros comenzamos a trabajar con la implementación de un código partiendo de las lecturas limpias, y nos limitamos a realizar la matriz de distancias de k-meros, enfocándonos en la paralelización de este para eficientar tiempos.

**Planteamiento de problema**

Se requiere implementar un código para obtener una matriz con la distancia de k-meros de un data de secuenciación, esta distancia está definida como el número de matches entre substrings de tamaño k de dos strings. Para esto se deben realizar otras operaciones primero, las cuales se definen en métodos.

Una vez implementado el código serial, se procederá a su paralelización utilizando OpenMP de manera eficiente usando lo aprendido en el curso.

**Objetivos**

* Obtener una matriz de k-mer distances para alineamiento genético mediante un programa en serie
* Reducir el tiempo de cómputo al paralelizar el programa para comparar tiempos

**Metas**

* Tener el cálculo de los speed ups al aumentar el número de datos
* Conocer el efecto del tamaño de los substrings en serie y en paralelo

**Hipótesis**

Al paralelizar el código utilizando OpenMP este tardará menos tiempo en computar, obteniendo un speed up acorde al número de hilos de la computadora y de los datos utilizados

**Contribuciones**

* Se paralelizó la comparación de lecturas, asignando un hilo a un string y utilizando ese mismo hilo para hacer las comparaciones de ese string hacia los demás.
* Se realizaron únicamente las comparaciones necesarias.
* Se utilizó un método que reduce el tiempo de comparación de strings de O(N²L) a O(L).

**Trabajos relacionados**

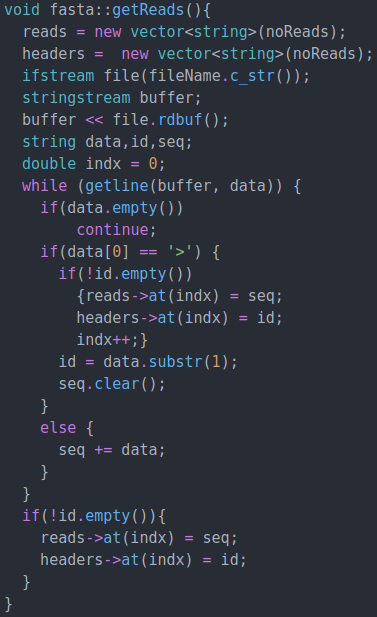
Actualmente existen muchas implementaciones y algoritmos para generar una matriz de distancias de kmeros, siendo un claro ejemplo la principal referencia de este proyecto, “*Efficient and Accurate OTU Clustering with GPU-Based Sequence Alignment and Dynamic Dendrogram Cutting*” donde el objetivo era paralelizar con CUDA aprovechando el poder de GPU’s para hacer análisis filogenéticos y que involucra la obtención de la distancia de kmeros.

Además, en lenguajes ampliamente empleados dentro del campo de la bioinformática como R existen librerías que permiten obtener la distancia de kmeros, como lo son “kmer” y “Biostrings” de Bioconductor, aunque también en lenguajes como python existen librerías como khmer.

**Métodos**

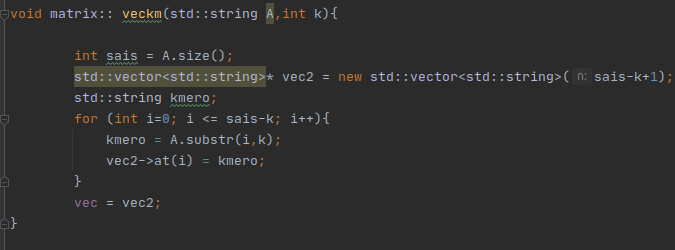
1. **Lectura de datos**

Se determina el número de lecturas en el archivo al contar la cantidad de líneas que inician con “>” y se identifican las que no lo tienen para agregarlas al vector de secuencias.



1. **División en substrings**

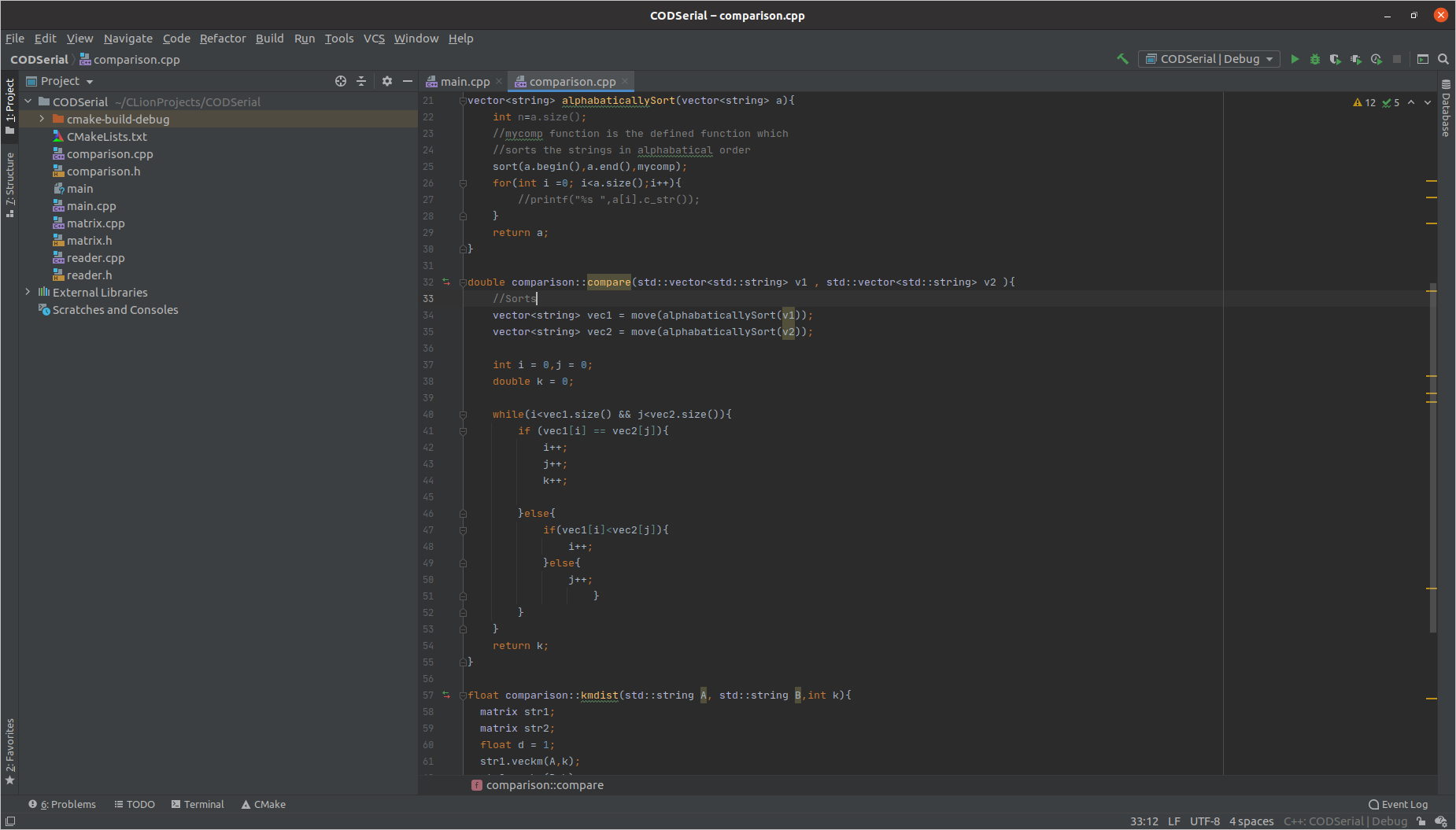
Dada una string y un tamaño K, se divide la string en substrings de este tamaño y se asignan a un vector, por ejemplo, dada la string “abcde” se transforma en un vector con las substrings “abc” “bcd” “cde”.



Se paralelizó este método, sin embargo, no se obtuvo ninguna eficiencia significativa, por lo que finalmente se decidió utilizar de manera serial.

1. **Ordenamiento alfabético ascendente de vectores de substrings y comparación entre ellos**

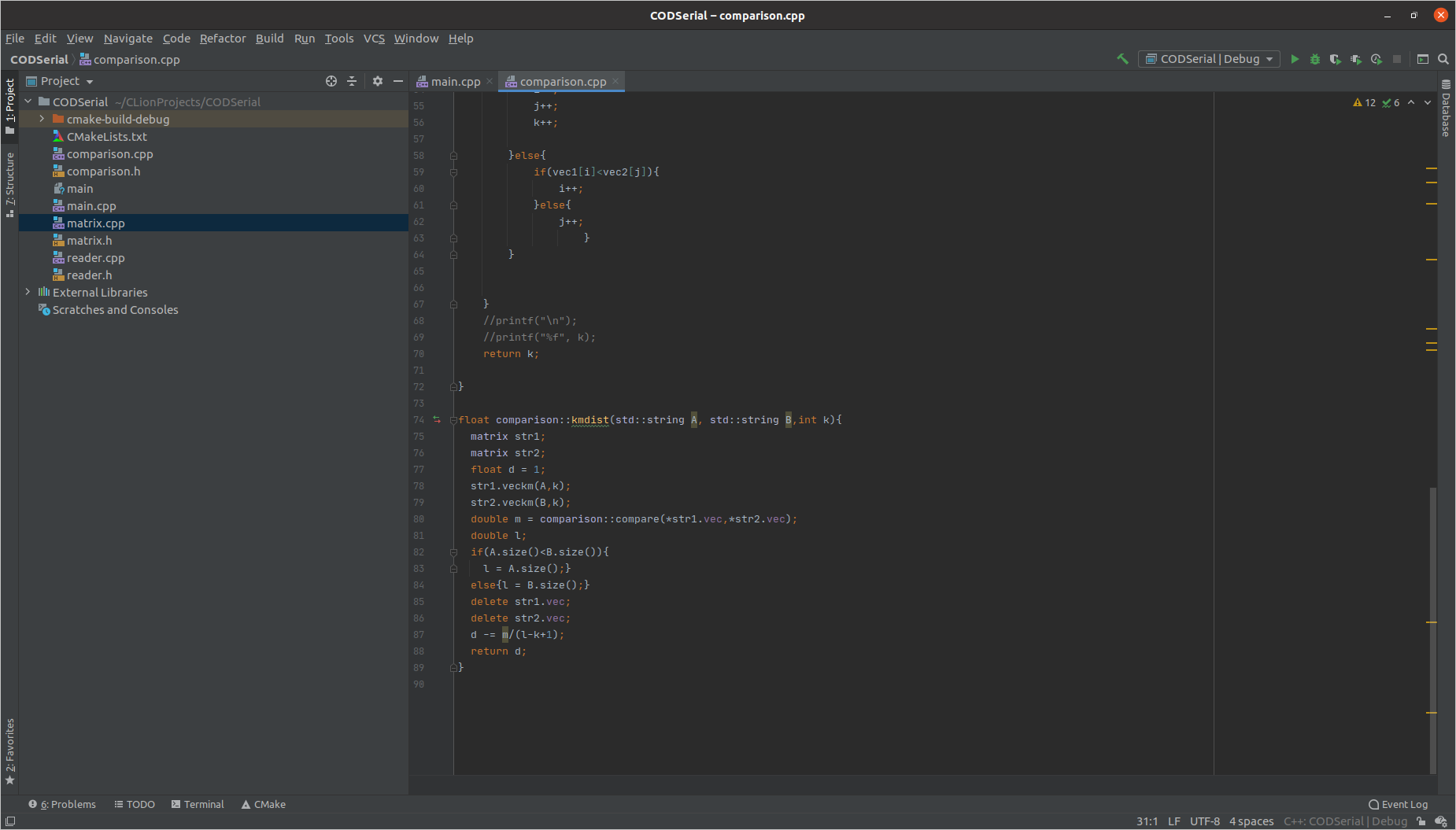
Este método regresa el número de matches que se da al comparar dos vectores de substrings, los cuales se ordenan alfabéticamente ascendentemente. Si hay un match, ambos vectores aumentan un índice, y si no, se aumenta el índice de la substring que esté antes en el alfabeto; esto termina hasta que se acaban los índices de cualquiera de los dos vectores.



1. **Cálculo de distancia de k-meros entre dos strings**

Se calcula la distancia de k-meros entre los dos strings usando como base la siguiente fórmula:



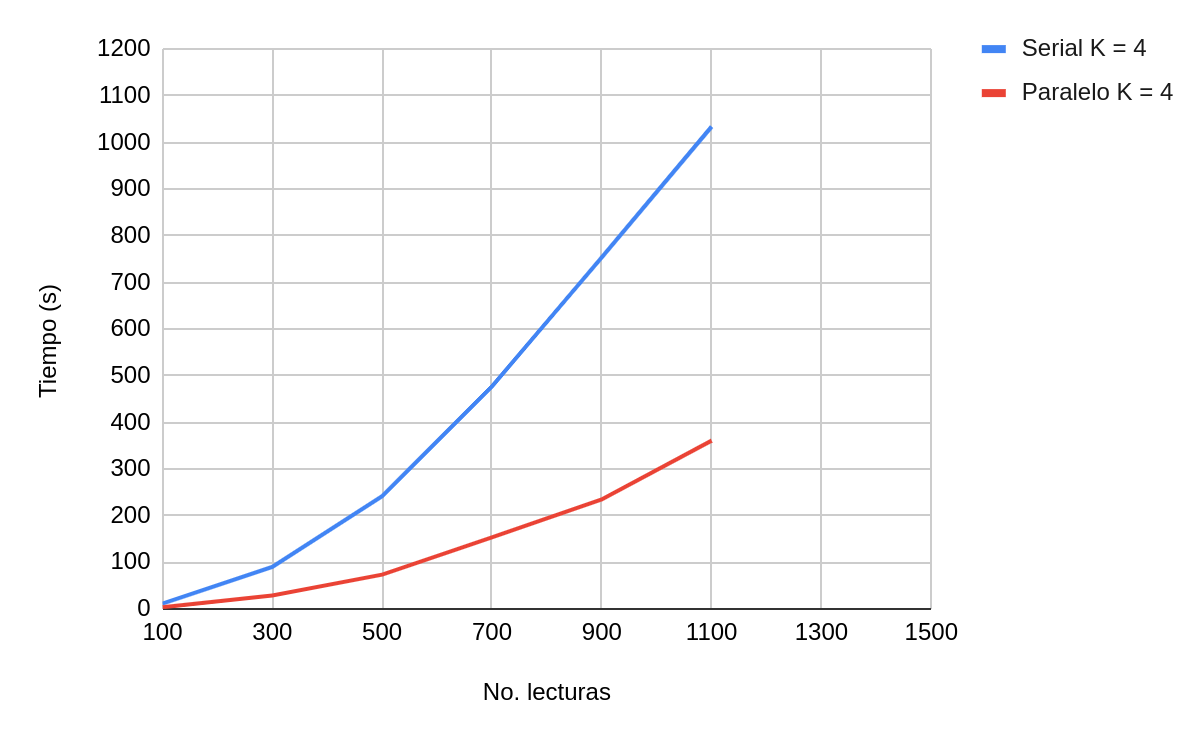


1. **Generación de matriz**

Para generar la matriz se utilizaron dos for anidados, donde se asigna cada hilo a un string y se realiza para este cada una de las comparaciones entre el data. Este fue el único método que se paralelizó por dos razones, primeramente es el que aporta más carga para los procesadores y es fácilmente paralelizable utilizando OpenMP.

**Resultados**

|  |  |  |  |
| --- | --- | --- | --- |
| No. reads | Tiempo Serial | Tiempo Paralelo | Speed Up |
| 100 | 11.3314 | 3.4748 | 3.261022217 |
| 300 | 89.8546 | 28.8557 | 3.113928964 |
| 500 | 241.3707 | 73.5363 | 3.282334031 |
| 700 | 476.9529 | 153.5434 | 3.106306751 |
| 900 | 753.635 | 234.6097 | 3.212292586 |
| 1100 | 1033.88 | 360.7082 | 2.866250338 |

****

**Conclusiones**

El liberar memoria es necesario para hacer espacio para próximas comparaciones y poder correr archivos grandes sin que se trabe la computadora.

Es importante revisar que la implementación paralela esté ejecutándose de manera paralela y no solo uno de los núcleos de la computadora esté activo para sacar provecho al paralelismo.

En implementaciones computacionalmente exigentes y tardadas el uso de estrategias como la paralelización es muy importante para disminuir lo más posible el tiempo de cómputo.

**Referencias**

* Thuy-Diem Nguyen, Schmidt, B., Zejun Zheng, & Chee-Keong Kwoh. (2015). *Efficient and Accurate OTU Clustering with GPU-Based Sequence Alignment and Dynamic Dendrogram Cutting. IEEE/ACM Transactions*