

Similitud de cadenas de ADN mediante el algoritmo de alineamiento nw

Grupo 21



Ignacio Gomis Lli

Lidia Montero Egidos

Sara Monzó Bravo

Paul Vargas Hurtado

Universitat de valència – Escola tècnica superior d’enginyeria

Contenido

[CP1: Análisis del algoritmo 3](#_Toc513061255)

[1-Objetivo y utilidad del algoritmo 3](#_Toc513061256)

[2-Descripción del algoritmo 4](#_Toc513061257)

[3-Análisis de coste y almacenamiento en memoria 6](#_Toc513061258)

[Coste computacional del algoritmo: 6](#_Toc513061259)

[Almacenamiento en memoria: 6](#_Toc513061260)

[4-Análisis de dependencia de datos 8](#_Toc513061261)

[5-Propuestas de paralelización 9](#_Toc513061262)

[Anexo: Código 11](#_Toc513061263)

[Anexo: Tiempo dedicado 18](#_Toc513061264)

[Anexo: Autoevaluación 19](#_Toc513061265)

[Práctica 0 20](#_Toc513061266)

[1-Tamaños de memoria 20](#_Toc513061267)

[2-Tiempos sin optimizar 21](#_Toc513061268)

[3-Tiempos optimizados 22](#_Toc513061269)

[4-Código 23](#_Toc513061270)

[5-Tiempo de trabajo 30](#_Toc513061271)

[Anexo: Autoevaluación 31](#_Toc513061272)

[CP2: Propuesta de implementación con OpenMP 32](#_Toc513061273)

[1-Descripción del algoritmo paralelo 32](#_Toc513061274)

[2-Implementación del algoritmo 35](#_Toc513061275)

[3-Analisis del algoritmo 47](#_Toc513061276)

[Coste computacional 47](#_Toc513061277)

[Coste de memoria 47](#_Toc513061278)

[Estimaciones 47](#_Toc513061279)

[Anexo: Tiempo trabajado 49](#_Toc513061280)

[Anexo: Autoevaluación 50](#_Toc513061281)

[Practica 1 51](#_Toc513061282)

[1-Tiempos de ejecución y MOPS 51](#_Toc513061283)

[2-Estudio de la aceleración y eficiencia 54](#_Toc513061284)

[3-Estudio del entrelazado 57](#_Toc513061285)

[6-Tiempo de trabajo 71](#_Toc513061286)

[Anexo: Autoevaluación 72](#_Toc513061287)

[CP3: Propuesta de implementación con MPI 73](#_Toc513061288)

[1-Descripción del algoritmo paralelo 73](#_Toc513061289)

[2-Implementacion del algoritmo 77](#_Toc513061290)

[3-Análisis del algoritmo 88](#_Toc513061291)

[Coste computacional 88](#_Toc513061292)

[Estimaciones 88](#_Toc513061293)

[Anexo: Tiempo trabajado 89](#_Toc513061294)

[Anexo: Autoevaluación 90](#_Toc513061295)

[Anexo: Referencias 91](#_Toc513061296)

# CP1: Análisis del algoritmo

## 1-Objetivo y utilidad del algoritmo

El objetivo del algoritmo Needleman-Wunsch, publicado en 1970, es obtener el mejor alineamiento de dos cadenas de proteínas o nucleótidos, (Needleman, S. and Wunsch, C. (1970)). Nosotros a partir de este algoritmo, obtendremos el porcentaje de similitud de este alineamiento.

Un ejemplo de esto sería si comparásemos AAAGT con AAGAG nos devolvería que la mejor similitud serían AA-AGT con AAGAG- con 4 coincidencias.

La principal utilidad de este algoritmo está en el campo de la bioinformática, para la comparación de muestras, (Wikipedia, 2018). Entre otras posibilidades, puede servir para ver la cercanía del ADN del ser humano a otros seres o comparar las mutaciones de ADN entre diversos individuos de una especie,

Entrada del algoritmo: Dos cadenas de texto en formato fasta (.fa o .fasta), selección de valores numéricos para coincidencias, fallos y huecos. Nosotros en el algoritmo fijaremos estos valores como (2,-2,-1)

Salida del algoritmo: En nuestro proyecto mostraremos el porcentaje de coincidencia entre cadenas, pero el algoritmo normalmente devuelve las dos cadenas con su mejor alineamiento.

## 2-Descripción del algoritmo

El algoritmo se basa en programación dinámica, al resolver el problema principal como un conjunto de subproblemas y luego escoger la mejor combinación como el resultado del problema principal, (Wikipedia, 2018).

Por tanto, inicialmente se construye una tabla como la siguiente:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | G | T | A | C | A | T | A |
|  | 0 | -1 | -2 | -3 | -4 | -5 | -6 | -7 |
| G | -1 |  |  |  |  |  |  |  |
| A | -2 |  |  |  |  |  |  |  |
| T | -3 |  |  |  |  |  |  |  |
| T | -4 |  |  |  |  |  |  |  |
| A | -5 |  |  |  |  |  |  |  |
| C | -6 |  |  |  |  |  |  |  |
| A | -7 |  |  |  |  |  |  |  |

En la cual los valores de la primera fila y segunda fila son la suma del anterior con la penalización por hueco (Que establecimos como -1) y en la posición 0,0 asignamos un 0.

M[0,0]=0

M[i,0]=M[i-1,0]+Hueco

M[0,j]=M[0,j-1]+Hueco

Luego, se completa la matriz, para cada celda se realizan las siguientes operaciones:

M[i,j]=Max(M[i-1,j]+Hueco , M[i,j-1]+Hueco , M[i-1,j-1]+Función coincidencia)

Función coincidencia= Si(Letra1=Letra2) Entonces (Valor coincidencia) Sino (Valor Fallo)

Nosotros hemos dado los valores siguientes: Valor coincidencia=2, valor fallo = -2

De una manera un poco más formal:

Además, se debe anotar en cada celda como se obtiene ese máximo, es decir, si viene de una diagonal, de lateral o arriba, y no solo una, sino que si hay empates se anotan varios. Estas direcciones sirven después para reconstruir la mejor cadena.

Tras hacer el cálculo tenemos la matriz así:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | G | T | A | C | A | T | A |
|  | 0 | -1 | -2 | -3 | -4 | -5 | -6 | -7 |
| G | -1 | 2 | 1 | 0 | -1 | -2 | -3 | -4 |
| A | -2 | 1 | 0 | 3 | 2 | 1 | 0 | -1 |
| T | -3 | 0 | 3 | 2 | 1 | 0 | 3 | 2 |
| T | -4 | -1 | 2 | 1 | 0 | -1 | 2 | 1 |
| A | -5 | -2 | 1 | 4 | 3 | 2 | 1 | 4 |
| C | -6 | -3 | 0 | 3 | 6 | 5 | 4 | 3 |
| A | -7 | -4 | -1 | 2 | 5 | 8 | 7 | 6 |

Finalmente, a partir de la última casilla, se obtienen todos los caminos generados al seguir las direcciones anteriormente anotadas. Entonces recorriendo, cada vez que se encuentra una diagonal, es una coincidencia y si encuentra una horizontal o una vertical, es un hueco (si es una lateral el hueco va en la segunda cadena, sino el hueco se coloca en la primera). Hay que señalar, que lo realmente importante este camino pase por el mayor valor de la matriz, (fuente requerida), sin embargo, al empezar desde la última celda, como esta depende de todas las celdas anteriores, tendrá un camino hasta la misma, y nos ahorra el tener que llevar un control de cuál es la casilla de valor máximo

Para nuestro ejemplo solo se genera un camino:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | G | T | A | C | A | T | A |
|  | 0 | -1 | -2 | -3 | -4 | -5 | -6 | -7 |
| G | -1 | 2 | 1 | 0 | -1 | -2 | -3 | -4 |
| A | -2 | 1 | 0 | 3 | 2 | 1 | 0 | -1 |
| T | -3 | 0 | 3 | 2 | 1 | 0 | 3 | 2 |
| T | -4 | -1 | 2 | 1 | 0 | -1 | 2 | 1 |
| A | -5 | -2 | 1 | 4 | 3 | 2 | 1 | 4 |
| C | -6 | -3 | 0 | 3 | 6 | 5 | 4 | 3 |
| A | -7 | -4 | -1 | 2 | 5 | 8 | 7 | 6 |

Por tanto, el alineamiento, que tiene 5 coincidencias de 7, es:

G-T-ACATA

GATTACA—

Al emplear el algoritmo utilizaremos cadenas reales de ADN obtenidas del centro nacional de información en biotecnología (National Center for Biotechnology Information,(n.d.).)

## 3-Análisis de coste y almacenamiento en memoria

### Coste computacional del algoritmo:

Desglosado por funciones del programa:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Función | Lecturas Matriz | Escrituras Matriz | Lecturas Cadenas | | Escrituras Cadenas | Operaciones | |
| CargarFichero | 0 | 0 | 2\*N | | 2\*N | 0 | |
| InicializarMatriz | 0 | 4\*(N+M+1) | 0 | | 0 | 0 | |
| CalcularCasilla | 3 | 2 | 4 | | 0 | 19 | |
| CompletarMatriz | 3NM | 2NM | 4NM | | 0 | 19NM | |
| AuxGetRuta  (caso mejor) | 0 | 0 | 0 | | 0 | 5 | |
| AuxGetRuta  (caso esperado) | 3 | 0 | 0 | | 0 | 13 | |
| AuxGetRuta  (caso peor) | 3 | 0 | 0 | | 0 | 15 | |
| GetRuta  (Caso óptimo) | 3\*min(N,M) | 0 | 0 | | 0 | 13\*min(N,M) | |
| GetRuta  (Caso peor, sin poda) | Problema NP-Completo | | | | | | |
| Orden Esperado | N\*M | N\*M | N\*M | N | | | N\*M |

Queremos señalar que tenemos preparadas dos versiones de CalcularCasilla, para probar cual tiene mejor rendimiento en el ordenador de laboratorio. La principal radica en que en una de las versiones no empleamos ningún salto condicional, lo cual nos genera algunas operaciones y lecturas adicionales, pero elimina las paradas de procesador por fallos de predicción. Por otro lado, al observar que la cantidad de operaciones y lecturas podía reducirse con dos operaciones de máximo entre dos valores, comprobaremos qué penaliza menos.

En segundo lugar, el problema de obtener la ruta hemos observado que se trata de un problema NP-Completo, cuyo coste escala de manera exponencial con la talla del problema. Los problemas de obtención de mejores rutas suelen ser NP-Completos (Por ejemplo, el problema del viajante) pero le hemos añadido una función de poda que limita la exploración a las cercanías de la diagonal, por lo que los valores esperados son cercanos al caso óptimo.

### Almacenamiento en memoria:

Si consideramos despreciables los costes de memoria de variables auxiliares, Strings de entrada y contadores de bucles y nos limitamos a la matriz de datos, el consumo en memoria es el siguiente:

Dados dos String de tamaño N y M, la matriz tiene un tamaño de N+1 x M+1, y estará compuesta de una estructura que almacena tres booleanos y un valor numérico. Este valor lo establecemos actualmente como un Int, pero en un futuro, si se diera el caso de que superásemos el máximo valor para el entero, podría pasar a un Double.

Coste de celda con Int: 8 Bytes

Coste de celda con Double: 16 Bytes

(Valores obtenidos con la función Sizeof (Struct Celda))

Por tanto, el tamaño de la matriz es de:

8\*(N+1)\*(M+1)= (8N+8)\*(M+1)=8NM+8N+8M+8 Bytes

Por ejemplo, si ponemos N=M=10^7 (máximo valor puesto en el enunciado) nuestro coste será: 800.000.160.000.008 Bytes = 800TB

Este valor es ridículamente grande, por tanto tenemos la siguiente formula, que nos calcula el tamaño máximo de N y M para que la matriz sea computable, según su memoria:

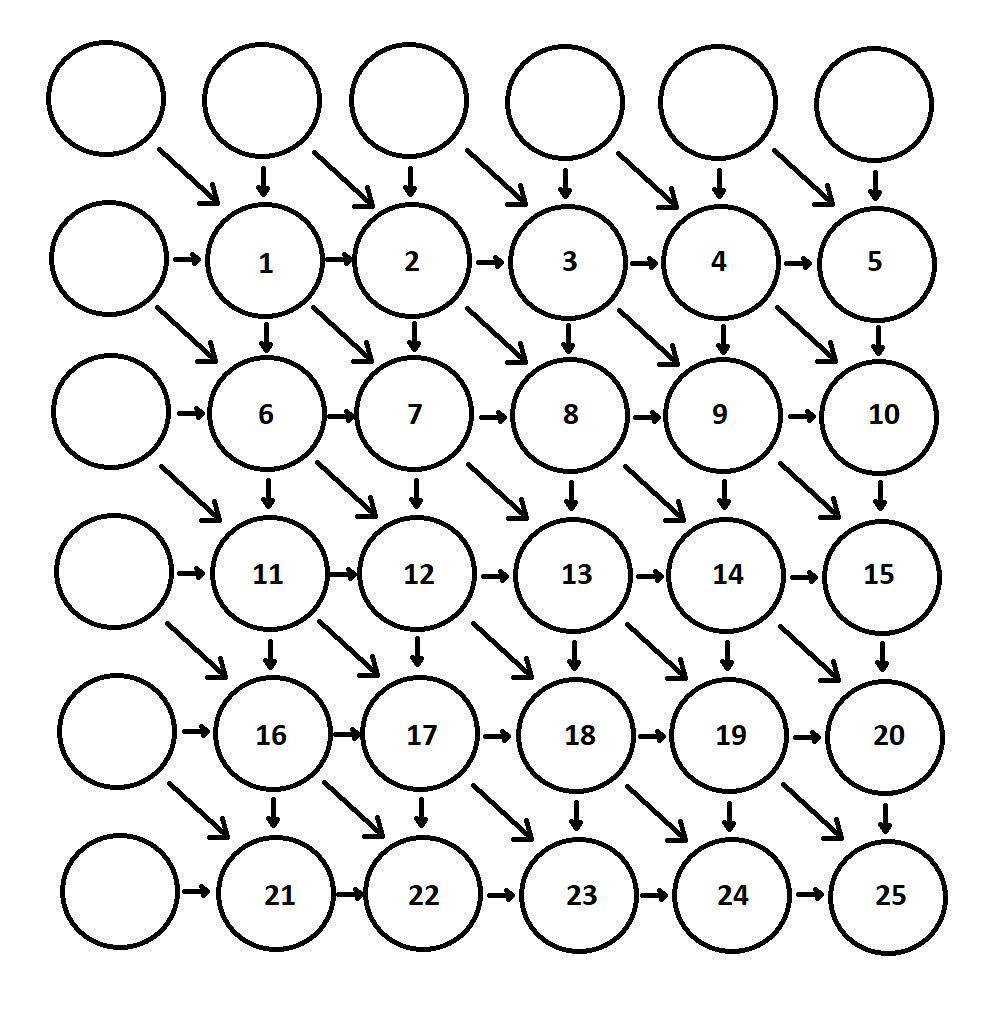
N=M< SQRT((Memoria-Margen)/8)

Siendo Margen, de tamaño suficiente para incluir dos String de Tamaño N y M y a lo sumo 1KB para variables temporales.

Al poder configurar la entrada de parámetros, se podría estudiar una cadena más larga, con varias ejecuciones distintas sobre diversas partes de la cadena, al tratar el problema total como un subconjunto de cadenas se puede obtener una aproximación al resultado real.

## 4-Análisis de dependencia de datos

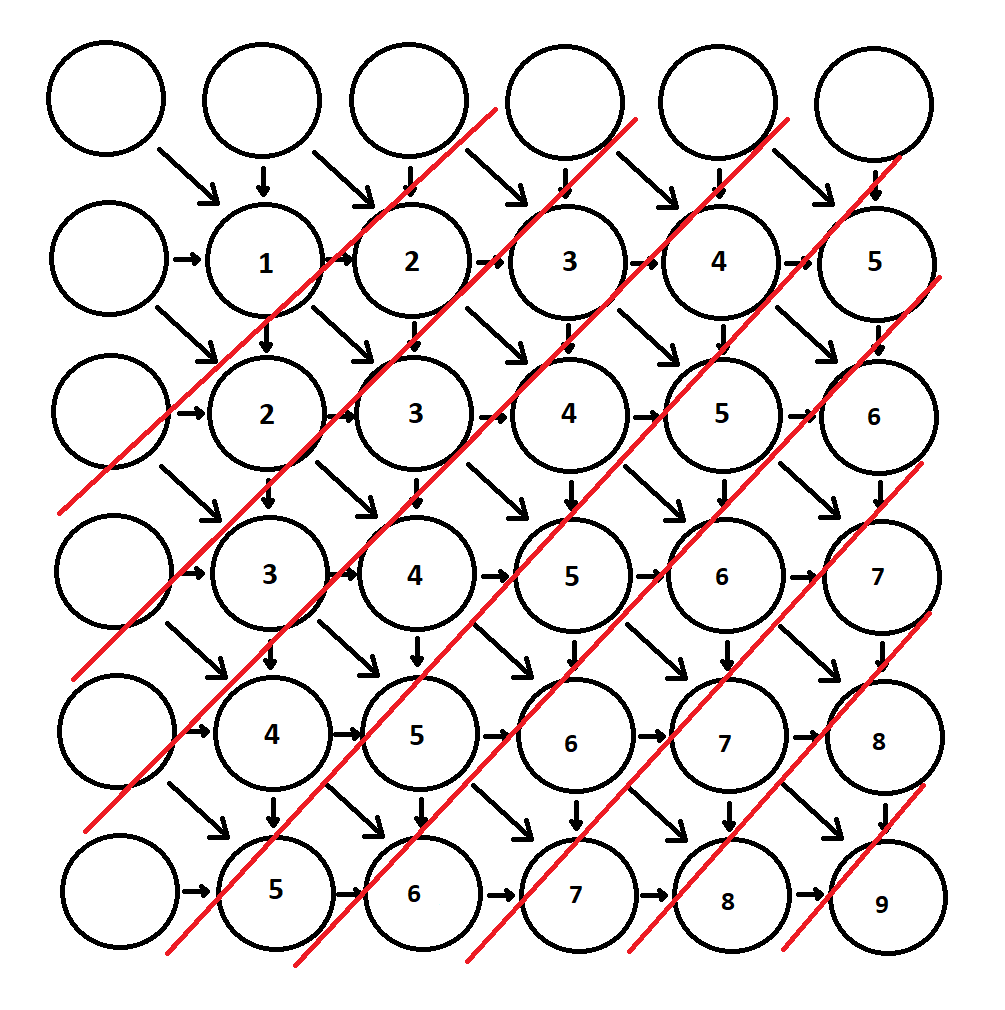
Como se puede observar, la dependencia de datos de una celda con índices [i,j] son las celdas [i-1,j] , [i,j-1] y [i-1,j-1], así como el índice [i-1] de una de las cadenas y el índice [j-1] de la otra cadena, sin embargo, las cadenas solo resultarán leídas, mientras que las celdas se escriben iterativamente, y solo una vez cada una de ellas, causando que una vez terminada una sección esta pueda ser leída sin riesgo.



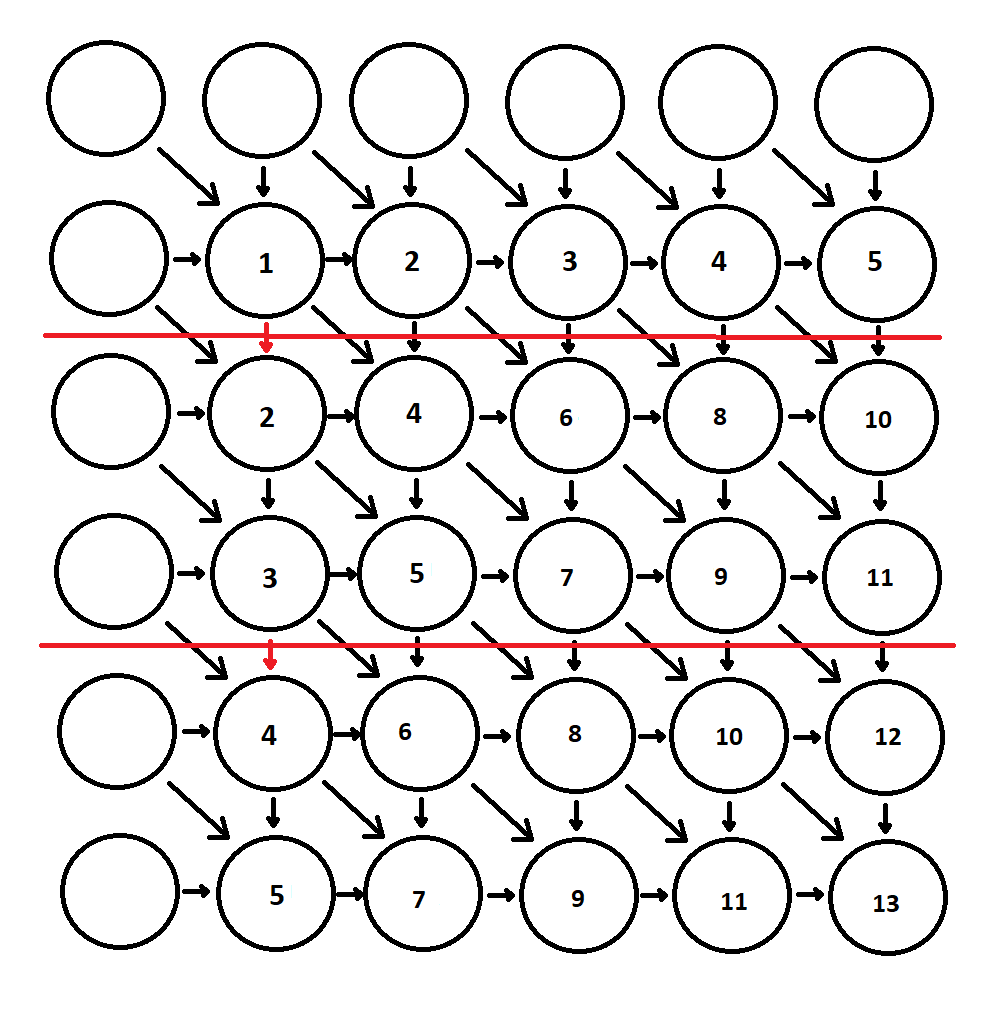
## 5-Propuestas de paralelización

Esencialmente, al tratarse de una matriz, la primera estrategia de paralelización que podemos pensar es por paralelismo de datos, pero al tener que generarla por partes podemos considerar que también la trataremos una parte de paralelismo por flujo de datos, siendo un.

En primer lugar, la paralelización ideal seguiría un esquema como el siguiente:



Sin embargo, la alternativa que proponemos, para tener fragmentada la matriz en tantas particiones como procesadores disponibles, sería usando el siguiente esquema (O la trasposición de esta misma matriz) de tal manera que cada procesador se encargue de una cantidad igual de celdas, a excepción del primero que calculará las que queden sin emparejar en grupos del tamaño de partición. Esta carga no se le puede asignar al último porque entonces podría adelantar al anterior, al tener una menor cantidad de celdas. El siguiente hilo sería lanzado al finalizar la primera columna de la sección.



## Anexo: Código

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <stdbool.h>  #include <string.h>  #include <time.h>  /\*Struct con el valor de la celda y las direcciones  Si las direcciones son verdadero, entonces ha heredado de esa direccion  \*/  struct Celda  **{**  int score**;**  short dir**;** //Array de booleanos  **};**  char**\*** CargarFichero**(**char**\*,**unsigned**,**unsigned**);**  struct Celda**\*\*** inicializarMatriz**(**unsigned**,** unsigned**);**  void CompletarMatriz**(**char**\*,**char**\*,**struct Celda**\*\*);**  void CalcularCasilla**(**unsigned**,** unsigned**,** bool**,** struct Celda**\*\*);**  unsigned GetRuta**(**struct Celda**\*\*,**unsigned**,**unsigned**);**  int AuxGetRuta**(**struct Celda**\*\*,** unsigned**,** unsigned**,** int**,** unsigned**\*);**  /\*Maximo entre dos valores\*/  unsigned maxU**(**unsigned arg1**,** unsigned arg2**)**  **{**  **if(**arg1**>**arg2**){**  **return** arg1**;**  **}**  **else**  **return** arg2**;**  **}**  int maxI**(**int arg1**,** int arg2**)**  **{**  **if(**arg1**>**arg2**){**  **return** arg1**;**  **}**  **else**  **return** arg2**;**  **}**  /\*Funcion para pasar a mayuscula una cadena. Obtenida de:  https://stackoverflow.com/questions/35181913/converting-char-to-uppercase-in-c  \*/  void Mayus**(**char **\*** temp**)** **{**  char **\*** name**;**  name **=** strtok**(**temp**,**":"**);**  char **\***s **=** name**;**  **while** **(\***s**)** **{**  **\***s **=** toupper**((**unsigned char**)** **\***s**);**  s**++;**  **}}**  /\*\*  \* Main Lleva control del orden de ejecucion de las funciones del algoritmo y la entrada de parametros  \* @author Lidia y Nacho  \* @date 8/2/2018  \* @param Nombre o ruta de Fichero1 (Necesario)  \* @param Nombre o ruta de Fichero2 (Necesario)  \* @param Tamano de cadena de fichero1 (Opcional o ambos ficheros si falta siguiente parametro)  \* @param Tamano de cadena de Fichero2 (Opcional)  \* @param Inicio de cadena de Fichero1 (Opcional o ambos ficheros si falta siguiente parametro)  \* @param Inicio de cadena de Fichero2 (Opcional)  \*/  int main**(** int argc**,** char **\***argv**[]** **)**  **{**  unsigned T1**,**T2**,**I1**,**I2**;**  **switch(**argc**)**  **{**  **case** 3**:** //Ambos desde el principio hasta lo que coja  T1**=**100**;**T2**=**100**;**I1**=**0**;**I2**=**0**;**    **break;**  **case** 4**:** //Ambos con Tamaño arg[3]  T1**=**atoi**(**argv**[**3**]);**T2**=**T1**;**I1**=**0**;**I2**=**0**;**  **break;**  **case** 5**:** //Cada uno con Tamaño arg[3] y arg[4]  T1**=**atoi**(**argv**[**3**]);**T2**=**atoi**(**argv**[**4**]);**I1**=**0**;**I2**=**0**;**  **break;**  **case** 6**:** //Cada uno con Tamaño arg[3] y arg[4] empezando desde arg[5]  T1**=**atoi**(**argv**[**3**]);**T2**=**atoi**(**argv**[**4**]);**I1**=**atoi**(**argv**[**5**]);**I2**=**I1**;**  **break;**  **case** 7**:** //Cada uno con Tamaño arg[3] y arg[4] empezando desde arg[5] y arg[6]  T1**=**atoi**(**argv**[**3**]);**T2**=**atoi**(**argv**[**4**]);**I1**=**atoi**(**argv**[**5**]);**I2**=**atoi**(**argv**[**6**]);**  **break;**  **default:** //Instrucciones de uso  printf**(**"Error en introduccion de datos:\n Se pueden introducir entre 2 y 6 argumentos:\n2 argumentos:\n Fichero\_1 Fichero\_2\n"**);**  printf**(**"3 argumentos:\n Fichero\_1 Fichero\_2 Tamano\_maximo\n"**);**  printf**(**"4 argumentos:\n Fichero\_1 Fichero\_2 TamanoMax\_Cadena1 TamanoMax\_Cadena2\n"**);**  printf**(**"5 argumentos:\n Fichero\_1 Fichero\_2 TamanoMax\_Cadena1 TamanoMax\_Cadena2 Inicio\_Cadenas\n"**);**  printf**(**"6 argumentos:\n Fichero\_1 Fichero\_2 TamanoMax\_Cadena1 TamanoMax\_Cadena2 Inicio\_C1 Inicio\_C2\n"**);**  printf**(**"Tamano e inicio escalados: 1:100\n"**);**  **}**  **if(**argc **>=** 3 **&&** argc **<=**7**)**  **{**  printf**(**"%s comparado con %s\n"**,**argv**[**1**],**argv**[**2**]);**  printf**(**"Tamanos: %d %d\n"**,**T1**\***100**,**T2**\***100**);**  printf**(**"Puntos de inicio: %d %d\n"**,**I1**\***100**,**I2**\***100**);**  struct timespec t1**,**t2**,**t3**,**t4**;**  double total**;**  //t1 = time(0);  clock\_gettime**(**CLOCK\_REALTIME**,** **&**t1**);**  char**\*** string1**=**CargarFichero**(**argv**[**1**],**T1**,**I1**);**  char**\*** string2**=**CargarFichero**(**argv**[**2**],**T2**,**I2**);**  // printf("Fin construccion cadenas\n");  struct Celda **\*\***Matriz**;**  **if(**strlen**(**string1**)==**0 **||** strlen**(**string2**)==**0**)**  **{**  printf**(**"Una cadena esta vacia"**);**  exit**(**2**);**  **}**  // printf("Lectura correcta \n");  Matriz**=**inicializarMatriz**(**strlen**(**string1**),**strlen**(**string2**));**  //t2 = time(0);  clock\_gettime**(**CLOCK\_REALTIME**,** **&**t2**);**  // printf("Matriz iniciada\n");  CompletarMatriz**(**string1**,**string2**,**Matriz**);**  //t3 = time(0);  clock\_gettime**(**CLOCK\_REALTIME**,** **&**t3**);**  // printf("Matriz completa\n");  int resultado**=** GetRuta**(**Matriz**,**strlen**(**string1**),**strlen**(**string2**));**  // printf("Ruta calculada\n");  //t4 = time(0);      clock\_gettime**(**CLOCK\_REALTIME**,** **&**t4**);**    total **=** **(** t2**.**tv\_sec **-** t1**.**tv\_sec **)\***1000  **+** **(** t2**.**tv\_nsec **-** t1**.**tv\_nsec **)** **/** **(**float**)(**1000000**);**  printf**(**"Inicializado: %lf\n"**,** total **);**    total **=** **(**t3**.**tv\_sec **-** t2**.**tv\_sec**)\***1000  **+** **(**t3**.**tv\_nsec **-** t2**.**tv\_nsec**)** **/** **(**float**)(**1000000**);**  printf**(**"Creacion de matriz: %lf\n"**,** total **);**    total **=** **(**t4**.**tv\_sec **-** t3**.**tv\_sec**)\***1000  **+** **(**t4**.**tv\_nsec **-** t3**.**tv\_nsec**)** **/** **(**float**)(**1000000**);**  printf**(**"Backtracking: %lf\n"**,** total **)** **;**    total **=** **(**t4**.**tv\_sec **-** t1**.**tv\_sec**)\***1000  **+** **(**t4**.**tv\_nsec **-** t1**.**tv\_nsec**)** **/** **(**float**)(**1000000**);**  printf**(**"Total: %lf\n"**,** total **);**  printf**(**"Coincidencia(porc): %d\n"**,** 100**\***resultado**/**maxU**(**strlen**(**string1**),**strlen**(**string2**)));**  **}**  **else**  **{**  exit**(**3**);**  **}**  printf**(**"Fin"**);**  exit**(**0**);**  **return** 0**;**  **}**  /\*\*  \* CargarFichero funcion que guarda en un string el contenido de un fichero .fasta  \* @author Lidia y Nacho  \* @date 8/2/2018  \* @param NombreFichero nombre del fichero, incluida extension y ruta relativa  \* @out string con el contenido en mayusculas  \*/  char**\*** CargarFichero**(**char**\*** NombreFichero**,**unsigned tamano**,**unsigned inicio**)**  **{**  tamano**\*=**100**;**  inicio**\*=**100**;**  FILE **\***archivo**;**  unsigned i**;**  char caracteres**[**100**];**  char **\***cadena**=**malloc**(**tamano**);** //<--Origen error  strcpy **(**cadena**,** ""**);**  archivo **=** fopen**(**NombreFichero**,**"r"**);**  **if** **(**archivo **==** **NULL)**  **{**  printf**(**"%s no existe"**,**NombreFichero**);**  exit**(**1**);**  **}**  **else**  **{**  fgets**(**caracteres**,**100**,**archivo**);** //Primera linea  **for(**i**=**0**;**i**<**inicio**;**i**++)**  fgets**(**caracteres**,**100**,**archivo**);**  i**=**tamano**;**  **while** **(**feof**(**archivo**)** **==** 0 **&&** strlen**(**cadena**)<**i**)** //Hasta fin de archivo o memoria  **{**  fgets**(**caracteres**,**100**,**archivo**);**  strcat**(**cadena**,** caracteres**);**  **}**  **}**  Mayus**(**cadena**);**  **return** cadena**;**  **}**  // inicializarMatriz funcion que crea la matriz de tamano r c e inicializa la primera fila y columna con valores negativos descendentes.  // @author Sara  // @date 12/2/2018  // @param unsigned r rows  // @param unsigned c cols  // @return arr matriz con los valores negativos  struct Celda**\*\*** inicializarMatriz**(**unsigned r**,** unsigned c**)**  **{**  unsigned i**;**  struct Celda **\*\***arr **=(**struct Celda **\*\*)**malloc**(**r**\***c**\*** **sizeof(**struct Celda**));**  **for** **(**i **=** 0**;** i **<=** r**;** **++**i**)**  arr**[**i**]** **=** **(**struct Celda **\*)**malloc**(**c **\*** **sizeof(**struct Celda**));**  //Casos base posicion: r = 0, c = 0  arr**[**0**][**0**].**score **=** 0**;**  arr**[**0**][**0**].**dir **=** 0**;**    **for(**i **=** 1 **;** i**<=**r**;** i**++)**  **{**  arr**[**i**][**0**].**score **=** **-**i**;**  arr**[**i**][**0**].**dir**=**0**;**  **}**    **for(**i **=** 1 **;** i**<=**c**;** i**++)**  **{**  arr**[**0**][**i**].**score **=** **-**i**;**  arr**[**0**][**i**].**dir**=**0**;**  **}**  **return** arr**;**  **}**  /\*\*  \* CompletarMatriz funcion que calcula el algoritmo Needleman-Wunsch para una matriz  \* @author Nacho  \* @date 7/2/2018  \* @param string1 Cadena de texto 1  \* @param string2 Cadena de texto 2  \* @param matrix Matriz de Celdas, su tamano debe ser el de las cadenas de texto +1  \*/  void CompletarMatriz**(**char**\*** string1**,**char**\*** string2**,**struct Celda**\*\*** matrix**)**  **{**    unsigned i**;**  unsigned j**;**  unsigned size1**=**strlen**(**string1**);**  unsigned size2**=**strlen**(**string2**);**  **for(**i**=**1**;**i**<=**size1**;++**i**)**  **for(**j**=**1**;**j**<=**size2**;++**j**)**  //El argumento de calcular casilla es cierto si ambos strings coinciden o uno de ellos es N  //Recordar que el tamano de la matriz es 1 mayor que los strings, y estos se alinean con el final.  **{**  CalcularCasilla**(**i**,** j**,** **(**string1**[**i**-**1**]==**string2**[**j**-**1**]||**string1**[**i**-**1**]==**'N'**||**string2**[**j**-**1**]==**'N'**),** matrix**);**  **}**  **return;**  **}**  /\*\*  \* CalcularCasilla funcion que calcula el contenido de una casillo de la matriz para el algoritmo Needleman-Wunsch  \* @author Nacho  \* @date 6/2/2018  \* @param i Indice de fila (No puede ser 0)  \* @param j Indice de columna (No puede ser 0)  \* @param igual Comparativa (Char==Char)  \* @param matrix Matriz de structs sobre la que se opera. In/Out  \*/  void CalcularCasilla**(**unsigned i**,** unsigned j**,** bool igual**,** struct Celda **\*\***matrix**)**  **{**  // Constantes Match 2, -2 dismatch, -1 Hueco  int A **=** matrix**[**i**-**1**][**j**].**score **-** 1**;**  int B **=** matrix**[**i**][**j**-**1**].**score **-** 1**;**  int C **=** matrix**[**i**-**1**][**j**-**1**].**score**+** **(**igual**\***4**)-**2**;** //C= arg + (argB\*(Match-Fallo))+Fallo    //Calculo de la dirección como un array de booleanos  int D**=**0**;**  D **+=** **(**A**>=**B **&&** A**>=**C**)<<**1**;** //Vertical en 1a posicion  D **+=** **(**B**>=**A **&&** B**>=**C**);** //Horizontal en 2a posicion  D **+=** **(**C**>=**B **&&** C**>=**A**)<<**2**;** //Diagonal en 3a posicion  matrix**[**i**][**j**].**dir **=** D**;**    matrix**[**i**][**j**].**score**=**maxI**(**maxI**(**A**,**B**),**C**);**    **}**  /\*\*  \* GetRuta Funcion de backtraking para saber cual es el mayor indice de coincidencias. Aumenta la cuenta si encuentra diagonales.  \* @author Paul  \* @author Nacho en la optimización  \* @date 12/2/2018  \* @date 14/2/2018 en optimizacion  \* @param matrix Matriz de structs sobre la que se opera. In/Out  \* @param i Indice de fila inicial (Debe ser la ultima)  \* @param j Indice de columna inicial (Debe ser la ultima)  \*/  unsigned GetRuta**(**struct Celda**\*\*** matrix**,** unsigned i**,** unsigned j**)**  **{**  unsigned maximo **=** 0**;**  //Reinicializado de los scores, ahora mediremos la distancia a la esquina 0,0  //El valor -1 representa dato desconocido  unsigned x**,**y**;**  **for(**x**=**0**;**x**<=**i**;**x**++)**  **for(**y**=**0**;**y**<=**j**;**y**++)**  matrix**[**x**][**y**].**score**=-**1**;**    AuxGetRuta**(**matrix**,** i**,** j**,** 0**,** **&**maximo**);**    **return** maximo**;**  **}**  int AuxGetRuta**(**struct Celda**\*\*** matrix**,** unsigned i**,** unsigned j**,** int cont**,** unsigned **\***maximo**)**  **{**    int A**=-**1**,**B**=-**1**,**C**=-**1**;**  //Poda, impide repetición de celdas si ya ha calculado el camino  **if(**matrix**[**i**][**j**].**score**>=**0**)**  **{**  **return** matrix**[**i**][**j**].**score**;**  **}**  //Poda, impide recorrer un nuevo camino si no hay posibilidad de superar la mejor marca  **if((**cont**+**i**<\***maximo **||** cont**+**j**<\***maximo**)&&(\***maximo**>**0**))**  **{**  matrix**[**i**][**j**].**score**=**0**;**  **return** 0**;**  **}**  //Caso base  **if(**i **==** 0 **||** j **==** 0**)**  **{**  **\***maximo **=** maxI**(**cont**,** **\***maximo**);**  matrix**[**i**][**j**].**score**=**0**;;**  **return** 0**;**  **}**  **else**  **{**  **if(**matrix**[**i**][**j**].**dir**>**3**)**  **{**  //En las diagonales se añade distancia si existen  A**=**AuxGetRuta**(**matrix**,** i **-** 1**,** j **-** 1**,** cont **+** 1**,** maximo**)+**1**;**  **}**  **if(**matrix**[**i**][**j**].**dir**==**2**||**matrix**[**i**][**j**].**dir**==**3**||**matrix**[**i**][**j**].**dir**==**6**||**matrix**[**i**][**j**].**dir**==**7**)**  **{**  B**=**AuxGetRuta**(**matrix**,** i **-** 1**,** j**,** cont**,** maximo**);**  **}**  **if(**matrix**[**i**][**j**].**dir**==**1**||**matrix**[**i**][**j**].**dir**==**3**||**matrix**[**i**][**j**].**dir**==**5**||**matrix**[**i**][**j**].**dir**==**7**)**  **{**  C**=**AuxGetRuta**(**matrix**,** i**,** j **-** 1**,** cont**,** maximo**);**  **}**  **}**    matrix**[**i**][**j**].**score**=**maxI**(**maxI**(**A**,**B**),**C**);**  **return** matrix**[**i**][**j**].**score**;**  **}** |

## **Anexo:** Tiempo dedicado

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Tiempo Total  (Horas) | Días | Personas | Trabajo diario medio  (Minutos) |
| Estudio del algoritmo | 4 | 2 | 2 | 60 |
| Implementación de código | 20 | 7 | 4 | 43 |
| Redacción de documentación | 10 | 5 | 3 | 40 |
| Total: | 34 | 14 | Variable | 48 |

## Anexo: Autoevaluación

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| Proyecto: | Similitud de cadenas de ADN mediante el algoritmo de alineamiento NW | |  |  |
|  |  | |  |  |
| Autores: | Ignacio Gomis Lli | |  |  |
|  | Lidia Montero Egidos | |  |  |
|  | Sara Monzó Bravo | |  |  |
|  | Paul Vargas Hurtado | |  |  |
|  |  | |  |  |
|  | Concepto evaluado | | Valor |  |
|  | Completitud: Todos los apartados incluidos, incluido tiempo dedicado (0-1) | | 1 |  |
|  | Claridad de redacción: redacción, ortografía, organización (0-1) | | 0.9 |  |
|  | Objetivo y utilidad del algoritmo (0-1) | | 0.8 |  |
|  | Descripción del algoritmo: Descripción y en lenguaje matemático (0-1) | | 1 |  |
|  | Estructura y tipo de datos, tamaño de memoria usado por el algoritmo (0-1) | | 0.8 |  |
|  | Análisis del coste del algoritmo: Operaciones y accesos a memoria (0-1) | | 0.9 |  |
|  | Análisis de dependencia de datos, se indica claramente que datos dependen de otros (0-1) | | 1 |  |
|  | Propuestas de paralelización: Descripción de la distribución/organización de los datos y tareas que se pueden realizar en paralelo (0-1) | | 0.9 |  |
|  | Referencias al final del documento (Al menos 5, 3 fuentes/revistas contrastadas y referenciadas en el texto con estilo Harvard) (0-1) | | 0.4 |  |
|  | Formato del documento (Tipo de letra, PDF o DOC sin comprimir) | | 0.9 |  |
|  | **TOTAL (Sobre 10)** | | 8.6 |  |
|  |  | |  |  |
| En caso de plagio o contenido sin referenciar, no se aceptará el documento para su evaluación | | | |  |
|  |  | |  |  |
|  | ¿PLAGIO? |  |  |  |
|  |  |  |  |  |

# Práctica 0

## 1-Tamaños de memoria

En primer lugar, los tamaños han sido seleccionados teniendo en cuenta los tamaños de la memoria del computador. Los hemos calculado mediante una tabla en Excel, de la cual incluimos un fragmento, ya que para obtener los valores hemos creado toda la secuencia:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Argumento | Num. Celdas | Bytes | MB | GB |  |  |
| 1 | 10000 | 80000 | 0.07629395 | 7.4506E-05 | <- | Contenido en un núcleo de nivel L2 |
| 2 | 40000 | 320000 | 0.30517578 | 0.00029802 |  |  |
| 3 | 90000 | 720000 | 0.68664551 | 0.00067055 |  |  |
| 4 | 160000 | 1280000 | 1.22070313 | 0.00119209 |  |  |
| 9 | 810000 | 6480000 | 6.17980957 | 0.00603497 |  |  |
| 14 | 1960000 | 15680000 | 14.9536133 | 0.01460314 | <- | Contenido en un núcleo de nivel L3 |
| 23 | 5290000 | 42320000 | 40.3594971 | 0.03941357 |  |  |
| 32 | 10240000 | 81920000 | 78.125 | 0.07629395 |  |  |
| 124 | 153760000 | 1230080000 | 1173.0957 | 1.14560127 |  |  |
| 900 | 8100000000 | 6.48E+10 | 61798.0957 | 60.3497028 |  |  |
| 926 | 8574760000 | 6.8598E+10 | 65420.2271 | 63.8869405 |  | Tope de memoria principal |
| 927 | 8593290000 | 6.8746E+10 | 65561.5997 | 64.0249997 |  |  |

Entonces, los datos escogidos para estudiar fueron los siguientes:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Argumento | Num. Celdas | Bytes | MB | GB | Tipo de memoria |
| 1 | 10000 | 80000 | 0.076293945 | 7.45E-05 | Cache L2 |
| 4 | 160000 | 1280000 | 1.220703125 | 0.00119 | Cache L3 |
| 9 | 810000 | 6480000 | 6.17980957 | 0.00603 | Cache L3 |
| 14 | 1960000 | 15680000 | 14.95361328 | 0.0146 | Cache L3 |
| 23 | 5290000 | 42320000 | 40.35949707 | 0.0394 | Principal |
| 51 | 26010000 | 208080000 | 198.4405518 | 0.194 | Principal |
| 124 | 153760000 | 1230080000 | 1173.095703 | 1.15 | Principal |
| 888 | 7885440000 | 63083520000 | 60161.13281 | 58.8 | Principal |

## 2-Tiempos sin optimizar

Hemos realizado las pruebas secuenciales, tanto sin optimizar como optimizando con la opción O2. Hay que señalar que estos datos únicamente corresponden al apartado de cálculo de la matriz, ya que es la parte que estamos estudiando del algoritmo. Los resultados están presentados a continuación, los cuales son la media de 5 ejecuciones, mostrando como gráficas los valores menores y como tablas aquellos resultados que dificultan la visión de los datos en las gráficas junto a algunos de los resultados mostrados para realizar la comparativa:

Podemos observar que mientras tiene una tendencia cuadratica, como se esperaba.

Referente a los MOPS podemos ver que tienen una tengencia creciente, pero con dos puntos en los que se denotan perdidas: Al superar el tamaño de la cache, donde se puede ver la penalización para tamaño 1400, que pese a que la matriz cabe en la cache completamente, hay que tener en cuenta adicionalmente un par de strings, lo cual causa que hayan cambios de bloques en la memoria de cache. La segunda perdida es en el caso 5100, en el cual no conseguimos explicar esa perdida, ya que es un valor intermedio entre 2300 y 12400, ambos con el mismo nivel de memoria, y esperaríamos que fuera un valor intermedio entre 489 y 522.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Talla | 2300 | 5100 | 12400 | 88800 |
| Tiempo | 205275.2 | 1073641 | 5595340.6 | 269330870.2 |
| Op. | 100510000 | 494190000 | 2921440000 | 1.49823E+11 |
| MOPS | 489.64 | 460.29 | 522.12 | 556.28 |

## 3-Tiempos optimizados

Mostramos a continuación los datos obtenidos con la opción de compilación –O2:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Talla | 2300 | 5100 | 12400 | 88800 |
| Tiempo | 68609.6 | 436604.6 | 2151090.4 | 109843345 |
| Op. | 100510000 | 494190000 | 2921440000 | 1.49823E+11 |
| MOPS | 1464.96 | 1131.89 | 1358.12 | 1363.97 |

Se puede observar que el comportamiento es muy similar al caso sin compilar, salvo por el detalle de que los MOPS son mayores con tamaños pequeños en comparación al trabajar con memoria principal, es por ello, que adicionalmente hemos obtenido el speed-up de la optimización con respecto a su tiempo sin optimizar:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Talla | 2300 | 5100 | 12400 | 88800 |
| Speed-Up | 2.99 | 2.45 | 2.60 | 2.46 |

Como podemos observar, la optimización resulta más positiva cuando el tamaño del problema es menor. No se puede desestimar el speed up de 2.5 de los casos mayores, donde en tiempo se aprecia mucho mejor, pero esto explica la diferencia en las gráficas de MOPS entre la versión sin optimizar y la versión optimizada.

## 4-Código

El código ha sido mantenido todo a excepción de las funciones para obtener el tiempo. El main por tanto ha quedado así:

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <stdbool.h>  #include <string.h>  #include <time.h>  /\*Struct con el valor de la celda y las direcciones  Si las direcciones son verdadero, entonces ha heredado de esa direccion  \*/  struct Celda  **{**  int score**;**  short dir**;** //Array de booleanos  **};**  char**\*** CargarFichero**(**char**\*,**unsigned**,**unsigned**);**  struct Celda**\*\*** inicializarMatriz**(**unsigned**,** unsigned**);**  void CompletarMatriz**(**char**\*,**char**\*,**struct Celda**\*\*);**  void CalcularCasilla**(**unsigned**,** unsigned**,** bool**,** struct Celda**\*\*);**  unsigned GetRuta**(**struct Celda**\*\*,**unsigned**,**unsigned**);**  int AuxGetRuta**(**struct Celda**\*\*,** unsigned**,** unsigned**,** int**,** unsigned**\*);**  /\*Maximo entre dos valores\*/  unsigned maxU**(**unsigned arg1**,** unsigned arg2**)**  **{**  **if(**arg1**>**arg2**){**  **return** arg1**;**  **}**  **else**  **return** arg2**;**  **}**  int maxI**(**int arg1**,** int arg2**)**  **{**  **if(**arg1**>**arg2**){**  **return** arg1**;**  **}**  **else**  **return** arg2**;**  **}**  /\*Funcion para pasar a mayuscula una cadena. Obtenida de:  https://stackoverflow.com/questions/35181913/converting-char-to-uppercase-in-c  \*/  void Mayus**(**char **\*** temp**)** **{**  char **\*** name**;**  name **=** strtok**(**temp**,**":"**);**  char **\***s **=** name**;**  **while** **(\***s**)** **{**  **\***s **=** toupper**((**unsigned char**)** **\***s**);**  s**++;**  **}}**  /\*\*  \* Main Lleva control del orden de ejecucion de las funciones del algoritmo y la entrada de parametros  \* @author Lidia y Nacho  \* @date 8/2/2018  \* @param Nombre o ruta de Fichero1 (Necesario)  \* @param Nombre o ruta de Fichero2 (Necesario)  \* @param Tamano de cadena de fichero1 (Opcional o ambos ficheros si falta siguiente parametro)  \* @param Tamano de cadena de Fichero2 (Opcional)  \* @param Inicio de cadena de Fichero1 (Opcional o ambos ficheros si falta siguiente parametro)  \* @param Inicio de cadena de Fichero2 (Opcional)  \*/  int main**(** int argc**,** char **\***argv**[]** **)**  **{**  unsigned T1**,**T2**,**I1**,**I2**;**  **switch(**argc**)**  **{**  **case** 3**:** //Ambos desde el principio hasta lo que coja  T1**=**100**;**T2**=**100**;**I1**=**0**;**I2**=**0**;**    **break;**  **case** 4**:** //Ambos con Tamaño arg[3]  T1**=**atoi**(**argv**[**3**]);**T2**=**T1**;**I1**=**0**;**I2**=**0**;**  **break;**  **case** 5**:** //Cada uno con Tamaño arg[3] y arg[4]  T1**=**atoi**(**argv**[**3**]);**T2**=**atoi**(**argv**[**4**]);**I1**=**0**;**I2**=**0**;**  **break;**  **case** 6**:** //Cada uno con Tamaño arg[3] y arg[4] empezando desde arg[5]  T1**=**atoi**(**argv**[**3**]);**T2**=**atoi**(**argv**[**4**]);**I1**=**atoi**(**argv**[**5**]);**I2**=**I1**;**  **break;**  **case** 7**:** //Cada uno con Tamaño arg[3] y arg[4] empezando desde arg[5] y arg[6]  T1**=**atoi**(**argv**[**3**]);**T2**=**atoi**(**argv**[**4**]);**I1**=**atoi**(**argv**[**5**]);**I2**=**atoi**(**argv**[**6**]);**  **break;**  **default:** //Instrucciones de uso  printf**(**"Error en introduccion de datos:\n Se pueden introducir entre 2 y 6 argumentos:\n2 argumentos:\n Fichero\_1 Fichero\_2\n"**);**  printf**(**"3 argumentos:\n Fichero\_1 Fichero\_2 Tamano\_maximo\n"**);**  printf**(**"4 argumentos:\n Fichero\_1 Fichero\_2 TamanoMax\_Cadena1 TamanoMax\_Cadena2\n"**);**  printf**(**"5 argumentos:\n Fichero\_1 Fichero\_2 TamanoMax\_Cadena1 TamanoMax\_Cadena2 Inicio\_Cadenas\n"**);**  printf**(**"6 argumentos:\n Fichero\_1 Fichero\_2 TamanoMax\_Cadena1 TamanoMax\_Cadena2 Inicio\_C1 Inicio\_C2\n"**);**  printf**(**"Tamano e inicio escalados: 1:100\n"**);**  **}**  **if(**argc **>=** 3 **&&** argc **<=**7**)**  **{**  printf**(**"%s comparado con %s\n"**,**argv**[**1**],**argv**[**2**]);**  printf**(**"Tamanos: %d %d\n"**,**T1**\***100**,**T2**\***100**);**  printf**(**"Puntos de inicio: %d %d\n"**,**I1**\***100**,**I2**\***100**);**  struct timeval t1**,**t2**,**t3**,**t4**;**  double total**;**  gettimeofday**(&**t1**,** **NULL);**  char**\*** string1**=**CargarFichero**(**argv**[**1**],**T1**,**I1**);**  char**\*** string2**=**CargarFichero**(**argv**[**2**],**T2**,**I2**);**  //printf("Fin construccion cadenas\n");  struct Celda **\*\***Matriz**;**  **if(**strlen**(**string1**)==**0 **||** strlen**(**string2**)==**0**)**  **{**  printf**(**"Una cadena esta vacia"**);**  exit**(**2**);**  **}**  //printf("Lectura correcta \n");  Matriz**=**inicializarMatriz**(**strlen**(**string1**),**strlen**(**string2**));**  gettimeofday**(&**t2**,** **NULL);**  //printf("Matriz iniciada\n");  CompletarMatriz**(**string1**,**string2**,**Matriz**);**  gettimeofday**(&**t3**,** **NULL);**  //printf("Matriz completa\n");  int resultado**=** GetRuta**(**Matriz**,**strlen**(**string1**),**strlen**(**string2**));**  //printf("Ruta calculada\n");    gettimeofday**(&**t4**,** **NULL);**    total **=** **((**t2**.**tv\_sec **\*** 1000000 **+** t2**.**tv\_usec**)-(**t1**.**tv\_sec **\*** 1000000 **+** t1**.**tv\_usec**));**  printf**(**"Inicializado: %lf\n"**,** total **);**    total **=** **((**t3**.**tv\_sec **\*** 1000000 **+** t3**.**tv\_usec**)-(**t2**.**tv\_sec **\*** 1000000 **+** t2**.**tv\_usec**));**  printf**(**"Creacion de matriz: %lf\n"**,** total **);**    total **=** **((**t4**.**tv\_sec **\*** 1000000 **+** t4**.**tv\_usec**)-(**t3**.**tv\_sec **\*** 1000000 **+** t3**.**tv\_usec**));**  printf**(**"Backtracking: %lf\n"**,** total **)** **;**    total **=** **((**t4**.**tv\_sec **\*** 1000000 **+** t4**.**tv\_usec**)-(**t1**.**tv\_sec **\*** 1000000 **+** t1**.**tv\_usec**));**  printf**(**"Total: %lf\n"**,** total **);**  printf**(**"Coincidencia(porc): %d\n"**,** 100**\***resultado**/**maxU**(**strlen**(**string1**),**strlen**(**string2**)));**  **}**  **else**  **{**  exit**(**3**);**  **}**  printf**(**"Fin"**);**  exit**(**0**);**  **return** 0**;**  **}**  /\*\*  \* CargarFichero funcion que guarda en un string el contenido de un fichero .fasta  \* @author Lidia y Nacho  \* @date 8/2/2018  \* @param NombreFichero nombre del fichero, incluida extension y ruta relativa  \* @out string con el contenido en mayusculas  \*/  char**\*** CargarFichero**(**char**\*** NombreFichero**,**unsigned tamano**,**unsigned inicio**)**  **{**  tamano**\*=**100**;**  inicio**\*=**100**;**  FILE **\***archivo**;**  unsigned i**;**  char caracteres**[**100**];**  char **\***cadena**=**malloc**(**tamano**);** //<--Origen error  strcpy **(**cadena**,** ""**);**  archivo **=** fopen**(**NombreFichero**,**"r"**);**  **if** **(**archivo **==** **NULL)**  **{**  printf**(**"%s no existe"**,**NombreFichero**);**  exit**(**1**);**  **}**  **else**  **{**  fgets**(**caracteres**,**100**,**archivo**);** //Primera linea  **for(**i**=**0**;**i**<**inicio**;**i**++)**  fgets**(**caracteres**,**100**,**archivo**);**  i**=**tamano**;**  **while** **(**feof**(**archivo**)** **==** 0 **&&** strlen**(**cadena**)<**i**)** //Hasta fin de archivo o memoria  **{**  fgets**(**caracteres**,**100**,**archivo**);**  strcat**(**cadena**,** caracteres**);**  **}**  **}**  Mayus**(**cadena**);**  **return** cadena**;**  **}**  // inicializarMatriz funcion que crea la matriz de tamano r c e inicializa la primera fila y columna con valores negativos descendentes.  // @author Sara  // @date 12/2/2018  // @param unsigned r rows  // @param unsigned c cols  // @return arr matriz con los valores negativos  struct Celda**\*\*** inicializarMatriz**(**unsigned r**,** unsigned c**)**  **{**  unsigned i**;**  struct Celda **\*\***arr **=(**struct Celda **\*\*)**malloc**(**r**\***c**\*** **sizeof(**struct Celda**));**  **for** **(**i **=** 0**;** i **<=** r**;** **++**i**)**  arr**[**i**]** **=** **(**struct Celda **\*)**malloc**(**c **\*** **sizeof(**struct Celda**));**  //Casos base posicion: r = 0, c = 0  arr**[**0**][**0**].**score **=** 0**;**  arr**[**0**][**0**].**dir **=** 0**;**    **for(**i **=** 1 **;** i**<=**r**;** i**++)**  **{**  arr**[**i**][**0**].**score **=** **-**i**;**  arr**[**i**][**0**].**dir**=**0**;**  **}**    **for(**i **=** 1 **;** i**<=**c**;** i**++)**  **{**  arr**[**0**][**i**].**score **=** **-**i**;**  arr**[**0**][**i**].**dir**=**0**;**  **}**  **return** arr**;**  **}**  /\*\*  \* CompletarMatriz funcion que calcula el algoritmo Needleman-Wunsch para una matriz  \* @author Nacho  \* @date 7/2/2018  \* @param string1 Cadena de texto 1  \* @param string2 Cadena de texto 2  \* @param matrix Matriz de Celdas, su tamano debe ser el de las cadenas de texto +1  \*/  void CompletarMatriz**(**char**\*** string1**,**char**\*** string2**,**struct Celda**\*\*** matrix**)**  **{**    unsigned i**;**  unsigned j**;**  unsigned size1**=**strlen**(**string1**);**  unsigned size2**=**strlen**(**string2**);**  **for(**i**=**1**;**i**<=**size1**;++**i**)**  **for(**j**=**1**;**j**<=**size2**;++**j**)**  //El argumento de calcular casilla es cierto si ambos strings coinciden o uno de ellos es N  //Recordar que el tamano de la matriz es 1 mayor que los strings, y estos se alinean con el final.  **{**  CalcularCasilla**(**i**,** j**,** **(**string1**[**i**-**1**]==**string2**[**j**-**1**]||**string1**[**i**-**1**]==**'N'**||**string2**[**j**-**1**]==**'N'**),** matrix**);**  **}**  **return;**  **}**  /\*\*  \* CalcularCasilla funcion que calcula el contenido de una casillo de la matriz para el algoritmo Needleman-Wunsch  \* @author Nacho  \* @date 6/2/2018  \* @param i Indice de fila (No puede ser 0)  \* @param j Indice de columna (No puede ser 0)  \* @param igual Comparativa (Char==Char)  \* @param matrix Matriz de structs sobre la que se opera. In/Out  \*/  void CalcularCasilla**(**unsigned i**,** unsigned j**,** bool igual**,** struct Celda **\*\***matrix**)**  **{**  // Constantes Match 2, -2 dismatch, -1 Hueco  int A **=** matrix**[**i**-**1**][**j**].**score **-** 1**;**  int B **=** matrix**[**i**][**j**-**1**].**score **-** 1**;**  int C **=** matrix**[**i**-**1**][**j**-**1**].**score**+** **(**igual**\***4**)-**2**;** //C= arg + (argB\*(Match-Fallo))+Fallo    //Calculo de la dirección como un array de booleanos  int D**=**0**;**  D **+=** **(**A**>=**B **&&** A**>=**C**)<<**1**;** //Vertical en 1a posicion  D **+=** **(**B**>=**A **&&** B**>=**C**);** //Horizontal en 2a posicion  D **+=** **(**C**>=**B **&&** C**>=**A**)<<**2**;** //Diagonal en 3a posicion  matrix**[**i**][**j**].**dir **=** D**;**    matrix**[**i**][**j**].**score**=**maxI**(**maxI**(**A**,**B**),**C**);**    **}**  /\*\*  \* GetRuta Funcion de backtraking para saber cual es el mayor indice de coincidencias. Aumenta la cuenta si encuentra diagonales.  \* @author Paul  \* @author Nacho en la optimización  \* @date 12/2/2018  \* @date 14/2/2018 en optimizacion  \* @param matrix Matriz de structs sobre la que se opera. In/Out  \* @param i Indice de fila inicial (Debe ser la ultima)  \* @param j Indice de columna inicial (Debe ser la ultima)  \*/  unsigned GetRuta**(**struct Celda**\*\*** matrix**,** unsigned i**,** unsigned j**)**  **{**  unsigned maximo **=** 0**;**  //Reinicializado de los scores, ahora mediremos la distancia a la esquina 0,0  //El valor -1 representa dato desconocido  unsigned x**,**y**;**  **for(**x**=**0**;**x**<=**i**;**x**++)**  **for(**y**=**0**;**y**<=**j**;**y**++)**  matrix**[**x**][**y**].**score**=-**1**;**    AuxGetRuta**(**matrix**,** i**,** j**,** 0**,** **&**maximo**);**    **return** maximo**;**  **}**  int AuxGetRuta**(**struct Celda**\*\*** matrix**,** unsigned i**,** unsigned j**,** int cont**,** unsigned **\***maximo**)**  **{**    int A**=-**1**,**B**=-**1**,**C**=-**1**;**  //Poda, impide repetición de celdas si ya ha calculado el camino  **if(**matrix**[**i**][**j**].**score**>=**0**)**  **{**  **return** matrix**[**i**][**j**].**score**;**  **}**  //Poda, impide recorrer un nuevo camino si no hay posibilidad de superar la mejor marca  **if((**cont**+**i**<\***maximo **||** cont**+**j**<\***maximo**)&&(\***maximo**>**0**))**  **{**  matrix**[**i**][**j**].**score**=**0**;**  **return** 0**;**  **}**  //Caso base  **if(**i **==** 0 **||** j **==** 0**)**  **{**  **\***maximo **=** maxI**(**cont**,** **\***maximo**);**  matrix**[**i**][**j**].**score**=**0**;;**  **return** 0**;**  **}**  **else**  **{**  **if(**matrix**[**i**][**j**].**dir**>**3**)**  **{**  //En las diagonales se añade distancia si existen  A**=**AuxGetRuta**(**matrix**,** i **-** 1**,** j **-** 1**,** cont **+** 1**,** maximo**)+**1**;**  **}**  **if(**matrix**[**i**][**j**].**dir**==**2**||**matrix**[**i**][**j**].**dir**==**3**||**matrix**[**i**][**j**].**dir**==**6**||**matrix**[**i**][**j**].**dir**==**7**)**  **{**  B**=**AuxGetRuta**(**matrix**,** i **-** 1**,** j**,** cont**,** maximo**);**  **}**  **if(**matrix**[**i**][**j**].**dir**==**1**||**matrix**[**i**][**j**].**dir**==**3**||**matrix**[**i**][**j**].**dir**==**5**||**matrix**[**i**][**j**].**dir**==**7**)**  **{**  C**=**AuxGetRuta**(**matrix**,** i**,** j **-** 1**,** cont**,** maximo**);**  **}**  **}**    matrix**[**i**][**j**].**score**=**maxI**(**maxI**(**A**,**B**),**C**);**  **return** matrix**[**i**][**j**].**score**;**  **}** |

## 5-Tiempo de trabajo

Tiempo de trabajo en el laboratorio: 2 horas, 3 personas.

Tiempo de elaboración de documentación: 2 horas, 4 personas.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Tiempo Total  (Horas) | Días | Personas | Trabajo diario medio  (Minutos) |
| Trabajo de laboratorio | 2 | 1 | 3 | 120 |
| Realización de documentación | 2 | 1 | 4 | 120 |
| 1ª corrección | 2 | 1 | 1 | 120 |
| Total: | 4 | 2 | Variable | 180 |

## Anexo: Autoevaluación

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Proyecto: | Similitud de cadenas de ADN mediante el algoritmo de alineamiento NW | |  |  |
|  |  | |  |  |
| Autores: | Ignacio Gomis Lli | |  |  |
|  | Lidia Montero Egidos | |  |  |
|  | Sara Monzó Bravo | |  |  |
|  | Paul Vargas Hurtado | |  |  |
|  |  | |  |  |
|  | Concepto evaluado | | Valor |  |
|  | Completitud de estudio experimental (0-1) | | 0.9 |  |
|  | Claridad de redacción y presentación de los datos. (0-1) | | 0.9 |  |
|  | Código del algoritmo secuencial definitivo. (0-2) | | 1.8 |  |
|  | Representación de los datos mediante tablas y/o gráficas. (0-2) | | 1 |  |
|  | Completitud de las medidas utilizadas en el estudio experimental. (0-1) | | 0.8 |  |
|  | Estudio experimental de tiempo de ejecución y velocidad. (0-2) | | 1.8 |  |
|  | Justificación de los datos experimentales obtenidos en función del algoritmo y la arquitectura del ordenador. (0-1) | | 1.5 |  |
|  |  |
|  |  |
|  |  |
|  | **TOTAL (Sobre 10)** | | 8.7 |  |
|  |  | |  |  |
| En caso de plagio o contenido sin referenciar, no se aceptará el documento para su evaluación | | | |  |
|  |  | |  |  |
|  | ¿PLAGIO? |  |  |  |
|  |  |  |  |  |

# CP2: Propuesta de implementación con OpenMP

## 1-Descripción del algoritmo paralelo

Al analizar el algoritmo, observamos que de los paradigmas de programación paralela explicados en clase y detallados en la bibliografía recomendada: Rauber y Rünger (2010) el más recomendable sería el paralelismo de datos, ya que trabajamos sobre una matriz de gran tamaño y la misma tarea con la misma carga computacional en cada sección.

Por otro lado, en la sección de la búsqueda de la mejor coincidencia podría emplearse el paradigma de divide y vencerás, pero el fragmento recursivo, además de que no es el objetivo de estudio de este proyecto, tiene un fuerte componente secuencial para impedir repeticiones de caminos, por la tanto hemos considerado que paralelizarlo solo resultaría negativo para el resultado.

Finalmente, para la configuración inicial de la matriz, hemos decidido aplicar el paradigma del maestro-esclavo, ya que lanzaremos un hilo independiente para cada fichero de entrada, tanto para leerlo como para inicializar las fila y columna 0 de la matriz, con la expectativa de que reducir a la mitad el tiempo de inicializado del programa. Este último caso es anecdótico, ya que es lo que menos coste temporal tiene en el algoritmo, pero consideramos que merecía la pena para recortar tiempos en llamadas con tamaños de matriz muy grandes.

Dadas las dependencias del algoritmo descritas durante el [CP1](#_4-Análisis_de_dependencia) podemos observar dos posibles soluciones, paralelizar antidiagonales, o paralelizar bloques asíncronamente, es decir, cada uno esperando a que los datos para avanzar estén disponibles.

Nos decidimos por la segunda opción porque paralelizar las antidiagonales, para garantizar resultados correctos requeriría muchos cambios de bloques de memoria o una solución asíncrona similar a la de los bloques.

Desde un principio nuestro código fue desarrollado con la intención de poder repartir el trabajo en múltiples bloques, así que para esta entrega, lo más destacable a variar del algortimo será la función anteriormente llamada CompletarMatriz, la cual únicamente organizaba el orden de cálculo de CalcularCasilla. Por tanto aquí reemplazaremos CompletarMatriz por un método que emplea OpenMP y un sistema de locks para ordenar las llamadas a CalcularCasilla.

También modificaremos el Main para poder llamar a la función con el algoritmo secuencial o el algoritmo con paralelización mediante OpenMP, dejándolo preparado para ampliarlo en un futuro para MPI.

Para clarificar el método que estamos proponiendo, adjuntamos el siguiente diagrama:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Locks: [4, 2, 2, 0] | | | | | | | | | | | | | | | |
| 0 | | | | 1 | | | | 2 | | | | 3 | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Tenemos en la matriz repartida en 4 bloques, delimitados por las líneas discontinuas y habiendo calculado ya las casillas coloreadas. Se puede observar que hay adjunto un vector que indica el número de filas completadas por cada bloque. Cuando un hilo completa una fila incrementa en uno su indicador, y luego comprueba si puede comenzar la siguiente iteración, comprobando que sus filas terminadas sean menores que las del bloque anterior, que en caso contrario, simplemente dormirá esperando el momento indicado. En este caso, el hilo 2 estaría esperando a que el hilo 1 completase su fila.

Claramente, el principal problema de esto son los tiempos de espera de los procesadores ya que el último hilo no podrá operar hasta que todos los anteriores hayan completado una fila, y de la misma manera el primer hilo quedará a la espera hasta que todos los posteriores finalicen. En la siguiente imagen podemos ver en azul, en el caso ideal, las celdas que estarían completas antes de la puesta en marcha del hilo 3 y en rojo las celdas que faltarían por hacer al finalizar el hilo 0, haciendo esto del orden de 24 unidades de tiempo que no tenemos máximo paralelismo, en este ejemplo.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | | | | 1 | | | | 2 | | | | 3 | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Sin embargo, la manera de minimizar este efecto sería el asignar múltiples bloques a un mismo hilo, ordenados de la manera indicada. La imagen adjunta, con la misma leyenda que la anterior, muestra que el tiempo de paro de procesadores resulta menor, 12 unidades de tiempo.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | | 1 | | 2 | | 3 | | 0 | | 1 | | 2 | | 3 | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Este trenzado reducirá los tiempos de inactividad, pero incrementará el número de dependencias de temporales en el programa, así que tendremos que encontrar el valor que mejor se adapte. Hay que señalar, que si el trenzado llega a tamaños muy pequeños, la solución acaba siendo la misma que trabajar por diagonales.

Entonces, la directiva de OMP que emplearemos, será #pragma parallel, acompañada de de get\_thread\_num() y get\_max\_threads(), ajustando la cantidad de los hilos mediante la variable de entorno. La idea es tener el máximo control posible del problema, asignando nosotros a los hilos indicados la carga de trabajo deseada y el momento indicado de trabajar.

Por otro lado, la lectura y actualización de los datos de la matriz es controlada por un lock, para evitar problemas de concurrencia.

## 2-Implementación del algoritmo

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <stdbool.h>  #include <string.h>  #include <time.h>  #include <omp.h>  /\*Struct con el valor de la celda y las direcciones  Si las direcciones son verdadero, entonces ha heredado de esa direccion  \*/  struct Celda  **{**  int score**;**  char dir**;** //Array de booleanos  **};**  void ImprimirInstruccionesDeUso**();**  char**\*** CargarFichero**(**char**\*,**unsigned**,**unsigned**);**  struct Celda**\*\*** inicializarMatriz**(**unsigned**,** unsigned**,**unsigned**);**  void CompletarMatrizSecuencial**(**char**\*,**char**\*,**struct Celda**\*\*);**  void CompletarMatrizOmp**(**char**\*,**char**\*,**struct Celda**\*\*,**unsigned**);**  unsigned**\*** AsignarVector**(**unsigned**,**unsigned**);**  void CalcularSubMatriz**(**struct Celda**\*\*,**unsigned**,**unsigned**,**char**\*,**char**\*,**unsigned**\*,**unsigned**);**  void CalcularCasilla**(**unsigned**,** unsigned**,** bool**,** struct Celda**\*\*);**  unsigned GetRuta**(**struct Celda**\*\*,**unsigned**,**unsigned**);**  int AuxGetRuta**(**struct Celda**\*\*,** unsigned**,** unsigned**,** int**,** unsigned**\*);**  //Funciones auxiliares  /\*Maximo entre dos valores\*/  unsigned maxU**(**unsigned arg1**,** unsigned arg2**)**  **{**  **if(**arg1**>**arg2**){**  **return** arg1**;**  **}**  **else**  **return** arg2**;**  **}**  int maxI**(**int arg1**,** int arg2**)**  **{**  **if(**arg1**>**arg2**){**  **return** arg1**;**  **}**  **else**  **return** arg2**;**  **}**  /\*Funcion para pasar a mayuscula una cadena. Obtenida de:  https://stackoverflow.com/questions/35181913/converting-char-to-uppercase-in-c  \*/  void Mayus**(**char **\*** temp**)** **{**  char **\*** name**;**  name **=** strtok**(**temp**,**":"**);**  char **\***s **=** name**;**  **while** **(\***s**)** **{**  **\***s **=** toupper**((**unsigned char**)** **\***s**);**  s**++;**  **}}**  /\*\*  \* CompletarMatriz funcion que gestiona los hilos para realizar el algoritmo Needleman-Wunsch en multiples procesadores  \* @author Nacho  \* @date 28/2/2018  \* @param string1 Cadena de texto 1  \* @param string2 Cadena de texto 2  \* @param matrix Matriz de Celdas, su tamano debe ser el de las cadenas de texto +1  \* @param sobrecarga Cantidad de bloques que realiza un hilo. A mayor cantidad menores tiempos, pero su valor maximo deberï¿½a ser la longitud del string.  \*/  void CompletarMatrizOmp**(**char**\*** string1**,**char**\*** string2**,**struct Celda**\*\*** matrix**,** unsigned sobrecarga**)**  **{**  unsigned i**;**  unsigned p**=**omp\_get\_max\_threads**();**  unsigned **\***posiciones**=**AsignarVector**(**strlen**(**string2**),**p**\***sobrecarga**);**  unsigned **\***locks **=(**unsigned **\*)**malloc**(**p**\***sobrecarga**\*** **sizeof(**unsigned**));**  **for** **(**i **=** 0**;** i **<** p**\***sobrecarga**;** **++**i**)**  **{**  locks**[**i**]** **=** 0**;**  **}**    #pragma omp parallel private(i) shared(matrix,posiciones,string1,string2,locks,p,sobrecarga)  **{**  unsigned id**=**omp\_get\_thread\_num**();**  **for(**i**=**0**;**i**<**sobrecarga**;**i**++)**  **if(**id**==**0**&&**i**==**0**)**  CalcularSubMatriz **(**matrix**,** 1**,**posiciones**[**0**],**string1**,**string2**,**locks**,**0**);**  **else**  CalcularSubMatriz **(**matrix**,** posiciones**[**id**-**1**+**p**\***i**],**posiciones**[**id**+**p**\***i**],**string1**,**string2**,**locks**,(**id**+**p**\***i**));**  **}**  **}**  /\*\*  \* AsignarVector funcion que calcula las posiciones finales de cada bloque  \* @author Lidia  \* @date 4/3/2018  \* @param tamano Tamano del ancho de la matriz  \* @param p Numero de procesadores  \* @return final Vector dinamico de posiciones  \*/  unsigned**\*** AsignarVector**(**unsigned tamano**,**unsigned p**)**  **{**  unsigned a**,**l**,** i**;**  unsigned **\***final **=(**unsigned **\*)**malloc**(**p**\*sizeof(**unsigned**));**  l**=**tamano**/**p**;**  //Garantizar un minimo avance  **if(**l**==**0**)**  l**=**1**;**  //Asignacion  a**=**l**;**  **for(**i**=**0**;**i**<**p**-**1**;++**i**)**  **{**  final**[**i**]=**a**;**  a**+=**l**;**  **}**  final**[**p**-**1**]=**tamano**;**  //Para evitar accidentes  **for(**i**=**0**;**i**<**p**;++**i**)**  **{**  **if(**final**[**i**]>**tamano**)**  final**[**i**]=**tamano**;**  **}**  **return** final**;**  **}**  /\*\*  \* CalcularSubMatriz rellena la matriz a partes, entre dos columnas objetivo  \* @author Paul  \* @date 01/03/2018  \* @param matrix Matriz sobre la que se opera  \* @param c1 indice de la columna que calcularemos  \* @param c2 indice de la columna que calcularemos  \* @param string1 Cadena de texto  \* @param string2 Cadena de texto  \* @param locks Array de cerrojos  \* @param id Identificador del hilo que esta ejecutando esta funcion  \*/  void CalcularSubMatriz**(**struct Celda**\*\*** matrix**,** unsigned c1**,** unsigned c2**,** char**\*** string1**,** char**\*** string2**,** unsigned**\*** locks**,** unsigned id**)**  **{**  int tiempo **=** 500**;**  unsigned size1**=**strlen**(**string1**);**  unsigned i**;**  unsigned j**;**    **for(** i **=** 1**;** i **<=** size1**;** **++**i**)**  **{**  **for(** j **=** c1**;** j **<=** c2**;** **++**j**)**  **{**  **while(** id **>** 0 **&&** locks**[**id **-** 1**]** **<=** locks**[**id**]** **)**  **{**  usleep**(**tiempo**);**  **}**    CalcularCasilla**(**i**,** j**,** string1**[**i **-** 1**]** **==** string2**[**j**-**1**]** **||** string1**[**i **-** 1**]** **==** 'N' **||** string2**[**j**-**1**]** **==** 'N'**,** matrix**);**  **}**  locks**[**id**]++;**  **}**      **return;**  **}**      /\*\*  \* Main Lleva control del orden de ejecucion de las funciones del algoritmo y la entrada de parametros  \* @author Lidia y Nacho  \* @date 8/2/2018  \* @date 3/3/2018 añadido OMP  \* @param Nombre o ruta de Fichero1 (Necesario)  \* @param Nombre o ruta de Fichero2 (Necesario)  \* @param Tamano de cadena de fichero1 (Opcional o ambos ficheros si falta siguiente parametro)  \* @param Tamano de cadena de Fichero2 (Opcional)  \* @param Inicio de cadena de Fichero1 (Opcional o ambos ficheros si falta siguiente parametro)  \* @param Inicio de cadena de Fichero2 (Opcional)  \*/  int main**(** int argc**,** char **\***argv**[]** **)**  **{**  unsigned T1**,**T2**,**I1**,**I2**,** sobrecarga**,** modo**;**  char**\*** nombre1**;**  char**\*** nombre2**;**  bool lecturaCorrecta**=**1**;**  **switch(**atoi**(**argv**[**1**]))**  **{**  **case** 2**:** //OpenMP  **{**  modo**=**2**;**  sobrecarga**=**atoi**(**argv**[**2**]);**  nombre1**=**argv**[**3**];**  nombre2**=**argv**[**4**];**  **switch(**argc**)**  **{**  **case** 5**:** //Ambos desde el principio hasta lo que coja  T1**=**1000**;**T2**=**1000**;**I1**=**0**;**I2**=**0**;**  **break;**  **case** 6**:** //Ambos con Tamano arg[5]  T1**=**atoi**(**argv**[**5**]);**T2**=**T1**;**I1**=**0**;**I2**=**0**;**  **break;**  **case** 7**:** //Cada uno con Tamano arg[5] y arg[6]  T1**=**atoi**(**argv**[**5**]);**T2**=**atoi**(**argv**[**6**]);**I1**=**0**;**I2**=**0**;**  **break;**  **case** 8**:** //Cada uno con Tamano arg[5] y arg[6] empezando desde arg[7]  T1**=**atoi**(**argv**[**5**]);**T2**=**atoi**(**argv**[**6**]);**I1**=**atoi**(**argv**[**7**]);**I2**=**I1**;**  **break;**  **case** 9**:** //Cada uno con Tamano arg[5] y arg[6] empezando desde arg[7] y arg[8]  T1**=**atoi**(**argv**[**5**]);**T2**=**atoi**(**argv**[**6**]);**I1**=**atoi**(**argv**[**7**]);**I2**=**atoi**(**argv**[**8**]);**  **break;**  **default:** //Instrucciones de uso  ImprimirInstruccionesDeUso**();**  lecturaCorrecta**=**0**;**  **break;**  **}**  **}**  **break;**  **case** 3**:** //MPI  **{**    **}**  **break;**  **case** 1**:**  **{**  modo**=**1**;**  nombre1**=**argv**[**2**];**  nombre2**=**argv**[**3**];**  **switch(**argc**)**  **{**  **case** 4**:** //Ambos desde el principio hasta lo que coja  T1**=**1000**;**T2**=**1000**;**I1**=**0**;**I2**=**0**;**  **break;**  **case** 5**:** //Ambos con Tamano arg[3]  T1**=**atoi**(**argv**[**4**]);**T2**=**T1**;**I1**=**0**;**I2**=**0**;**  **break;**  **case** 6**:** //Cada uno con Tamano arg[3] y arg[4]  T1**=**atoi**(**argv**[**4**]);**T2**=**atoi**(**argv**[**5**]);**I1**=**0**;**I2**=**0**;**  **break;**  **case** 7**:** //Cada uno con Tamano arg[3] y arg[4] empezando desde arg[5]  T1**=**atoi**(**argv**[**4**]);**T2**=**atoi**(**argv**[**5**]);**I1**=**atoi**(**argv**[**6**]);**I2**=**I1**;**  **break;**  **case** 8**:** //Cada uno con Tamano arg[3] y arg[4] empezando desde arg[5] y arg[6]  T1**=**atoi**(**argv**[**4**]);**T2**=**atoi**(**argv**[**5**]);**I1**=**atoi**(**argv**[**6**]);**I2**=**atoi**(**argv**[**7**]);**  **break;**  **default:** //Instrucciones de uso  ImprimirInstruccionesDeUso**();**  lecturaCorrecta**=**0**;**  **}**  **}**  **break;**  **default:**  **{**  lecturaCorrecta**=**0**;**  ImprimirInstruccionesDeUso**();**  **}**  **break;**  **}**    **if(**lecturaCorrecta**==**1**)**  **{**  printf**(**"%s comparado con %s\n"**,**nombre1**,**nombre2**);**  printf**(**"Tamanos: %d %d\n"**,**T1**\***100**,**T2**\***100**);**  printf**(**"Puntos de inicio: %d %d\n"**,**I1**\***100**,**I2**\***100**);**  struct timeval t1**,**t2**,**t3**,**t4**;**  double total**;**  gettimeofday**(&**t1**,** **NULL);**    char**\*** string1**;**  char**\*** string2**;**  **switch(**modo**)**  **{**  **case** 1**:** //Secuencial  **{**  string1**=**CargarFichero**(**nombre1**,**T1**,**I1**);**  string2**=**CargarFichero**(**nombre2**,**T2**,**I2**);**  **}**  **break;**  **case** 2**:** //omp  **{**  #pragma omp parallel sections shared(nombre1,nombre2,T1,I1,T2,I2)  **{**  #pragma omp section  string1**=**CargarFichero**(**nombre1**,**T1**,**I1**);**  #pragma omp section  string2**=**CargarFichero**(**nombre2**,**T2**,**I2**);**  **}**  **}**  **break;**    **default:**  **{**  exit**(**4**);**  **}**  **break;**    **}**    struct Celda **\*\***Matriz**;**  **if(**strlen**(**string1**)==**0 **||** strlen**(**string2**)==**0**)**  **{**  printf**(**"Una cadena esta vacia"**);**  exit**(**2**);**  **}**  Matriz**=**inicializarMatriz**(**strlen**(**string1**),**strlen**(**string2**),**modo**);**  gettimeofday**(&**t2**,** **NULL);**  **if(**modo**==**1**)**  CompletarMatrizSecuencial**(**string1**,**string2**,**Matriz**);**  **if(**modo**==**2**)**  CompletarMatrizOmp**(**string1**,**string2**,**Matriz**,**sobrecarga**);**  gettimeofday**(&**t3**,** **NULL);**  int resultado**=** GetRuta**(**Matriz**,**strlen**(**string1**),**strlen**(**string2**));**    gettimeofday**(&**t4**,** **NULL);**    total **=** **((**t2**.**tv\_sec **\*** 1000000 **+** t2**.**tv\_usec**)-(**t1**.**tv\_sec **\*** 1000000 **+** t1**.**tv\_usec**));**  printf**(**"Inicializado: %lf\n"**,** total **);**    total **=** **((**t3**.**tv\_sec **\*** 1000000 **+** t3**.**tv\_usec**)-(**t2**.**tv\_sec **\*** 1000000 **+** t2**.**tv\_usec**));**  printf**(**"Creacion de matriz: %lf\n"**,** total **);**    total **=** **((**t4**.**tv\_sec **\*** 1000000 **+** t4**.**tv\_usec**)-(**t3**.**tv\_sec **\*** 1000000 **+** t3**.**tv\_usec**));**  printf**(**"Backtracking: %lf\n"**,** total **)** **;**    total **=** **((**t4**.**tv\_sec **\*** 1000000 **+** t4**.**tv\_usec**)-(**t1**.**tv\_sec **\*** 1000000 **+** t1**.**tv\_usec**));**  printf**(**"Total: %lf\n"**,** total **);**  printf**(**"Coincidencia(porc): %d\n"**,** 100**\***resultado**/**maxU**(**strlen**(**string1**),**strlen**(**string2**)));**  **}**  **else**  **{**  exit**(**3**);**  **}**  printf**(**"Fin"**);**  exit**(**0**);**  **return** 0**;**  **}**  /\*\*  \* ImprimirInstruccionesDeUso imprime la informacion de uso de este programa, asi como los autores.  \* @author Nacho  \* @date 3/03/2018  \*/  void ImprimirInstruccionesDeUso**()**  **{**  printf**(**"Programa para el calculo de porcentaje de similitud de dos cadenas de Adn\n"**);**  printf**(**"Autores:\n"**);**  printf**(**"\tIgnacio Gomis Lli\n"**);**  printf**(**"\tLidia Montero Egidos\n"**);**  printf**(**"\tSara Monzo Bravo\n"**);**  printf**(**"\tPaul Vargas Hurtado\n"**);**  printf**(**"Asignatura: Arquitectura de procesadores, 3ro Ingenieria Informatica\n"**);**  printf**(**"Profesorado:\n"**);**  printf**(**"\t Jose Manuel Claver Iborra\n"**);**  printf**(**"\t Adria Gimenez Pastor\n"**);**  printf**(**"Universidad de Valencia, ETSE\n"**);**  printf**(**"Curso 2017/2018\n"**);**  printf**(**"\n"**);**  printf**(**"Instrucciones de uso:\n"**);**  printf**(**"-Los ficheros deben estar escritos en formato fasta\n"**);**  printf**(**"-Los argumentos de tamano tienen un escalado 1:100, es decir, en caso de que\n"**);**  printf**(**"se introduzca un 1 nos referimos a la posicion 100 del fichero\n"**);**  printf**(**"\n"**);**  printf**(**"Existen distintos modos de ejecucion, se determina cual se emplea segun el\n"**);**  printf**(**"primer argumento\n"**);**  printf**(**"-Ejecucion secuencial, primer argumento 1\n"**);**  printf**(**"3 argumentos:\n 1 Bloques\_Por\_Hilo Fichero\_1 Fichero\_2\n"**);**  printf**(**"4 argumentos:\n 1 Fichero\_1 Fichero\_2 Tamano\_maximo\n"**);**  printf**(**"5 argumentos:\n 1 Fichero\_1 Fichero\_2 TamanoMax\_Cadena1 TamanoMax\_Cadena2\n"**);**  printf**(**"6 argumentos:\n 1 Fichero\_1 Fichero\_2 TamanoMax\_Cadena1 TamanoMax\_Cadena2 Inicio\_Cadenas\n"**);**  printf**(**"7 argumentos:\n 1 Fichero\_1 Fichero\_2 TamanoMax\_Cadena1 TamanoMax\_Cadena2 Inicio\_C1 Inicio\_C2\n"**);**  printf**(**"\n"**);**  printf**(**"-Ejecucion con OMP, segundo argumento 2, requiere opcion de compilado -fopenmp\n"**);**  printf**(**"4 argumentos:\n 2 Bloques\_Por\_Hilo Fichero\_1 Fichero\_2\n"**);**  printf**(**"5 argumentos:\n 2 Bloques\_Por\_Hilo Fichero\_1 Fichero\_2 Tamano\_maximo\n"**);**  printf**(**"6 argumentos:\n 2 Bloques\_Por\_Hilo Fichero\_1 Fichero\_2 TamanoMax\_Cadena1 TamanoMax\_Cadena2\n"**);**  printf**(**"7 argumentos:\n 2 Bloques\_Por\_Hilo Fichero\_1 Fichero\_2 TamanoMax\_Cadena1 TamanoMax\_Cadena2 Inicio\_Cadenas\n"**);**  printf**(**"8 argumentos:\n 2 Bloques\_Por\_Hilo Fichero\_1 Fichero\_2 TamanoMax\_Cadena1 TamanoMax\_Cadena2 Inicio\_C1 Inicio\_C2\n"**);**  **}**  /\*\*  \* CargarFichero funcion que guarda en un string el contenido de un fichero .fasta  \* @author Lidia y Nacho  \* @date 8/2/2018  \* @param NombreFichero nombre del fichero, incluida extension y ruta relativa  \* @out string con el contenido en mayusculas  \*/  char**\*** CargarFichero**(**char**\*** NombreFichero**,**unsigned tamano**,**unsigned inicio**)**  **{**  tamano**\*=**100**;**  inicio**\*=**100**;**  FILE **\***archivo**;**  unsigned i**;**  char caracteres**[**100**];**  char **\***cadena**=**malloc**(**tamano**);** //<--Origen error  strcpy **(**cadena**,** ""**);**  archivo **=** fopen**(**NombreFichero**,**"r"**);**  **if** **(**archivo **==** **NULL)**  **{**  printf**(**"%s no existe"**,**NombreFichero**);**  exit**(**1**);**  **}**  **else**  **{**  fgets**(**caracteres**,**100**,**archivo**);** //Primera linea  **for(**i**=**0**;**i**<**inicio**;**i**++)**  fgets**(**caracteres**,**100**,**archivo**);**  i**=**tamano**;**  **while** **(**feof**(**archivo**)** **==** 0 **&&** strlen**(**cadena**)<**i**)** //Hasta fin de archivo o memoria  **{**  fgets**(**caracteres**,**100**,**archivo**);**  strcat**(**cadena**,** caracteres**);**  **}**  **}**  Mayus**(**cadena**);**  **return** cadena**;**  **}**  // inicializarMatriz funcion que crea la matriz de tamano r c e inicializa la primera fila y columna con valores negativos descendentes.  // @author Sara  // @date 12/2/2018  // @param unsigned r rows  // @param unsigned c cols  // @return arr matriz con los valores negativos  struct Celda**\*\*** inicializarMatriz**(**unsigned r**,** unsigned c**,** unsigned m**)**  **{**  unsigned i**;**  struct Celda **\*\***arr **=(**struct Celda **\*\*)**malloc**(**r**\***c**\*** **sizeof(**struct Celda**));**  **for** **(**i **=** 0**;** i **<=** r**;** **++**i**)**  arr**[**i**]** **=** **(**struct Celda **\*)**malloc**(**c **\*** **sizeof(**struct Celda**));**  //Casos base posicion: r = 0, c = 0  arr**[**0**][**0**].**score **=** 0**;**  arr**[**0**][**0**].**dir **=** 0**;**  **switch(**m**)**  **{**  **case** 1**:** //Secuencial  **{**  **for(**i **=** 1 **;** i**<=**r**;** i**++)**  **{**  arr**[**i**][**0**].**score **=** **-**i**;**  arr**[**i**][**0**].**dir**=**0**;**  **}**  **for(**i **=** 1 **;** i**<=**c**;** i**++)**  **{**  arr**[**0**][**i**].**score **=** **-**i**;**  arr**[**0**][**i**].**dir**=**0**;**  **}**  **}**  **case** 2**:** //Omp  **{**  #pragma omp parallel sections shared(arr) private (i)  **{**  #pragma omp section  **{**  **for(**i **=** 1 **;** i**<=**r**;** i**++)**  **{**  arr**[**i**][**0**].**score **=** **-**i**;**  arr**[**i**][**0**].**dir**=**0**;**  **}**  **}**  #pragma omp section  **{**  **for(**i **=** 1 **;** i**<=**c**;** i**++)**  **{**  arr**[**0**][**i**].**score **=** **-**i**;**  arr**[**0**][**i**].**dir**=**0**;**  **}**  **}**  **}**  **}**  **break;**  **default:**  **{**  exit**(**4**);**  **}**  **break;**  **}**    **return** arr**;**  **}**  /\*\*  \* CompletarMatriz funcion que calcula el algoritmo Needleman-Wunsch para una matriz  \* @author Nacho  \* @date 7/2/2018  \* @param string1 Cadena de texto 1  \* @param string2 Cadena de texto 2  \* @param matrix Matriz de Celdas, su tamano debe ser el de las cadenas de texto +1  \*/  void CompletarMatrizSecuencial**(**char**\*** string1**,**char**\*** string2**,**struct Celda**\*\*** matrix**)**  **{**    unsigned i**;**  unsigned j**;**  unsigned size1**=**strlen**(**string1**);**  unsigned size2**=**strlen**(**string2**);**  **for(**i**=**1**;**i**<=**size1**;++**i**)**  **for(**j**=**1**;**j**<=**size2**;++**j**)**  //El argumento de calcular casilla es cierto si ambos strings coinciden o uno de ellos es N  //Recordar que el tamano de la matriz es 1 mayor que los strings, y estos se alinean con el final.  **{**  CalcularCasilla**(**i**,** j**,** **(**string1**[**i**-**1**]==**string2**[**j**-**1**]||**string1**[**i**-**1**]==**'N'**||**string2**[**j**-**1**]==**'N'**),** matrix**);**  **}**  **return;**  **}**  /\*\*  \* CalcularCasilla funcion que calcula el contenido de una casillo de la matriz para el algoritmo Needleman-Wunsch  \* @author Nacho  \* @date 6/2/2018  \* @param i Indice de fila (No puede ser 0)  \* @param j Indice de columna (No puede ser 0)  \* @param igual Comparativa (Char==Char)  \* @param matrix Matriz de structs sobre la que se opera. In/Out  \*/  void CalcularCasilla**(**unsigned i**,** unsigned j**,** bool igual**,** struct Celda **\*\***matrix**)**  **{**  // Constantes Match 2, -2 dismatch, -1 Hueco  int A **=** matrix**[**i**-**1**][**j**].**score **-** 1**;**  int B **=** matrix**[**i**][**j**-**1**].**score **-** 1**;**  int C **=** matrix**[**i**-**1**][**j**-**1**].**score**+** **(**igual**\***4**)-**2**;** //C= arg + (argB\*(Match-Fallo))+Fallo    //Calculo de la dirección como un array de booleanos  int D**=**0**;**  D **+=** **(**A**>=**B **&&** A**>=**C**)<<**1**;** //Vertical en 1a posicion  D **+=** **(**B**>=**A **&&** B**>=**C**);** //Horizontal en 2a posicion  D **+=** **(**C**>=**B **&&** C**>=**A**)<<**2**;** //Diagonal en 3a posicion  matrix**[**i**][**j**].**dir **=** D**;**    matrix**[**i**][**j**].**score**=**maxI**(**maxI**(**A**,**B**),**C**);**    **}**  /\*\*  \* GetRuta Funcion de backtraking para saber cual es el mayor indice de coincidencias. Aumenta la cuenta si encuentra diagonales.  \* @author Paul  \* @author Nacho en la optimización  \* @date 12/2/2018  \* @date 14/2/2018 en optimizacion  \* @param matrix Matriz de structs sobre la que se opera. In/Out  \* @param i Indice de fila inicial (Debe ser la ultima)  \* @param j Indice de columna inicial (Debe ser la ultima)  \*/  unsigned GetRuta**(**struct Celda**\*\*** matrix**,** unsigned i**,** unsigned j**)**  **{**  unsigned maximo **=** 0**;**  //Reinicializado de los scores, ahora mediremos la distancia a la esquina 0,0  //El valor -1 representa dato desconocido  unsigned x**,**y**;**  **for(**x**=**0**;**x**<=**i**;**x**++)**  **for(**y**=**0**;**y**<=**j**;**y**++)**  matrix**[**x**][**y**].**score**=-**1**;**    AuxGetRuta**(**matrix**,** i**,** j**,** 0**,** **&**maximo**);**    **return** maximo**;**  **}**  int AuxGetRuta**(**struct Celda**\*\*** matrix**,** unsigned i**,** unsigned j**,** int cont**,** unsigned **\***maximo**)**  **{**    int A**=-**1**,**B**=-**1**,**C**=-**1**;**  //Poda, impide repetición de celdas si ya ha calculado el camino  **if(**matrix**[**i**][**j**].**score**>=**0**)**  **{**  **return** matrix**[**i**][**j**].**score**;**  **}**  //Poda, impide recorrer un nuevo camino si no hay posibilidad de superar la mejor marca  **if((**cont**+**i**<\***maximo **||** cont**+**j**<\***maximo**)&&(\***maximo**>**0**))**  **{**  matrix**[**i**][**j**].**score**=**0**;**  **return** 0**;**  **}**  //Caso base  **if(**i **==** 0 **||** j **==** 0**)**  **{**  **\***maximo **=** maxI**(**cont**,** **\***maximo**);**  matrix**[**i**][**j**].**score**=**0**;;**  **return** 0**;**  **}**  **else**  **{**  **if(**matrix**[**i**][**j**].**dir**>**3**)**  **{**  //En las diagonales se añade distancia si existen  A**=**AuxGetRuta**(**matrix**,** i **-** 1**,** j **-** 1**,** cont **+** 1**,** maximo**)+**1**;**  **}**  **if(**matrix**[**i**][**j**].**dir**==**2**||**matrix**[**i**][**j**].**dir**==**3**||**matrix**[**i**][**j**].**dir**==**6**||**matrix**[**i**][**j**].**dir**==**7**)**  **{**  B**=**AuxGetRuta**(**matrix**,** i **-** 1**,** j**,** cont**,** maximo**);**  **}**  **if(**matrix**[**i**][**j**].**dir**==**1**||**matrix**[**i**][**j**].**dir**==**3**||**matrix**[**i**][**j**].**dir**==**5**||**matrix**[**i**][**j**].**dir**==**7**)**  **{**  C**=**AuxGetRuta**(**matrix**,** i**,** j **-** 1**,** cont**,** maximo**);**  **}**  **}**    matrix**[**i**][**j**].**score**=**maxI**(**maxI**(**A**,**B**),**C**);**  **return** matrix**[**i**][**j**].**score**;**  **}** |

## 3-Analisis del algoritmo

### Coste computacional

Reduciendo la [tabla anterior](#_Coste_computacional_del) a los cambios realizados para operar con OMP, y siendo S el número de veces que un hilo consulta el cerrojo y L el número de bloques en los que se ha partido la matriz.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Función | Lecturas Matriz | Escrituras Matriz | Lecturas Cadenas | Escrituras Cadenas | Operaciones |
| CalcularCasilla | 3 | 2 | 4 | 0 | 19 |
| AsignarVector | 0 | 0 | 0 | L | 3+2L |
| CompletarSubmatriz (Propias) | 0 | 0 | S | N | 2S |
| CompletarMatriz(Total) | 3NM | 2NM | 4NM+LS | L(N+1) | 19NM+2LS+2L+3 |
| Orden Esperado | N\*M | N\*M | N\*M+L\*S | L\*N | N\*M+L\*S |

Como podemos comparar con la tabla anterior, los sobrecostes, la inicialización y el control de los hilos han incrementado las operaciones totales de la función Completar Matriz, aunque la cota superior sigue estando delimitada por N\*M, nos encontramos con L\*S, que aunque L será conocido y de tamaño deseado, S resultará impredecible, y en algún caso podría ser muy elevado, aunque suponemos que el coste vendrá más marcado por las operaciones de calcular casillas.

### Coste de memoria

Tras realizar las pruebas con un único procesador hemos comprobado que el empleo de Int para las celdas es suficiente, sin necesidad de pasar a usar Doubles, por lo tanto el tamaño de la matriz es el mismo que calculamos en el [CP1](#_Almacenamiento_en_memoria:).

La paralelización nos añade al coste de memoria dos vectores de Unsigneds con la cantidad de hilos\*bloquesPorHilo, lo cual podemos considerar despreciable en tamaños menores o incluso del coste de una fila adicional en la matriz, es decir, para un problema de P procesadores con Q bloques por hilo, tendría un peso total de P\*Q\*4 bytes.

### Estimaciones

Debido a la estructura de esta paralelización, asumiendo que los procesadores fuesen igual de rápidos entre sí y la carga estuviera perfectamente distribuida y la matriz tuviera un gran tamaño, nos encontraríamos con que este problema sería escalable, ya que mientras los hilos no tuviesen que detenerse, todos podrían funcionar concurrentemente durante la mayor parte del algoritmo. En este caso los speed-ups podrían tender a P y la eficiencia a 1, ya que el código secuencial es únicamente la configuración de los hilos.

Sin embargo, eso es el caso ideal, en la práctica nos encontraremos con que los sobrecostes pueden resultar muy elevados si no usamos un trenzado lo bastante alto o si por el contrario el trenzado resulta tan minucioso que los hilos se encuentren más tiempo comprobando que pueden avanzar que calculándose, además de que si el ancho del bloque es menor que el de un bloque en la memoria de cache, se podrían generar demasiadas comunicaciones entre las memorias. Es por ello que esperamos que si manteniendo el tamaño y los hilos, variando el entrelazado, el speed-up realizará una curva que alcanzará un máximo en un punto intermedio.

Por el contrario, si mantenemos fijo el entrelazado e hilos y aumentamos el tamaño, los sobrecostes de inicio y final aumentan de manera lineal y finalmente si aumentamos el número de hilos, el sobrecoste por inicio y final también aumenta.

Por tanto, podemos suponer que la diferencia entre los valores ideales teorizados en un inicio será ocasionada únicamente por estos sobrecostes, los cuales podrían comprometer que nuestro algoritmo resulte escalable, así como que obtengamos eficiencias más bajas de lo esperado.

## Anexo: Tiempo trabajado

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Tiempo Total  (Horas) | Días | Personas | Trabajo diario medio  (Minutos) |
| Estudio de la paralelización | 3 | 2 | 1 | 90 |
| Implementación de código | 8 | 4 | 4 | 30 |
| Redacción de documentación | 10 | 3 | 3 | 65 |
| Total: | 21 | 9 | Variable | 140 |

## Anexo: Autoevaluación

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Proyecto: | Similitud de cadenas de ADN mediante el algoritmo de alineamiento NW | |  |  |
|  |  | |  |  |
| Autores: | Ignacio Gomis Lli | |  |  |
|  | Lidia Montero Egidos | |  |  |
|  | Sara Monzó Bravo | |  |  |
|  | Paul Vargas Hurtado | |  |  |
|  |  | |  |  |
|  | Concepto evaluado | | Valor |  |
|  | Completitud (0-1) | | 1 |  |
|  | Claridad de redacción (0-1) | | 1 |  |
|  | Referencia al análisis de dependencias en el CP1 (0-0.5) | | 0.5 |  |
|  | Discusión de paradigmas de programación aplicables al caso. (0-1) | | 0.75 |  |
|  | Análisis del coste computacional del algoritmo paralelo (0-1) | | 0.75 |  |
|  | Propuesta de directivas de OpenMP y organización del código para la paralelización del algoritmo (0-2) | | 2 |  |
|  | Estimación analítica de aceleración, eficiencia y escalabilidad (0-1) | | 0.5 |  |
|  |  |
|  |  |
|  |  |
|  | Nuevas referencias (0-0.5) | | 0.5 |  |
|  | Formato del documento (0-0.5) | | 1 |  |
|  | **TOTAL (Sobre 10)** | | 8 |  |
|  |  | |  |  |
| En caso de plagio o contenido sin referenciar, no se aceptará el documento para su evaluación | | | |  |
|  |  | |  |  |
|  | ¿PLAGIO? |  |  |  |
|  |  |  |  |  |

# Practica 1

En primer lugar, sobre las ejecuciones, se han realizado 5 pruebas para la obtención de cada dato y tomado la media de ellas. Los experimentos menores (Tallas menores o iguales a 12400 ó 12 procesadores o menos) se realizaron en distintos compute-0-\* y los experimentos mayores (Talla 88800 ó 24 procesadores) se realizaron en BOE.

## 1-Tiempos de ejecución y MOPS

En primer lugar hicimos una comparativa de tiempos variando procesadores y tamaño, con los datos en microsegundos, manteniendo el entrelazado de los datos al mínimo. Los resultados fueron los siguientes, tanto para el programa sin optimizar:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Talla vs Proc | 100 | 400 | 900 | 1400 | 2300 | 5100 | 12400 | 88800 |
| 1 procesador | 751 | 7717 | 30120 | 87027 | 205275 | 1073641 | 5595340 | 269330870 |
| 2 procesadores | 1020 | 7097 | 20659 | 42064 | 109596 | 568592 | 2619038 | 147805353 |
| 4 procesadores | 1685 | 4408 | 13898 | 27689 | 69274 | 316593 | 1731952 | 81986493 |
| 8 procesadores | 3224 | 5123 | 11227 | 21301 | 51096 | 227264 | 1306567 | 47755538 |
| 12 procesadores | 4852 | 6449 | 10243 | 18011 | 38705 | 164942 | 908848 | 32661632 |
| 24 procesadores | 9941 | 12211 | 14854 | 22624 | 39763 | 132917 | 614190 | 24178014 |

Como se puede ver, los tiempos de ejecución aproximadamente se dividen a la mitad al doblar los procesadores, aunque para tamaños pequeños, los tiempos con más procesadores es mayor. Esto se debe a que además de los costes de creación de los hilos, debemos contar que nuestra solución es asincrona, por lo que las detenciones internas al rellenar la matriz penalizan demasiado.

A continuación se presentan los tiempos obtenidos con la optimización -O2

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Talla vs Proc | 100 | 400 | 900 | 1400 | 2300 | 5100 | 12400 | 88800 |
| 1 procesador | 185 | 2301 | 9889 | 30167 | 68609 | 436604 | 2151090 | 109843345 |
| 2 procesadores | 357 | 2651 | 8470 | 17623 | 46334 | 215449 | 1171073 | 60393258 |
| 4 procesadores | 1350 | 2322 | 5401 | 11553 | 26162 | 114109 | 607285 | 36358419 |
| 8 procesadores | 2568 | 3433 | 6696 | 9455 | 19980 | 82441 | 483293 | 18838726 |
| 12 procesadores | 3648 | 5607 | 6928 | 10147 | 17543 | 62675 | 328862 | 13113761 |
| 24 procesadores | 8891 | 9897 | 11005 | 16367 | 26957 | 70148 | 268817 | 9648851 |

Para observar la mejora que representa la optimización en esta fase, tenemos la siguiente gráfica, la cual muestra el speed up de la optimización O2 respecto de la versión inicial para cada par de argumentos:

Por otro lado, las MOPS han tenido los siguientes resultados, en los cuales se puede ver, el incremento de las operaciones por segundo es mayor conforme aumentan los procesadores, como cabe esperar.

|  |  |  |  |
| --- | --- | --- | --- |
| Talla (O0) | 5100 | 12400 | 88800 |
| MOPS 1 procesador | 460.2935246 | 522.120137 | 556.2799388 |
| MOPS 2 procesadores | 869.1466397 | 1115.46284 | 1013.653139 |
| MOPS 4 procesadores | 1560.959185 | 1686.7902 | 1827.415139 |
| MOPS 8 procesadores | 2174.515674 | 2235.96613 | 3137.298129 |
| MOPS 12 procesadores | 2996.129567 | 3214.44219 | 4587.13634 |
| MOPS 24 procesadores | 3718.023374 | 4756.5675 | 6196.677539 |

|  |  |  |  |
| --- | --- | --- | --- |
| Talla (O2) | 5100 | 12400 | 88800 |
| MOPS 1 procesador | 1131.893709 | 1358.12051 | 1363.973029 |
| MOPS 2 procesadores | 2293.767899 | 2494.66942 | 2480.796102 |
| MOPS 4 procesadores | 4330.828728 | 4810.65252 | 4120.733565 |
| MOPS 8 procesadores | 5994.439687 | 6044.86305 | 7952.945438 |
| MOPS 12 procesadores | 7884.911784 | 8883.48304 | 11424.89564 |
| MOPS 24 procesadores | 7044.941994 | 10867.7489 | 15527.58529 |

## 2-Estudio de la aceleración y eficiencia

Como muestran las siguientes gráficas, nuestro algoritmo ha llegado a obtener cifras de un speed up de 11, con 24 procesadores, lo cual resulta un dato realmente positivo. Hay que señalar que para tamaños pequeños, una menor cantidad de procesadores es más funcional, pero conforme aumenta el tamaño rápidamente se ven los efectos del trabajo en paralelo.

|  |  |  |
| --- | --- | --- |
| Speed-Up (O0) | 12400 | 88800 |
| 1 procesador | 1 | 1 |
| 2 procesadores | 2.13641029 | 1.8221997 |
| 4 procesadores | 3.23065533 | 3.28506389 |
| 8 procesadores | 4.28247441 | 5.63978298 |
| 12 procesadores | 6.15651833 | 8.24609341 |
| 24 procesadores | 9.11010162 | 11.1394949 |

|  |  |  |
| --- | --- | --- |
| Speed-Up (O2) | 12400 | 88800 |
| 1 procesador | 1 | 1 |
| 2 procesadores | 1.83685424 | 1.81880143 |
| 4 procesadores | 3.54213965 | 3.0211254 |
| 8 procesadores | 4.45090328 | 5.83072045 |
| 12 procesadores | 6.54101234 | 8.37618882 |
| 24 procesadores | 8.00205046 | 11.3840853 |

Por otro lado, al revisar las eficiencias, se puede ver claramente que la eficiencia de los procesadores aumenta con la talla, en algunos casos aislados incluso superando a la ejecución secuencial (recordemos que estos datos se obtienen con una media de 5 ejecuciones)

|  |  |  |
| --- | --- | --- |
| Eficiencia (O0) | 12400 | 88800 |
| 1 procesador | 1 | 1 |
| 2 procesadores | 1.06820515 | 0.91109985 |
| 4 procesadores | 0.80766383 | 0.82126597 |
| 8 procesadores | 0.5353093 | 0.70497287 |
| 12 procesadores | 0.51304319 | 0.68717445 |
| 24 procesadores | 0.37958757 | 0.46414562 |

|  |  |  |
| --- | --- | --- |
| Eficiencia (O2) | 12400 | 88800 |
| 1 procesador | 1 | 1 |
| 2 procesadores | 0.91842712 | 0.90940072 |
| 4 procesadores | 0.88553491 | 0.75528135 |
| 8 procesadores | 0.55636291 | 0.72884006 |
| 12 procesadores | 0.54508436 | 0.69801574 |
| 24 procesadores | 0.33341877 | 0.47433689 |

## 3-Estudio del entrelazado

Además de estos tiempos, tambien probamos a modificar el entrelazado de los datos manteniendo el tamaño en un valor elevado. Debido a la implementación que hemos hecho, no hemos empleado ningun schedule, pero ha sido implementada una versión alternativa, por lo que el parametro no representa el numero de columnas de cada bloque, como haría el schedule(static,n) sino que representa el numero de bloques que debe hacer cada procesador. Es decir, con talla 100, 2 procesadores y entrelazado 2, el primer procesador será responsable de las columnas 0-24 y 50-74. Estos tiempos han sido los siguientes en la versión sin optimizar:

Y los siguientes en la versión optimizada

Como se puede ver a simple vista, conforme aumenta el entrelazado, los tiempos se incrementan, e incluso pueden llegar a ser mucho mayores que una ejecución secuencial, dadas las esperas y multiples cargas de bloques, tal como pronosticamos en el [CP2](#_Estimaciones), pero esperabamos que hubiera un mínimo distinto del caso de 1 bloque por procesador, cosa que no se ha cumplido (solo se ha dado en un unico caso), quedando patente que a mejor opción ha sido, la estudiada desde un principio. Adjuntamos las gráficas de speed-up de ambos casos:

4-Estudio de la escalabilidad

A continuación mostramos las gráficas de la eficiencia según la talla y los procesadores. Se puede observar como los problemas resultan más escalables a medida que la talla del problema aumenta, es decir, que la variación de eficiencia es menor al incrementar el número de procesadores.

No se puede decir que sea un problema totalmente escalable, pero como indicamos en su momento, en un caso extremadamente grande sí que podría resultar escalable.5- Código OMP

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <stdbool.h>  #include <string.h>  #include <time.h>  #include <omp.h>  /\*Struct con el valor de la celda y las direcciones  Si las direcciones son verdadero, entonces ha heredado de esa direccion  \*/  struct Celda  **{**  int score**;**  char dir**;** //Array de booleanos  **};**  void ImprimirInstruccionesDeUso**();**  char**\*** CargarFichero**(**char**\*,**unsigned**,**unsigned**);**  struct Celda**\*\*** inicializarMatriz**(**unsigned**,** unsigned**,**unsigned**);**  void CompletarMatrizSecuencial**(**char**\*,**char**\*,**struct Celda**\*\*);**  void CompletarMatrizOmp**(**char**\*,**char**\*,**struct Celda**\*\*,**unsigned**);**  unsigned**\*** AsignarVector**(**unsigned**,**unsigned**);**  void CalcularSubMatriz**(**struct Celda**\*\*,**unsigned**,**unsigned**,**char**\*,**char**\*,**unsigned**\*,**unsigned**);**  void CalcularCasilla**(**unsigned**,** unsigned**,** bool**,** struct Celda**\*\*);**  unsigned GetRuta**(**struct Celda**\*\*,**unsigned**,**unsigned**);**  int AuxGetRuta**(**struct Celda**\*\*,** unsigned**,** unsigned**,** int**,** unsigned**\*);**  //Funciones auxiliares  /\*Maximo entre dos valores\*/  unsigned maxU**(**unsigned arg1**,** unsigned arg2**)**  **{**  **if(**arg1**>**arg2**){**  **return** arg1**;**  **}**  **else**  **return** arg2**;**  **}**  int maxI**(**int arg1**,** int arg2**)**  **{**  **if(**arg1**>**arg2**){**  **return** arg1**;**  **}**  **else**  **return** arg2**;**  **}**  /\*Funcion para pasar a mayuscula una cadena. Obtenida de:  https://stackoverflow.com/questions/35181913/converting-char-to-uppercase-in-c  \*/  void Mayus**(**char **\*** temp**)** **{**  char **\*** name**;**  name **=** strtok**(**temp**,**":"**);**  char **\***s **=** name**;**  **while** **(\***s**)** **{**  **\***s **=** toupper**((**unsigned char**)** **\***s**);**  s**++;**  **}}**  /\*\*  \* Main Lleva control del orden de ejecucion de las funciones del algoritmo y la entrada de parametros  \* @author Lidia y Nacho  \* @date 8/2/2018  \* @date 3/3/2018 añadido OMP  \* @param Nombre o ruta de Fichero1 (Necesario)  \* @param Nombre o ruta de Fichero2 (Necesario)  \* @param Tamano de cadena de fichero1 (Opcional o ambos ficheros si falta siguiente parametro)  \* @param Tamano de cadena de Fichero2 (Opcional)  \* @param Inicio de cadena de Fichero1 (Opcional o ambos ficheros si falta siguiente parametro)  \* @param Inicio de cadena de Fichero2 (Opcional)  \*/  int main**(** int argc**,** char **\***argv**[]** **)**  **{**  unsigned T1**,**T2**,**I1**,**I2**,** sobrecarga**,** modo**;**  char**\*** nombre1**;**  char**\*** nombre2**;**  bool lecturaCorrecta**=**1**;**  **switch(**atoi**(**argv**[**1**]))**  **{**  **case** 2**:** //OpenMP  **{**  modo**=**2**;**  sobrecarga**=**atoi**(**argv**[**2**]);**  nombre1**=**argv**[**3**];**  nombre2**=**argv**[**4**];**  **switch(**argc**)**  **{**  **case** 5**:** //Ambos desde el principio hasta lo que coja  T1**=**1000**;**T2**=**1000**;**I1**=**0**;**I2**=**0**;**  **break;**  **case** 6**:** //Ambos con Tamano arg[5]  T1**=**atoi**(**argv**[**5**]);**T2**=**T1**;**I1**=**0**;**I2**=**0**;**  **break;**  **case** 7**:** //Cada uno con Tamano arg[5] y arg[6]  T1**=**atoi**(**argv**[**5**]);**T2**=**atoi**(**argv**[**6**]);**I1**=**0**;**I2**=**0**;**  **break;**  **case** 8**:** //Cada uno con Tamano arg[5] y arg[6] empezando desde arg[7]  T1**=**atoi**(**argv**[**5**]);**T2**=**atoi**(**argv**[**6**]);**I1**=**atoi**(**argv**[**7**]);**I2**=**I1**;**  **break;**  **case** 9**:** //Cada uno con Tamano arg[5] y arg[6] empezando desde arg[7] y arg[8]  T1**=**atoi**(**argv**[**5**]);**T2**=**atoi**(**argv**[**6**]);**I1**=**atoi**(**argv**[**7**]);**I2**=**atoi**(**argv**[**8**]);**  **break;**  **default:** //Instrucciones de uso  ImprimirInstruccionesDeUso**();**  lecturaCorrecta**=**0**;**  **break;**  **}**  **}**  **break;**  **case** 3**:** //MPI  **{**    **}**  **break;**  **case** 1**:**  **{**  modo**=**1**;**  nombre1**=**argv**[**2**];**  nombre2**=**argv**[**3**];**  **switch(**argc**)**  **{**  **case** 4**:** //Ambos desde el principio hasta lo que coja  T1**=**1000**;**T2**=**1000**;**I1**=**0**;**I2**=**0**;**  **break;**  **case** 5**:** //Ambos con Tamano arg[3]  T1**=**atoi**(**argv**[**4**]);**T2**=**T1**;**I1**=**0**;**I2**=**0**;**  **break;**  **case** 6**:** //Cada uno con Tamano arg[3] y arg[4]  T1**=**atoi**(**argv**[**4**]);**T2**=**atoi**(**argv**[**5**]);**I1**=**0**;**I2**=**0**;**  **break;**  **case** 7**:** //Cada uno con Tamano arg[3] y arg[4] empezando desde arg[5]  T1**=**atoi**(**argv**[**4**]);**T2**=**atoi**(**argv**[**5**]);**I1**=**atoi**(**argv**[**6**]);**I2**=**I1**;**  **break;**  **case** 8**:** //Cada uno con Tamano arg[3] y arg[4] empezando desde arg[5] y arg[6]  T1**=**atoi**(**argv**[**4**]);**T2**=**atoi**(**argv**[**5**]);**I1**=**atoi**(**argv**[**6**]);**I2**=**atoi**(**argv**[**7**]);**  **break;**  **default:** //Instrucciones de uso  ImprimirInstruccionesDeUso**();**  lecturaCorrecta**=**0**;**  **}**  **}**  **break;**  **default:**  **{**  lecturaCorrecta**=**0**;**  ImprimirInstruccionesDeUso**();**  **}**  **break;**  **}**    **if(**lecturaCorrecta**==**1**)**  **{**  printf**(**"%s comparado con %s\n"**,**nombre1**,**nombre2**);**  printf**(**"Tamanos: %d %d\n"**,**T1**\***100**,**T2**\***100**);**  printf**(**"Puntos de inicio: %d %d\n"**,**I1**\***100**,**I2**\***100**);**  struct timeval t1**,**t2**,**t3**,**t4**;**  double total**;**  gettimeofday**(&**t1**,** **NULL);**    char**\*** string1**;**  char**\*** string2**;**  **switch(**modo**)**  **{**  **case** 2**:** //omp  **{**  #pragma omp parallel sections shared(nombre1,nombre2,T1,I1,T2,I2)  **{**  #pragma omp section  string1**=**CargarFichero**(**nombre1**,**T1**,**I1**);**  #pragma omp section  string2**=**CargarFichero**(**nombre2**,**T2**,**I2**);**  **}**  **}**  **break;**  **default:**  **{**  string1**=**CargarFichero**(**nombre1**,**T1**,**I1**);**  string2**=**CargarFichero**(**nombre2**,**T2**,**I2**);**  **}**  **break;**    **}**    struct Celda **\*\***Matriz**;**  **if(**strlen**(**string1**)==**0 **||** strlen**(**string2**)==**0**)**  **{**  printf**(**"Una cadena esta vacia"**);**  exit**(**2**);**  **}**  Matriz**=**inicializarMatriz**(**strlen**(**string1**),**strlen**(**string2**),**modo**);**  gettimeofday**(&**t2**,** **NULL);**  **if(**modo**==**1**)**  CompletarMatrizSecuencial**(**string1**,**string2**,**Matriz**);**  **if(**modo**==**2**)**  CompletarMatrizOmp**(**string1**,**string2**,**Matriz**,**sobrecarga**);**  gettimeofday**(&**t3**,** **NULL);**  int resultado**=** GetRuta**(**Matriz**,**strlen**(**string1**),**strlen**(**string2**));**    gettimeofday**(&**t4**,** **NULL);**    total **=** **((**t2**.**tv\_sec **\*** 1000000 **+** t2**.**tv\_usec**)-(**t1**.**tv\_sec **\*** 1000000 **+** t1**.**tv\_usec**));**  printf**(**"Inicializado: %lf\n"**,** total **);**    total **=** **((**t3**.**tv\_sec **\*** 1000000 **+** t3**.**tv\_usec**)-(**t2**.**tv\_sec **\*** 1000000 **+** t2**.**tv\_usec**));**  printf**(**"Creacion de matriz: %lf\n"**,** total **);**    total **=** **((**t4**.**tv\_sec **\*** 1000000 **+** t4**.**tv\_usec**)-(**t3**.**tv\_sec **\*** 1000000 **+** t3**.**tv\_usec**));**  printf**(**"Backtracking: %lf\n"**,** total **)** **;**    total **=** **((**t4**.**tv\_sec **\*** 1000000 **+** t4**.**tv\_usec**)-(**t1**.**tv\_sec **\*** 1000000 **+** t1**.**tv\_usec**));**  printf**(**"Total: %lf\n"**,** total **);**  printf**(**"Coincidencia(porc): %d\n"**,** 100**\***resultado**/**maxU**(**strlen**(**string1**),**strlen**(**string2**)));**  **}**  **else**  **{**  exit**(**3**);**  **}**  printf**(**"Fin"**);**  exit**(**0**);**  **return** 0**;**  **}**  /\*\*  \* ImprimirInstruccionesDeUso imprime la informacion de uso de este programa, asi como los autores.  \* @author Nacho  \* @date 3/03/2018  \*/  void ImprimirInstruccionesDeUso**()**  **{**  printf**(**"Programa para el calculo de porcentaje de similitud de dos cadenas de Adn\n"**);**  printf**(**"Autores:\n"**);**  printf**(**"\tIgnacio Gomis Lli\n"**);**  printf**(**"\tLidia Montero Egidos\n"**);**  printf**(**"\tSara Monzo Bravo\n"**);**  printf**(**"\tPaul Vargas Hurtado\n"**);**  printf**(**"Asignatura: Arquitectura de procesadores, 3ro Ingenieria Informatica\n"**);**  printf**(**"Profesorado:\n"**);**  printf**(**"\t Jose Manuel Claver Iborra\n"**);**  printf**(**"\t Adria Gimenez Pastor\n"**);**  printf**(**"Universidad de Valencia, ETSE\n"**);**  printf**(**"Curso 2017/2018\n"**);**  printf**(**"\n"**);**  printf**(**"Instrucciones de uso:\n"**);**  printf**(**"-Los ficheros deben estar escritos en formato fasta\n"**);**  printf**(**"-Los argumentos de tamano tienen un escalado 1:100, es decir, en caso de que\n"**);**  printf**(**"se introduzca un 1 nos referimos a la posicion 100 del fichero\n"**);**  printf**(**"\n"**);**  printf**(**"Existen distintos modos de ejecucion, se determina cual se emplea segun el\n"**);**  printf**(**"primer argumento\n"**);**  printf**(**"-Ejecucion secuencial, primer argumento 1\n"**);**  printf**(**"3 argumentos:\n 1 Bloques\_Por\_Hilo Fichero\_1 Fichero\_2\n"**);**  printf**(**"4 argumentos:\n 1 Fichero\_1 Fichero\_2 Tamano\_maximo\n"**);**  printf**(**"5 argumentos:\n 1 Fichero\_1 Fichero\_2 TamanoMax\_Cadena1 TamanoMax\_Cadena2\n"**);**  printf**(**"6 argumentos:\n 1 Fichero\_1 Fichero\_2 TamanoMax\_Cadena1 TamanoMax\_Cadena2 Inicio\_Cadenas\n"**);**  printf**(**"7 argumentos:\n 1 Fichero\_1 Fichero\_2 TamanoMax\_Cadena1 TamanoMax\_Cadena2 Inicio\_C1 Inicio\_C2\n"**);**  printf**(**"\n"**);**  printf**(**"-Ejecucion con OMP, segundo argumento 2, requiere opcion de compilado -fopenmp\n"**);**  printf**(**"4 argumentos:\n 2 Bloques\_Por\_Hilo Fichero\_1 Fichero\_2\n"**);**  printf**(**"5 argumentos:\n 2 Bloques\_Por\_Hilo Fichero\_1 Fichero\_2 Tamano\_maximo\n"**);**  printf**(**"6 argumentos:\n 2 Bloques\_Por\_Hilo Fichero\_1 Fichero\_2 TamanoMax\_Cadena1 TamanoMax\_Cadena2\n"**);**  printf**(**"7 argumentos:\n 2 Bloques\_Por\_Hilo Fichero\_1 Fichero\_2 TamanoMax\_Cadena1 TamanoMax\_Cadena2 Inicio\_Cadenas\n"**);**  printf**(**"8 argumentos:\n 2 Bloques\_Por\_Hilo Fichero\_1 Fichero\_2 TamanoMax\_Cadena1 TamanoMax\_Cadena2 Inicio\_C1 Inicio\_C2\n"**);**  **}**  /\*\*  \* CargarFichero funcion que guarda en un string el contenido de un fichero .fasta  \* @author Lidia y Nacho  \* @date 8/2/2018  \* @param NombreFichero nombre del fichero, incluida extension y ruta relativa  \* @out string con el contenido en mayusculas  \*/  char**\*** CargarFichero**(**char**\*** NombreFichero**,**unsigned tamano**,**unsigned inicio**)**  **{**  tamano**\*=**100**;**  FILE **\***archivo**;**  unsigned i**;**  char caracteres**[**100**];**  char **\***cadena**=**malloc**(**tamano**);** //<--Origen error  strcpy **(**cadena**,** ""**);**  archivo **=** fopen**(**NombreFichero**,**"r"**);**  **if** **(**archivo **==** **NULL)**  **{**  printf**(**"%s no existe"**,**NombreFichero**);**  exit**(**1**);**  **}**  **else**  **{**  fgets**(**caracteres**,**100**,**archivo**);** //Primera linea  **for(**i**=**0**;(**i**<**inicio**)&&(**feof**(**archivo**)** **==** 0**);**i**++)**  **{**  fgets**(**caracteres**,**100**,**archivo**);**  **}**  i**=**tamano**;**  **while** **(**feof**(**archivo**)** **==** 0**)** //Hasta fin de archivo o memoria  **{**  fgets**(**caracteres**,**100**,**archivo**);**  strcat**(**cadena**,** caracteres**);**  **}**  **}**  Mayus**(**cadena**);**  **return** cadena**;**  **}**  // inicializarMatriz funcion que crea la matriz de tamano r c e inicializa la primera fila y columna con valores negativos descendentes.  // @author Sara  // @date 12/2/2018  // @param unsigned r rows  // @param unsigned c cols  // @param unsigned m modo de ejecucion  // @return arr matriz con los valores negativos  struct Celda**\*\*** inicializarMatriz**(**unsigned r**,** unsigned c**,** unsigned m**)**  **{**  unsigned i**;**  struct Celda **\*\***arr **=(**struct Celda **\*\*)**malloc**((**r**+**1**)\*** **sizeof(**struct Celda **\*));**  struct Celda **\***mem**=** **(**struct Celda **\*)** malloc**((**r**+**1**)\*(**c**+**1**)\*sizeof(**struct Celda**));**  **for** **(**i**=**0**;**i**<=**r**;++**i**)**  arr**[**i**]=** mem **+** i**\*(**c**+**1**);**  //Casos base posicion: r = 0, c = 0  arr**[**0**][**0**].**score **=** 0**;**  arr**[**0**][**0**].**dir **=** 0**;**  **switch(**m**)**  **{**  **case** 2**:** //Omp  **{**  #pragma omp parallel sections shared(arr) private (i)  **{**  #pragma omp section  **{**  **for(**i **=** 1 **;** i**<=**r**;** i**++)**  **{**  arr**[**i**][**0**].**score **=** **-**i**;**  arr**[**i**][**0**].**dir**=**0**;**  **}**  **}**  #pragma omp section  **{**  **for(**i **=** 1 **;** i**<=**c**;** i**++)**  **{**  arr**[**0**][**i**].**score **=** **-**i**;**  arr**[**0**][**i**].**dir**=**0**;**  **}**  **}**  **}**  **}**  **break;**  **default:**  **{**  **for(**i **=** 1 **;** i**<=**r**;** i**++)**  **{**  arr**[**i**][**0**].**score **=** **-**i**;**  arr**[**i**][**0**].**dir**=**0**;**  **}**  **for(**i **=** 1 **;** i**<=**c**;** i**++)**  **{**  arr**[**0**][**i**].**score **=** **-**i**;**  arr**[**0**][**i**].**dir**=**0**;**  **}**  **}**  **break;**  **}**    **return** arr**;**  **}**  /\*\*  \* CompletarMatriz funcion que calcula el algoritmo Needleman-Wunsch para una matriz  \* @author Nacho  \* @date 7/2/2018  \* @param string1 Cadena de texto 1  \* @param string2 Cadena de texto 2  \* @param matrix Matriz de Celdas, su tamano debe ser el de las cadenas de texto +1  \*/  void CompletarMatrizSecuencial**(**char**\*** string1**,**char**\*** string2**,**struct Celda**\*\*** matrix**)**  **{**    unsigned i**;**  unsigned j**;**  unsigned size1**=**strlen**(**string1**);**  unsigned size2**=**strlen**(**string2**);**  **for(**i**=**1**;**i**<=**size1**;++**i**)**  **for(**j**=**1**;**j**<=**size2**;++**j**)**  //El argumento de calcular casilla es cierto si ambos strings coinciden o uno de ellos es N  //Recordar que el tamano de la matriz es 1 mayor que los strings, y estos se alinean con el final.  **{**  CalcularCasilla**(**i**,** j**,** **(**string1**[**i**-**1**]==**string2**[**j**-**1**]||**string1**[**i**-**1**]==**'N'**||**string2**[**j**-**1**]==**'N'**),** matrix**);**  **}**  **return;**  **}**  /\*\*  \* CompletarMatriz funcion que gestiona los hilos para realizar el algoritmo Needleman-Wunsch en multiples procesadores  \* @author Nacho  \* @date 28/2/2018  \* @param string1 Cadena de texto 1  \* @param string2 Cadena de texto 2  \* @param matrix Matriz de Celdas, su tamano debe ser el de las cadenas de texto +1  \* @param sobrecarga Cantidad de bloques que realiza un hilo. A mayor cantidad menores tiempos, pero su valor maximo deberï¿½a ser la longitud del string.  \*/  void CompletarMatrizOmp**(**char**\*** string1**,**char**\*** string2**,**struct Celda**\*\*** matrix**,** unsigned sobrecarga**)**  **{**  unsigned i**;**  unsigned p**=**omp\_get\_max\_threads**();**  unsigned **\***posiciones**=**AsignarVector**(**strlen**(**string2**),**p**\***sobrecarga**);**  unsigned **\***locks **=(**unsigned **\*)**malloc**(**p**\***sobrecarga**\*** **sizeof(**unsigned**));**  **for** **(**i **=** 0**;** i **<** p**\***sobrecarga**;** **++**i**)**  **{**  locks**[**i**]** **=** 0**;**  **}**    #pragma omp parallel private(i) shared(matrix,posiciones,string1,string2,locks,p,sobrecarga)  **{**  unsigned id**=**omp\_get\_thread\_num**();**  **for(**i**=**0**;**i**<**sobrecarga**;**i**++)**  **if(**id**==**0**&&**i**==**0**)**  CalcularSubMatriz **(**matrix**,** 1**,**posiciones**[**0**],**string1**,**string2**,**locks**,**0**);**  **else**  CalcularSubMatriz **(**matrix**,** posiciones**[**id**-**1**+**p**\***i**],**posiciones**[**id**+**p**\***i**],**string1**,**string2**,**locks**,(**id**+**p**\***i**));**  **}**  **}**  /\*\*  \* AsignarVector funcion que calcula las posiciones finales de cada bloque  \* @author Lidia  \* @date 4/3/2018  \* @param tamano Tamano del ancho de la matriz  \* @param p Numero de procesadores  \* @return final Vector dinamico de posiciones  \*/  unsigned**\*** AsignarVector**(**unsigned tamano**,**unsigned p**)**  **{**  unsigned a**,**l**,** i**;**  unsigned **\***final **=(**unsigned **\*)**malloc**(**p**\*sizeof(**unsigned**));**  l**=**tamano**/**p**;**  //Garantizar un minimo avance  **if(**l**==**0**)**  l**=**1**;**  //Asignacion  a**=**l**;**  **for(**i**=**0**;**i**<**p**-**1**;++**i**)**  **{**  final**[**i**]=**a**;**  a**+=**l**;**  **}**  final**[**p**-**1**]=**tamano**;**  //Para evitar accidentes  **for(**i**=**0**;**i**<**p**;++**i**)**  **{**  **if(**final**[**i**]>**tamano**)**  final**[**i**]=**tamano**;**  **}**  **return** final**;**  **}**  /\*\*  \* CalcularSubMatriz rellena la matriz a partes, entre dos columnas objetivo  \* @author Paul  \* @date 01/03/2018  \* @param matrix Matriz sobre la que se opera  \* @param c1 indice de la columna que calcularemos  \* @param c2 indice de la columna que calcularemos  \* @param string1 Cadena de texto  \* @param string2 Cadena de texto  \* @param locks Array de cerrojos  \* @param id Identificador del hilo que esta ejecutando esta funcion  \*/  void CalcularSubMatriz**(**struct Celda**\*\*** matrix**,** unsigned c1**,** unsigned c2**,** char**\*** string1**,** char**\*** string2**,** unsigned**\*** locks**,** unsigned id**)**  **{**  int tiempo **=** 500**;**  unsigned size1**=**strlen**(**string1**);**  unsigned i**;**  unsigned j**;**    **for(** i **=** 1**;** i **<=** size1**;** **++**i**)**  **{**  **for(** j **=** c1**;** j **<=** c2**;** **++**j**)**  **{**  **while(** id **>** 0 **&&** locks**[**id **-** 1**]** **<=** locks**[**id**]** **)**  **{**  usleep**(**tiempo**);**  **}**    CalcularCasilla**(**i**,** j**,** string1**[**i **-** 1**]** **==** string2**[**j**-**1**]** **||** string1**[**i **-** 1**]** **==** 'N' **||** string2**[**j**-**1**]** **==** 'N'**,** matrix**);**  **}**  locks**[**id**]++;**  **}**      **return;**  **}**  /\*\*  \* CalcularCasilla funcion que calcula el contenido de una casillo de la matriz para el algoritmo Needleman-Wunsch  \* @author Nacho  \* @date 6/2/2018  \* @param i Indice de fila (No puede ser 0)  \* @param j Indice de columna (No puede ser 0)  \* @param igual Comparativa (Char==Char)  \* @param matrix Matriz de structs sobre la que se opera. In/Out  \*/  void CalcularCasilla**(**unsigned i**,** unsigned j**,** bool igual**,** struct Celda **\*\***matrix**)**  **{**  // Constantes Match 2, -2 dismatch, -1 Hueco  int A **=** matrix**[**i**-**1**][**j**].**score **-** 1**;**  int B **=** matrix**[**i**][**j**-**1**].**score **-** 1**;**  int C **=** matrix**[**i**-**1**][**j**-**1**].**score**+** **(**igual**\***4**)-**2**;** //C= arg + (argB\*(Match-Fallo))+Fallo    //Calculo de la dirección como un array de booleanos  int D**=**0**;**  D **+=** **(**A**>=**B **&&** A**>=**C**)<<**1**;** //Vertical en 1a posicion  D **+=** **(**B**>=**A **&&** B**>=**C**);** //Horizontal en 2a posicion  D **+=** **(**C**>=**B **&&** C**>=**A**)<<**2**;** //Diagonal en 3a posicion  matrix**[**i**][**j**].**dir **=** D**;**    matrix**[**i**][**j**].**score**=**maxI**(**maxI**(**A**,**B**),**C**);**    **}**  /\*\*  \* GetRuta Funcion de backtraking para saber cual es el mayor indice de coincidencias. Aumenta la cuenta si encuentra diagonales.  \* @author Paul  \* @author Nacho en la optimización  \* @date 12/2/2018  \* @date 14/2/2018 en optimizacion  \* @param matrix Matriz de structs sobre la que se opera. In/Out  \* @param i Indice de fila inicial (Debe ser la ultima)  \* @param j Indice de columna inicial (Debe ser la ultima)  \*/  unsigned GetRuta**(**struct Celda**\*\*** matrix**,** unsigned i**,** unsigned j**)**  **{**  unsigned maximo **=** 0**;**  //Reinicializado de los scores, ahora mediremos la distancia a la esquina 0,0  //El valor -1 representa dato desconocido  unsigned x**,**y**;**  **for(**x**=**0**;**x**<=**i**;**x**++)**  **for(**y**=**0**;**y**<=**j**;**y**++)**  matrix**[**x**][**y**].**score**=-**1**;**    AuxGetRuta**(**matrix**,** i**,** j**,** 0**,** **&**maximo**);**    **return** maximo**;**  **}**  int AuxGetRuta**(**struct Celda**\*\*** matrix**,** unsigned i**,** unsigned j**,** int cont**,** unsigned **\***maximo**)**  **{**    int A**=-**1**,**B**=-**1**,**C**=-**1**;**  //Poda, impide repetición de celdas si ya ha calculado el camino  **if(**matrix**[**i**][**j**].**score**>=**0**)**  **{**  **return** matrix**[**i**][**j**].**score**;**  **}**  //Poda, impide recorrer un nuevo camino si no hay posibilidad de superar la mejor marca  **if((**cont**+**i**<\***maximo **||** cont**+**j**<\***maximo**)&&(\***maximo**>**0**))**  **{**  matrix**[**i**][**j**].**score**=**0**;**  **return** 0**;**  **}**  //Caso base  **if(**i **==** 0 **||** j **==** 0**)**  **{**  **\***maximo **=** maxI**(**cont**,** **\***maximo**);**  matrix**[**i**][**j**].**score**=**0**;;**  **return** 0**;**  **}**  **else**  **{**  **if(**matrix**[**i**][**j**].**dir**>**3**)**  **{**  //En las diagonales se añade distancia si existen  A**=**AuxGetRuta**(**matrix**,** i **-** 1**,** j **-** 1**,** cont **+** 1**,** maximo**)+**1**;**  **}**  **if(**matrix**[**i**][**j**].**dir**==**2**||**matrix**[**i**][**j**].**dir**==**3**||**matrix**[**i**][**j**].**dir**==**6**||**matrix**[**i**][**j**].**dir**==**7**)**  **{**  B**=**AuxGetRuta**(**matrix**,** i **-** 1**,** j**,** cont**,** maximo**);**  **}**  **if(**matrix**[**i**][**j**].**dir**==**1**||**matrix**[**i**][**j**].**dir**==**3**||**matrix**[**i**][**j**].**dir**==**5**||**matrix**[**i**][**j**].**dir**==**7**)**  **{**  C**=**AuxGetRuta**(**matrix**,** i**,** j **-** 1**,** cont**,** maximo**);**  **}**  **}**    matrix**[**i**][**j**].**score**=**maxI**(**maxI**(**A**,**B**),**C**);**  **return** matrix**[**i**][**j**].**score**;**  **}** |

## 6-Tiempo de trabajo

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Tiempo Total  (Horas) | Días | Personas | Trabajo diario medio  (Minutos) |
| Pruebas | 2 | 1 | 4 | 120 |
| Realización de documentación | 6 | 2 | 1 | 180 |
| Total: | 8 | 3 | Variable | 180 |

## Anexo: Autoevaluación

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Proyecto: | Similitud de cadenas de ADN mediante el algoritmo de alineamiento NW | |  |  |
|  |  | |  |  |
| Autores: | Ignacio Gomis Lli | |  |  |
|  | Lidia Montero Egidos | |  |  |
|  | Sara Monzó Bravo | |  |  |
|  | Paul Vargas Hurtado | |  |  |
|  |  | |  |  |
|  | Concepto evaluado | | Valor |  |
|  | Completitud de estudio experimental (0-1) | | 0.8 |  |
|  | Claridad de redacción y presentación de los datos. (0-1) | | 1 |  |
|  | Código del algoritmo final en OpenMP. (0-1) | | 1 |  |
|  | Representación de los datos mediante tablas y/o gráficas. (0-1) | | 0.9 |  |
|  | Completitud de las medidas y tallas utilizadas en el estudio experimental. (0-1) | | 0.9 |  |
|  | Estudio de diferentes opciones de planificación. (0-1) | | 0.5 |  |
|  | Estudio experimental de la velocidad, aceleración y eficiencia. (0-2) | | 2 |  |
|  | Estudio experimental de la escalabilidad (0-1) | | 0.5 |  |
|  | Justificación de los datos experimentales obtenidos en función del algoritmo y la arquitectura del ordenador. (0-1) | | 0 |  |
|  |  |
|  |  |
|  |  |
|  | **TOTAL (Sobre 10)** | | 7.6 |  |
|  |  | |  |  |
| En caso de plagio o contenido sin referenciar, no se aceptará el documento para su evaluación | | | |  |
|  |  | |  |  |
|  | ¿PLAGIO? |  |  |  |
|  |  |  |  |  |

# CP3: Propuesta de implementación con MPI

## 1-Descripción del algoritmo paralelo

Tras conocer el funcionamiento de MPI y pensar en como aplicarlo a nuestro algoritmo, nuestra primera opción fue comparar la solución realizada con OMP para ver si podría ser aplicable con modificaciones en el código.

Afortunadamente, la solución que propusimos en su momento para OMP es fácilmente aplicable a MPI, ya que, si bien tendremos que realizar cambios para que los procesadores se comuniquen entre sí, el problema que teníamos antes con las esperas se resuelve de manera natural en MPI, sin tener que crear ningún método de espera por nuestra parte.

Por recapitular nuestra solución, emplearemos el paradigma de programación de paralelismo de datos para la resolución de la matriz. Sin embargo no paralelizaremos nada más con MPI.

Dadas las dependencias del algoritmo descritas durante el [CP1](#_4-Análisis_de_dependencia) podemos observar dos posibles soluciones, paralelizar antidiagonales o paralelizar bloques asíncronamente, es decir, cada uno esperando a que los datos para avanzar estén disponibles.

Nos decidimos por la segunda opción porque paralelizar las antidiagonales, para garantizar resultados correctos requeriría muchos cambios de bloques de memoria o una solución asíncrona similar a la de los bloques.

Desde un principio nuestro código fue desarrollado con la intención de poder repartir el trabajo en múltiples bloques, así que para esta entrega, lo más destacable a variar del algortimo será crear una nueva variación de la función CompletarMatriz, adaptada para MPI de manera que reparta entre los distintos procesadores la matriz y estos procesadores se comuniquen entre sí los datos pertinentes.

Aunque dejamos el código de OMP preparado para hacer llamadas al programa mediante MPI, tras observar que se requiere una distinta llamada para compilar y ejecutar hemos decidido simplemente crear un nuevo fichero de código, con un main más simple, solo para el caso con MPI.

Para recordar el método que estamos proponiendo, adjuntamos el siguiente diagrama:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | | | | | | |
| 0 | | | | 1 | | | | 2 | | | | 3 | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Tenemos en la matriz repartida en 4 bloques, delimitados por las líneas discontinuas y habiendo calculado ya las casillas coloreadas. Se puede observar que hay adjunto un vector que indica el número de filas completadas por cada bloque. Cuando un hilo completa una fila incrementa en uno su indicador, y luego comprueba si puede comenzar la siguiente iteración, comprobando que sus filas terminadas sean menores que las del bloque anterior, que en caso contrario, simplemente dormirá esperando el momento indicado. En este caso, el hilo 2 estaría esperando a que el hilo 1 completase su fila.

Propusimos una solución en OMP para reducir los paros de los procesadores, y los resultados no fueron positivos en absoluto, como se puede ver en la [Practica 1](#_3-Estudio_del_entrelazado). Por recordar el esquema, lo mostramos a continuación:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | | 1 | | 2 | | 3 | | 0 | | 1 | | 2 | | 3 | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Esto deberá ser probado todavía en MPI, pues nos cabe la duda sobre el tiempo que un mensaje MPI no bloqueante puede estar esperando a que haya un receptor. Si este tiempo fuera suficiente, no habrá problema en implementarlo de la misma manera que implementamos para OMP, distribuyendo a cada procesador varias submatrices, con el objetivo de reducir los tiempos de inactividad de cada uno de los procesadores.

Dado que la arquitectura se basa en paso de mensajes, ¿en este algoritmo cuantas comunicaciones realizaremos?

En primer lugar tendremos que hacer que cada procesador tenga ambos strings, realizando dos Broadcasts. En segundo lugar, necesitaremos que cada procesador tenga su primera fila inicializada, la cual será transmitida con un Scatter, mientras que el procesador 0 tendrá inicializada la primera columna. Por otro lado, para recoger los datos finales, tendremos que realizar un bucle for de Gathers para recoger cada fila de la matriz. Esta decisión puede sonar poco eficiente, pero la alternativa sería trasponer la matriz y operar por columnas, cosa que no creemos que sea adecuada, ya que forzaría a realizar muchos cambios de bloques de memoria. Cuanto nos penaliza esta decisión se muestra en el siguiente cálculo:

Sean F filas, C columnas, P procesadores, Tin tiempo de inicialización y Tc tiempo de comunicación.

Por tanto, nuestro tiempo de empeora es de

Las comunicaciones internas de la matriz, serán 1 comunicación de un único entero por cada fila y cada procesador, salvo el último. Por tanto:

F\*(P-1) sends

El cómputo total de tiempo de comunicaciones resulta:

|  |  |  |
| --- | --- | --- |
| Operación | Tipo de dato | Tiempo |
| 2 BCast | Char\* |  |
| Scatter | Struct |  |
| F\*(P-1) Send | Int |  |
| F Gather | Struct |  |

Con Tcc el coste de una comunicación para enviar un Char, Tcs el coste de una comunicación para enviar un struct de 16 Bytes y Tci el coste de una comunicación para enviar un Int

Finalmente, el coste de memoria del algoritmo se incrementará. Considerando despreciables buffers de comunicaciones (2 enteros por procesador) y los vectores con los strings, y tomando solo en consideración la matriz, nuestro consumo total de memoria se dobla, ya que tenemos en el primer proceso toda una matriz, y en cada procesador un fragmento de la misma. Esto nos complicará el cálculo para nuestro caso mayor, pero lo podremos solucionar destinando los procesos a distintos nodos, reduciendo el coste en memoria del primer computador del 200% a (100+(N\*100/P)) % siendo P el numero de procesos y N la cantidad ejecutada en ese computador.

Este último coste, podría reducirse en un (100/P %) si el primer proceso operase sobre la propia matriz, asi que intentaremos dar esta solución para que nuestro caso mayor probado previamente pueda ser ejecutado en BOE.

## 2-Implementacion del algoritmo

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <stdbool.h>  #include <string.h>  #include <time.h>  #include "mpi.h"  /\*Struct con el valor de la celda y las direcciones  Si las direcciones son verdadero, entonces ha heredado de esa direccion  \*/  struct Celda  **{**  int score**;**  char dir**;** //Array de booleanos  **};**  void ImprimirInstruccionesDeUso**();**  char**\*** CargarFichero**(**char**\*,**unsigned**,**unsigned**);**  struct Celda**\*\*** inicializarMatriz**(**unsigned**,** unsigned**);**  void CompletarMatrizMPI**(**char**\*** string1**,** char**\*,** struct Celda**\*\*,** int**);**  void CalcularSubMatrizMPI**(**struct Celda**\*\*,** char**\*,** char**\*,** int**,**int**);**  void CalcularCasilla**(**unsigned**,** unsigned**,** bool**,** struct Celda**\*\*,**bool**,**int**,**int**,**int**);**  unsigned GetRuta**(**struct Celda**\*\*,**unsigned**,**unsigned**);**  int AuxGetRuta**(**struct Celda**\*\*,** unsigned**,** unsigned**,** int**,** unsigned**\*);**  void print**(**struct Celda**\*\*,** char**\*,**int**,**int**);**  //Funciones auxiliares  /\*Maximo entre dos valores\*/  unsigned maxU**(**unsigned arg1**,** unsigned arg2**)**  **{**  **if(**arg1**>**arg2**){**  **return** arg1**;**  **}**  **else**  **return** arg2**;**  **}**  int maxI**(**int arg1**,** int arg2**)**  **{**  **if(**arg1**>**arg2**){**  **return** arg1**;**  **}**  **else**  **return** arg2**;**  **}**  /\*Funcion para pasar a mayuscula una cadena. Obtenida de:  https://stackoverflow.com/questions/35181913/converting-char-to-uppercase-in-c  \*/  void Mayus**(**char **\*** temp**)** **{**  char **\*** name**;**  name **=** strtok**(**temp**,**":"**);**  char **\***s **=** name**;**  **while** **(\***s**)** **{**  **\***s **=** toupper**((**unsigned char**)** **\***s**);**  s**++;**  **}**  **}**  /\*\*  \* Main Lleva control del orden de ejecucion de las funciones del algoritmo y la entrada de parametros  \* @author Lidia y Nacho  \* @date 8/2/2018  \* @date 24/4/2018 Rehecho unicamente para MPI  \* @param Nombre o ruta de Fichero1 (Necesario)  \* @param Nombre o ruta de Fichero2 (Necesario)  \* @param Tamano de cadena de fichero1 (Opcional o ambos ficheros si falta siguiente parametro)  \* @param Tamano de cadena de Fichero2 (Opcional)  \* @param Inicio de cadena de Fichero1 (Opcional o ambos ficheros si falta siguiente parametro)  \* @param Inicio de cadena de Fichero2 (Opcional)  \*/  int main**(** int argc**,** char **\***argv**[]** **)**  **{**  char**\*** string1**;**  char**\*** string2**;**  struct Celda **\*\***Matriz**;**  MPI\_Init**(&**argc**,&**argv**);**  int rank**,**x**,**y**;**  MPI\_Comm\_rank**(**MPI\_COMM\_WORLD**,&**rank**);**  unsigned T1**,**T2**,**I1**,**I2**,** sobrecarga**;**  char**\*** nombre1**;**  char**\*** nombre2**;**  bool lecturaCorrecta**=**1**;**  struct timeval t1**,**t2**,**t3**,**t4**;**  double total**;**  **if(**argc**>=**4**)**  **{**  sobrecarga**=**atoi**(**argv**[**1**]);**  nombre1**=**argv**[**2**];**  nombre2**=**argv**[**3**];**  **switch(**argc**)**  **{**  **case** 4**:** //Ambos desde el principio hasta lo que coja  T1**=**1000**;**T2**=**1000**;**I1**=**0**;**I2**=**0**;**  **break;**  **case** 5**:** //Ambos con Tamano arg[4]  T1**=**atoi**(**argv**[**4**]);**T2**=**T1**;**I1**=**0**;**I2**=**0**;**  **break;**  **case** 6**:** //Cada uno con Tamano arg[4] y arg[5]  T1**=**atoi**(**argv**[**4**]);**T2**=**atoi**(**argv**[**5**]);**I1**=**0**;**I2**=**0**;**  **break;**  **case** 7**:** //Cada uno con Tamano arg[4] y arg[5] empezando desde arg[6]  T1**=**atoi**(**argv**[**4**]);**T2**=**atoi**(**argv**[**5**]);**I1**=**atoi**(**argv**[**6**]);**I2**=**I1**;**  **break;**  **case** 8**:** //Cada uno con Tamano arg[4] y arg[5] empezando desde arg[6] y arg[7]  T1**=**atoi**(**argv**[**4**]);**T2**=**atoi**(**argv**[**5**]);**I1**=**atoi**(**argv**[**6**]);**I2**=**atoi**(**argv**[**7**]);**  **break;**  **default:** //Instrucciones de uso  ImprimirInstruccionesDeUso**();**  lecturaCorrecta**=**0**;**  **break;**  **}**  **}**  **else**  **{**  ImprimirInstruccionesDeUso**();**  lecturaCorrecta**=**0**;**  **}**      **if(**lecturaCorrecta**==**1**)**  **{**  **if(**rank**==**0**)**  **{**  printf**(**"%s comparado con %s\n"**,**nombre1**,**nombre2**);**  printf**(**"Tamanos: %d %d\n"**,**T1**\***100**,**T2**\***100**);**  printf**(**"Puntos de inicio: %d %d\n"**,**I1**\***100**,**I2**\***100**);**      gettimeofday**(&**t1**,** **NULL);**  **}**  **if(**rank**==**0**){**  string1**=**CargarFichero**(**nombre1**,**T1**,**I1**);**  string2**=**CargarFichero**(**nombre2**,**T2**,**I2**);**  x**=**strlen**(**string1**);**  y**=**strlen**(**string2**);**  MPI\_Bcast**(&**x**,**1**,**MPI\_INT**,**0**,**MPI\_COMM\_WORLD**);**  MPI\_Bcast**(&**y**,**1**,**MPI\_INT**,**0**,**MPI\_COMM\_WORLD**);**  **}**  **else**  **{**  MPI\_Bcast**(&**x**,**1**,**MPI\_INT**,**0**,**MPI\_COMM\_WORLD**);**  MPI\_Bcast**(&**y**,**1**,**MPI\_INT**,**0**,**MPI\_COMM\_WORLD**);**  string1**=**malloc**(**x**);**  string2**=**malloc**(**y**);**  **}**  MPI\_Bcast**(&**string1**[**0**],**x**,**MPI\_CHAR**,**0**,**MPI\_COMM\_WORLD**);**  MPI\_Bcast**(&**string2**[**0**],**y**,**MPI\_CHAR**,**0**,**MPI\_COMM\_WORLD**);**  **if(**strlen**(**string1**)==**0 **||** strlen**(**string2**)==**0**)**  **{**  printf**(**"Una cadena esta vacia"**);**  exit**(**2**);**  **}**  **if(**rank**==**0**)**  **{**  Matriz**=**inicializarMatriz**(**strlen**(**string1**),**strlen**(**string2**));**  gettimeofday**(&**t2**,** **NULL);**  **}**    //if(rank==0)print(Matriz,"PreCalculo",7,7);  CompletarMatrizMPI**(**string1**,**string2**,**Matriz**,**sobrecarga**);**  //if(rank==0)print(Matriz,"Salida",7,7);  **if(**rank**==**0**){**  gettimeofday**(&**t3**,** **NULL);**  int resultado**=** GetRuta**(**Matriz**,**strlen**(**string1**),**strlen**(**string2**));**    gettimeofday**(&**t4**,** **NULL);**    total **=** **((**t2**.**tv\_sec **\*** 1000000 **+** t2**.**tv\_usec**)-(**t1**.**tv\_sec **\*** 1000000 **+** t1**.**tv\_usec**));**  printf**(**"Inicializado: %lf\n"**,** total **);**    total **=** **((**t3**.**tv\_sec **\*** 1000000 **+** t3**.**tv\_usec**)-(**t2**.**tv\_sec **\*** 1000000 **+** t2**.**tv\_usec**));**  printf**(**"Creacion de matriz: %lf\n"**,** total **);**    total **=** **((**t4**.**tv\_sec **\*** 1000000 **+** t4**.**tv\_usec**)-(**t3**.**tv\_sec **\*** 1000000 **+** t3**.**tv\_usec**));**  printf**(**"Backtracking: %lf\n"**,** total **)** **;**    total **=** **((**t4**.**tv\_sec **\*** 1000000 **+** t4**.**tv\_usec**)-(**t1**.**tv\_sec **\*** 1000000 **+** t1**.**tv\_usec**));**  printf**(**"Total: %lf\n"**,** total **);**  printf**(**"Coincidencia(porc): %d\n"**,** 100**\***resultado**/**maxU**(**strlen**(**string1**),**strlen**(**string2**)));**  **}**  **}**  **else**  **{**  exit**(**3**);**  **}**  MPI\_Barrier**(**MPI\_COMM\_WORLD**);**  MPI\_Finalize**();**  exit**(**0**);**  **return** 0**;**  **}**  /\*\*  \* ImprimirInstruccionesDeUso imprime la informacion de uso de este programa, asi como los autores.  \* @author Nacho  \* @date 3/03/2018  \* @date 24/4/2018 Adaptado a MPI  \*/  void ImprimirInstruccionesDeUso**()**  **{**  printf**(**"Programa para el calculo de porcentaje de similitud de dos cadenas de Adn\n"**);**  printf**(**"Autores:\n"**);**  printf**(**"\tIgnacio Gomis Lli\n"**);**  printf**(**"\tLidia Montero Egidos\n"**);**  printf**(**"\tSara Monzo Bravo\n"**);**  printf**(**"\tPaul Vargas Hurtado\n"**);**  printf**(**"Asignatura: Arquitectura de procesadores, 3ro Ingenieria Informatica\n"**);**  printf**(**"Profesorado:\n"**);**  printf**(**"\t Jose Manuel Claver Iborra\n"**);**  printf**(**"\t Adria Gimenez Pastor\n"**);**  printf**(**"Universidad de Valencia, ETSE\n"**);**  printf**(**"Curso 2017/2018\n"**);**  printf**(**"\n"**);**  printf**(**"Instrucciones de uso:\n"**);**  printf**(**"-Los ficheros deben estar escritos en formato fasta\n"**);**  printf**(**"-Los argumentos de tamano tienen un escalado 1:100, es decir, en caso de que\n"**);**  printf**(**"se introduzca un 1 nos referimos a la posicion 100 del fichero\n"**);**  printf**(**"\n"**);**  printf**(**"-Ejecucion con MPI, segundo argumento 2, requiere opcion de compilado -fopenmp\n"**);**  printf**(**"3 argumentos:\n Bloques\_Por\_Hilo Fichero\_1 Fichero\_2\n"**);**  printf**(**"4 argumentos:\n Bloques\_Por\_Hilo Fichero\_1 Fichero\_2 Tamano\_maximo\n"**);**  printf**(**"5 argumentos:\n Bloques\_Por\_Hilo Fichero\_1 Fichero\_2 TamanoMax\_Cadena1 TamanoMax\_Cadena2\n"**);**  printf**(**"6 argumentos:\n Bloques\_Por\_Hilo Fichero\_1 Fichero\_2 TamanoMax\_Cadena1 TamanoMax\_Cadena2 Inicio\_Cadenas\n"**);**  printf**(**"7 argumentos:\n Bloques\_Por\_Hilo Fichero\_1 Fichero\_2 TamanoMax\_Cadena1 TamanoMax\_Cadena2 Inicio\_C1 Inicio\_C2\n"**);**  **}**  /\*\*  \* CargarFichero funcion que guarda en un string el contenido de un fichero .fasta  \* @author Lidia y Nacho  \* @date 8/2/2018  \* @param NombreFichero nombre del fichero, incluida extension y ruta relativa  \* @out string con el contenido en mayusculas  \*/  char**\*** CargarFichero**(**char**\*** NombreFichero**,**unsigned tamano**,**unsigned inicio**)**  **{**  tamano**\*=**100**;**  FILE **\***archivo**;**  unsigned i**;**  char caracteres**[**100**];**  char **\***cadena**=**malloc**(**tamano**);** //<--Origen error  strcpy **(**cadena**,** ""**);**  archivo **=** fopen**(**NombreFichero**,**"r"**);**  **if** **(**archivo **==** **NULL)**  **{**  printf**(**"%s no existe"**,**NombreFichero**);**  exit**(**1**);**  **}**  **else**  **{**  fgets**(**caracteres**,**100**,**archivo**);** //Primera linea  **for(**i**=**0**;(**i**<**inicio**)&&(**feof**(**archivo**)** **==** 0**);**i**++)**  **{**  fgets**(**caracteres**,**100**,**archivo**);**  **}**  i**=**tamano**;**  **while** **(**feof**(**archivo**)** **==** 0**)** //Hasta fin de archivo o memoria  **{**  fgets**(**caracteres**,**100**,**archivo**);**  strcat**(**cadena**,** caracteres**);**  **}**  **}**  Mayus**(**cadena**);**  **return** cadena**;**  **}**  // inicializarMatriz funcion que crea la matriz de tamano r c e inicializa la primera fila y columna con valores negativos descendentes.  // @author Sara  // @date 12/2/2018  // @param unsigned r rows  // @param unsigned c cols  // @return arr matriz con los valores negativos  struct Celda**\*\*** inicializarMatriz**(**unsigned r**,** unsigned c**)**  **{**  unsigned i**;**  struct Celda **\*\***arr **=(**struct Celda **\*\*)**malloc**((**r**+**1**)\*** **sizeof(**struct Celda **\*));**  struct Celda **\***mem**=** **(**struct Celda **\*)** malloc**((**r**+**1**)\*(**c**+**1**)\*sizeof(**struct Celda**));**  **for** **(**i**=**0**;**i**<=**r**;++**i**)**  arr**[**i**]=** mem **+** i**\*(**c**+**1**);**  //Casos base posicion: r = 0, c = 0  arr**[**0**][**0**].**score **=** 0**;**  arr**[**0**][**0**].**dir **=** 0**;**  **for(**i **=** 1 **;** i**<=**r**;** i**++)**  **{**  arr**[**i**][**0**].**score **=** **-**i**;**  arr**[**i**][**0**].**dir**=**0**;**  **}**  **for(**i **=** 1 **;** i**<=**c**;** i**++)**  **{**  arr**[**0**][**i**].**score **=** **-**i**;**  arr**[**0**][**i**].**dir**=**0**;**  **}**  **return** arr**;**  **}**  void CompletarMatrizMPI**(**char**\*** string1**,** char**\*** string2**,** struct Celda**\*\*** Matriz**,** int sobrecarga**)**  **{**  int P**,**id**,**i**,**j**,**r**,**l**;**  MPI\_Barrier**(**MPI\_COMM\_WORLD**);**  MPI\_Comm\_size**(**MPI\_COMM\_WORLD**,&**P**);**  MPI\_Comm\_rank**(**MPI\_COMM\_WORLD**,&**id**);**  //Crear datos locales  **if(**id**==**0**)**  **{**  r**=(**strlen**(**string1**)+**1**);**  l**=**strlen**(**string2**);**  **}**  MPI\_Bcast**(&**r**,** 1**,** MPI\_INT**,** 0**,** MPI\_COMM\_WORLD**);**  MPI\_Bcast**(&**l**,** 1**,** MPI\_INT**,** 0**,** MPI\_COMM\_WORLD**);**  int k**=**l**%**P**;**  int num**=((**l**-**k**)/**P**);**  struct Celda **\*\***matriz\_l **=(**struct Celda **\*\*)**malloc**((**r**)\*** **sizeof(**struct Celda **\*));**  **if(**id**!=**0**){**  struct Celda **\***mem**=** **(**struct Celda **\*)** malloc**(**r**\***num**\*sizeof(**struct Celda**));**  **for** **(**i**=**0**;**i**<**r**;++**i**)**  matriz\_l**[**i**]=** mem **+** i**\*(**num**);**  **}**  **else{**  struct Celda **\***mem**=** **(**struct Celda **\*)** malloc**(**r**\*(**num**+(**k**)+**1**)\*sizeof(**struct Celda**));**  **for** **(**i**=**0**;**i**<**r**;++**i**)**  matriz\_l**[**i**]=** mem **+** i**\*(**num**+(**k**)+**1**);**  **}**  //Crear tipo para Struct  // Origen de código: https://stackoverflow.com/questions/9864510/struct-serialization-in-c-and-transfer-over-mpi  const int nitems**=**2**;**  int blocklengths**[**2**]** **=** **{**1**,**1**};**  MPI\_Datatype types**[**2**]** **=** **{**MPI\_INT**,** MPI\_CHAR**};**  MPI\_Datatype mpi\_celda**;**  MPI\_Aint offsets**[**2**];**  offsets**[**0**]** **=** offsetof**(**struct Celda**,** score**);**  offsets**[**1**]** **=** offsetof**(**struct Celda**,** dir**);**  MPI\_Type\_create\_struct**(**nitems**,** blocklengths**,** offsets**,** types**,** **&**mpi\_celda**);**  MPI\_Type\_commit**(&**mpi\_celda**);**    //Enviar a los procesos  //MPI\_Bcast(string1, r-1, MPI\_CHAR, 0, MPI\_COMM\_WORLD);  **if(**id**==**0**)**  **{**  MPI\_Scatter**(&**Matriz**[**0**][**k**+**1**],** num**,** mpi\_celda**,** **&**matriz\_l**[**0**][**k**+**1**],** num**,**mpi\_celda**,** 0**,** MPI\_COMM\_WORLD**);**  **for(**i**=**0**;**i**<=**k**;**i**++)**  **{**  matriz\_l**[**0**][**i**]=**Matriz**[**0**][**i**];**  **}**  **for(**j**=**1**;**j**<**r**;**j**++)**  **{**  matriz\_l**[**j**][**0**]=**Matriz**[**j**][**0**];**  **}**  **}**  **else**  **{**  MPI\_Scatter**(&**Matriz**[**0**][**k**+**1**],** num**,** mpi\_celda**,** **&**matriz\_l**[**0**][**0**],** num**,**mpi\_celda**,** 0**,** MPI\_COMM\_WORLD**);**  //print(matriz\_l,"Local\_sub",7,num-1);  **}**    //Ejecutar código  CalcularSubMatrizMPI**(**matriz\_l**,**string1**,**string2**,**id**,**P**-**1**);**  //Recolectar de los procesos  **if(**id**==**0**)**  **{**  **for(**j**=**1**;**j**<**r**;**j**++){**  MPI\_Gather**(&**matriz\_l**[**j**][**k**+**1**],** num**,** mpi\_celda**,** **&**Matriz**[**j**][**k**+**1**],** num**,** mpi\_celda**,** 0**,** MPI\_COMM\_WORLD**);**  **for(**i**=**1**;**i**<=**k**;**i**++)**  **{**  Matriz**[**j**-**1**][**i**].**score**=**matriz\_l**[**j**-**1**][**i**].**score**;**  Matriz**[**j**-**1**][**i**].**dir**=**matriz\_l**[**j**-**1**][**i**].**dir**;**  **}}**  **}**  **else**  **{**  //print(matriz\_l,"Local\_sub",7,num-1);  **for(**j**=**1**;**j**<**r**;**j**++)**  MPI\_Gather**(&**matriz\_l**[**j**][**0**],** num**,** mpi\_celda**,** **&**Matriz**[**j**][**k**+**1**],** num**,** mpi\_celda**,** 0**,** MPI\_COMM\_WORLD**);**  **}**  //FinalizarMPI  MPI\_Barrier**(**MPI\_COMM\_WORLD**);**  **}**  /\*\*  \* CalcularSubMatrizMPI rellena una submatriz, de manera completa.  \* En el caso 0 con tamaño N se deben pasar como string2 un string de tamaño N-1.  \* En el resto de casos se deben pasar como string2 un string de tamaño N. Además recibe del anterior hilo los datos necesarios para el cálculo.  \* En todos los casos salvo el ultimo, envia los datos necesarios para el calculo al siguiente hilo  \* @author Nacho  \* @date 17/04/2018  \* @param matrix Submatriz sobre la que se opera  \* @param tam ancho de matriz (columnas)  \* @param string1 Subcadena de texto 1 (filas)  \* @param string2 Subcadena de texto 2 (columnas)  \* @param id Identificador del hilo que esta ejecutando esta funcion  \* @param maxId Maximo identificador de hilo  \*/  void CalcularSubMatrizMPI**(**struct Celda**\*\*** matrix**,** char**\*** string1**,** char**\*** string2**,** int id**,**int maxId**)**  **{**  MPI\_Status status**;**  int size1**=**strlen**(**string1**);** //Tamaño dependiente del char\* recibido, filas  int tam**;**  **if(**id**>**0**)**  tam**=((**strlen**(**string2**)-**strlen**(**string2**)%(**maxId**+**1**))/(**maxId**+**1**));**  **else**  tam**=((**strlen**(**string2**)-**strlen**(**string2**)%(**maxId**+**1**))/(**maxId**+**1**))+**strlen**(**string2**)%(**maxId**+**1**);**  int minBlucle**=**id**\***tam**+**7**%**3**;**  int i**;**  int j**;**  int valor1**;**  int valor2**=**0**;**  valor1**=**matrix**[**0**][**0**].**score**+**1**;**  **for(** i **=** 1**;** i **<=** size1**;** **++**i**)** //PARA CADA FILA  **{**  **if(**id**!=**0**)**  **{**    valor2**=**valor1**;** //Desplazar buffer de datos    MPI\_Recv**(&**valor1**,**1**,**MPI\_INT**,**id**-**1**,**i**,**MPI\_COMM\_WORLD**,&**status**);** //Recibir dato para nueva linea    CalcularCasilla**(**i**,** 0**,** string1**[**i**-**1**]** **==** string2**[**minBlucle**]** **||** string1**[**i**-**1**]** **==** 'N' **||** string2**[**minBlucle**]** **==** 'N'**,** matrix**,**true**,**valor1**,**valor2**,**id**);**  **for(** j **=** 1**;** j **<** tam**;** **++**j**)**  **{**  CalcularCasilla**(**i**,** j**,** string1**[**i**-**1**]** **==** string2**[**minBlucle**+**j**]** **||** string1**[**i**-**1**]** **==** 'N' **||** string2**[**minBlucle**+**j**]** **==** 'N'**,** matrix**,**false**,**0**,**0**,**id**);**  **}**  **if(**id**!=**maxId**)** //Si no es el ultimo  MPI\_Send**(&**matrix**[**i**][**tam**].**score**,**1**,**MPI\_INT**,**id**+**1**,**i**,**MPI\_COMM\_WORLD**);** //Enviar dato a siguiente procesador  **}**  **else**  **{**  **for(** j **=** 1**;** j **<=** tam**;** **++**j**)**  **{**  CalcularCasilla**(**i**,** j**,** string1**[**i**-**1**]** **==** string2**[**j**-**1**]** **||** string1**[**i**-**1**]** **==** 'N' **||** string2**[**j**-**1**]** **==** 'N'**,** matrix**,**false**,**0**,**0**,**id**);**  **}**  **if(**id**!=**maxId**)** //Si no es el ultimo  MPI\_Send**(&**matrix**[**i**][**tam**].**score**,**1**,**MPI\_INT**,**id**+**1**,**i**,**MPI\_COMM\_WORLD**);** //Enviar dato a siguiente procesador  **}**  **}**  **return;**  **}**  /\*\*  \* CalcularCasilla funcion que calcula el contenido de una casillo de la matriz para el algoritmo Needleman-Wunsch  \* @author Nacho  \* @date 17/4/2018  \* @param i Indice de fila (No puede ser 0)  \* @param j Indice de columna (No puede ser 0)  \* @param igual Comparativa (Char==Char)  \* @param matrix Matriz de structs sobre la que se opera. In/Out  \* @param MPIrec booleano sobre si debe utilizar valores pasados por argumentos (true) o de la matriz (false)  \* @param MPIData1 Valor a la izquierda del calculado si MPIrec  \* @param MPIData2 Valor a la diagonal del calculado si MPIrec  \*/  void CalcularCasilla**(**unsigned i**,** unsigned j**,** bool igual**,** struct Celda **\*\***matrix**,**bool MPIrec**,**int MPIData1**,** int MPIData2**,** int id**)**  **{**  int A**,**B**,**C**;**  **if(!**MPIrec**){**  // Constantes Match 2, -2 dismatch, -1 Hueco  A **=** matrix**[**i**-**1**][**j**].**score **-** 1**;**  B **=** matrix**[**i**][**j**-**1**].**score **-** 1**;**  C **=** matrix**[**i**-**1**][**j**-**1**].**score**+** **(**igual**\***4**)-**2**;** //C= arg + (argB\*(Match-Fallo))+Fallo  **}**  **else**  **{**  A **=** matrix**[**i**-**1**][**j**].**score **-** 1**;**  B **=** MPIData1 **-** 1**;**  C **=** MPIData2**+** **(**igual**\***4**)-**2**;**  **}**    //Calculo de la direccion como un array de booleanos  int D**=**0**;**  D **+=** **(**A**>=**B **&&** A**>=**C**)<<**1**;** //Vertical en 1a posicion  D **+=** **(**B**>=**A **&&** B**>=**C**);** //Horizontal en 2a posicion  D **+=** **(**C**>=**B **&&** C**>=**A**)<<**2**;** //Diagonal en 3a posicion  **if(**id**==**0**)**  **{**  matrix**[**i**][**j**].**dir **=** D**;**  matrix**[**i**][**j**].**score**=**maxI**(**maxI**(**A**,**B**),**C**);**  **}**  **else**  **{**  matrix**[**i**][**j**].**dir **=** D**;**  matrix**[**i**][**j**].**score**=**maxI**(**maxI**(**A**,**B**),**C**);**  **}**  **}**  /\*\*  \* GetRuta Funcion de backtraking para saber cual es el mayor indice de coincidencias. Aumenta la cuenta si encuentra diagonales.  \* @author Paul  \* @author Nacho en la optimización  \* @date 12/2/2018  \* @date 14/2/2018 en optimizacion  \* @param matrix Matriz de structs sobre la que se opera. In/Out  \* @param i Indice de fila inicial (Debe ser la ultima)  \* @param j Indice de columna inicial (Debe ser la ultima)  \*/  unsigned GetRuta**(**struct Celda**\*\*** matrix**,** unsigned i**,** unsigned j**)**  **{**  unsigned maximo **=** 0**;**  //Reinicializado de los scores, ahora mediremos la distancia a la esquina 0,0  //El valor -1 representa dato desconocido  unsigned x**,**y**;**  **for(**x**=**0**;**x**<=**i**;**x**++)**  **for(**y**=**0**;**y**<=**j**;**y**++)**  matrix**[**x**][**y**].**score**=-**1**;**    AuxGetRuta**(**matrix**,** i**,** j**,** 0**,** **&**maximo**);**    **return** maximo**;**  **}**  int AuxGetRuta**(**struct Celda**\*\*** matrix**,** unsigned i**,** unsigned j**,** int cont**,** unsigned **\***maximo**)**  **{**    int A**=-**1**,**B**=-**1**,**C**=-**1**;**  //Poda, impide repetición de celdas si ya ha calculado el camino  **if(**matrix**[**i**][**j**].**score**>=**0**)**  **{**  **return** matrix**[**i**][**j**].**score**;**  **}**  //Poda, impide recorrer un nuevo camino si no hay posibilidad de superar la mejor marca  **if((**cont**+**i**<\***maximo **||** cont**+**j**<\***maximo**)&&(\***maximo**>**0**))**  **{**  matrix**[**i**][**j**].**score**=**0**;**  **return** 0**;**  **}**  //Caso base  **if(**i **==** 0 **||** j **==** 0**)**  **{**  **\***maximo **=** maxI**(**cont**,** **\***maximo**);**  matrix**[**i**][**j**].**score**=**0**;;**  **return** 0**;**  **}**  **else**  **{**  **if(**matrix**[**i**][**j**].**dir**>**3**)**  **{**  //En las diagonales se añade distancia si existen  A**=**AuxGetRuta**(**matrix**,** i **-** 1**,** j **-** 1**,** cont **+** 1**,** maximo**)+**1**;**  **}**  **if(**matrix**[**i**][**j**].**dir**==**2**||**matrix**[**i**][**j**].**dir**==**3**||**matrix**[**i**][**j**].**dir**==**6**||**matrix**[**i**][**j**].**dir**==**7**)**  **{**  B**=**AuxGetRuta**(**matrix**,** i **-** 1**,** j**,** cont**,** maximo**);**  **}**  **if(**matrix**[**i**][**j**].**dir**==**1**||**matrix**[**i**][**j**].**dir**==**3**||**matrix**[**i**][**j**].**dir**==**5**||**matrix**[**i**][**j**].**dir**==**7**)**  **{**  C**=**AuxGetRuta**(**matrix**,** i**,** j **-** 1**,** cont**,** maximo**);**  **}**  **}**    matrix**[**i**][**j**].**score**=**maxI**(**maxI**(**A**,**B**),**C**);**  **return** matrix**[**i**][**j**].**score**;**  **}**  void print**(**struct Celda**\*\*** matrix**,**char**\*** titulo**,**int X**,**int Y**)**  **{**  int x**;**  int y**;**  printf**(**"%s"**,**titulo**);**  printf**(**"\n"**);**  **for(**x**=**0**;**x**<=**X**;**x**++)** **{**  **for(**y**=**0**;**y**<=**Y**;**y**++)**  printf**(**"%d "**,** matrix**[**x**][**y**].**score**);**  printf**(**"\n"**);**  **}**  printf**(**"\n"**);**  printf**(**"\n"**);**  **for(**x**=**0**;**x**<=**X**;**x**++)** **{**  **for(**y**=**0**;**y**<=**Y**;**y**++)**  printf**(**"%d "**,** matrix**[**x**][**y**].**dir**);**  printf**(**"\n"**);**  **}**  printf**(**"\n"**);**  **}** |

## 3-Análisis del algoritmo

### Coste computacional

Reduciendo la [tabla anterior](#_Coste_computacional_del) a los cambios realizados para operar con MPI, y siendo L el número de bloques en los que se ha partido la matriz.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Función | Lecturas Matriz | Escrituras Matriz | Lecturas Cadenas | Escrituras Cadenas | Operaciones |
| CalcularCasilla | 1 ó 3 | 2 | 4 | 0 | 17 ó 19 |
| CompletarSubmatriz (hilo>0) | 1 | 0 | 0 | 0 | 1 |
| CompletarMatriz(hilo 0) | M%L+N | M%L+N | 0 | 0 | 2(M%L+N) |
| Orden Esperado | N\*M | N\*M | N\*M | 0 | N\*M |

A pesar del aumento en coste de comunicaciones, podemos observar que los costes computacionales son menores que en el caso de OMP y de ordenes iguales al caso secuencial.

### Estimaciones

Nuestro algoritmo sigue la misma estrategia que en OMP asi que nuestra expectativa es similar a la mencionada entonces, ya que asumiendo grandes tamaños y bloques de carga similar tendremos un problema escalable. Cabe señalar que ahora no tenemos sistemas de locks, sino que tenemos paso de mensajes no bloqueantes, lo cual esperamos que tenga unos resultados aun más positivos que OMP. Por otro lado, la distribución de tamaños de vectores también ha cambiado, siendo el de mayor tamaño el primero. Eso tendrá un efecto negativo, ya que estimamos que tendríamos mayores costes si un procesador tiene mayor carga que sus procesadores siguientes. Otro dato a tener en cuenta es que en caso de emplear un sistema distribuido de nodos, las comunicaciones pueden tener mayores costes que si son dentro de un mismo procesador, y en problemas realmente grandes no podremos lanzar más de unos pocos procesos en el nodo que incluya al 0.

## Anexo: Tiempo trabajado

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Tiempo Total  (Horas) | Días | Personas | Trabajo diario medio  (Minutos) |
| Estudio de la paralelización | 0.25 | 1 | 1 | 15 |
| Implementación de código | 17 | 7 | 3 | 150 |
| Redacción de documentación | 6 | 2 | 2 | 150 |
| Total: | 23.25 | 10 | Variable | 150 |

## Anexo: Autoevaluación

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Proyecto: | Similitud de cadenas de ADN mediante el algoritmo de alineamiento NW | |  |  |
|  |  | |  |  |
| Autores: | Ignacio Gomis Lli | |  |  |
|  | Lidia Montero Egidos | |  |  |
|  | Sara Monzó Bravo | |  |  |
|  | Paul Vargas Hurtado | |  |  |
|  |  | |  |  |
|  | Concepto evaluado | | Valor |  |
|  | Completitud (0-0.5) | | 0.5 |  |
|  | Claridad de redacción (0-0.5) | | 0.4 |  |
|  | Referencia al análisis de dependencias en el CP1 (0-0.5) | | 0.5 |  |
|  | Discusión de paradigmas de programación aplicables al caso. (0-0.5) | | 0.3 |  |
|  | Análisis del coste computacional del algoritmo paralelo (0-2) | | 1.50 |  |
|  | Propuesta de rutinas de MPI para la paralelización del algoritmo (0-2) | | 1.75 |  |
|  | Estimación analítica de aceleración, eficiencia y escalabilidad (0-1) | | 0.25 |  |
|  |  |
|  |  |
|  |  |
|  | Nuevas referencias (0-0.5) | | 0 |  |
|  | Formato del documento (0-0.5) | | 0.5 |  |
|  | **TOTAL (Sobre 10)** | | 5.7 |  |
|  |  | |  |  |
| En caso de plagio o contenido sin referenciar, no se aceptará el documento para su evaluación | | | |  |
|  |  | |  |  |
|  | ¿PLAGIO? |  |  |  |
|  |  |  |  |  |

# Anexo: Referencias

Wikipedia (7/2/2018). Wikipedia. [online]. Available from [https://en.wikipedia.org/wiki/Needleman%E2%80%93Wunsch\_algorithm](https://en.wikipedia.org/wiki/Needleman%E2%80%93Wunsch_algorithm%20) [10/2/2018]

National Center for Biotechnology Information (n.d.). NCBI.[online]. Available from <https://www.ncbi.nlm.nih.gov/genome/?term=homo+sapien> [10/2/2018]

Needleman, Saul B. and Wunsch, Christian D. (1970). "A general method applicable to the search for similarities in the amino acid sequence of two proteins", in Journal of Molecular Biology. Vol. 48 No.3, pp .443–53.

Chetan (11/08/2008) Technology Blog. [online]. Available from <http://technology66.blogspot.com.es/2008/08/sequence-alignment-techniques.html> [10/02/2018]

Rauber, Thomas and Rünger, Gudula (2010). Parallel programming for multicore and cluster systems, Berlin, Springer-Verlag.