ANEXO 1:

CÓDIGO VERSIÓN SERIE

- Función *multiplealign* : encargada de leer el fichero que contiene las secuencias, guardarlas en memoria y posteriormente irle pasando un par de secuencias a la función *alinear*, que es la encargada de realizar un alineamiento de 2 secuencias basándose en el algoritmo Needleman-Wunsch

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <omp.h>
#include "alinear.c"
#include "sample.h"


void EliminarMemoria(char *memo)
{

	free(memo);
}


void FAlignmentSerie(char *memo, int tamany, int mida)
{
	int i,j;
	int s1,s2;
	float s;
	char *seq1, *seq2, *inicio;

	for (i=0; i < (tamany*tamany); i++)
	{
		s1 = i/tamany;
		s2 = i%tamany;

		j = 0;
		inicio = memo;

		if(*inicio != '\n')
		{
			seq1 = memo + ((mida+1)*s1) + j;
			seq2 = memo + ((mida+1)*s2) + j;
			j++;
			inicio++;
		}
		alinear(seq1,seq2);
	}
}

int main(int argc, char** argv)
{

	FILE *reg = NULL;
	int mida,nseqs,t,numthreads;
	int i=0;
	char *p;
	struct timespec t1,t2;

	numthreads = atoi(argv[1]);
	reg = fopen(argv[2], "r");

	fscanf(reg,"%d",&nseqs);
	fseek(reg,1,SEEK_CUR);

	fscanf(reg,"%d",&mida);
```

```
	fseek(reg,1,SEEK_CUR);
	mida= mida +2;

	char buffer[mida];	char *memo;

	memo = malloc(sizeof(char)*(mida+1)*nseqs);

	  Sample_Init(0, 0);

	while(fgets(buffer,mida,reg)!=NULL)
	{
		memcpy(memo+((mida+1)*i),buffer,mida);
		i++;
	}

	FAlignmentSerie(memo,nseqs,mida);


	Sample_End();

	EliminarMemoria(memo);
}
```

- Función *alinear* : recibe 2 secuencias como parámetros y realiza un alineamiento de éstas 2 secuencias basándose en el algoritmo de alineamiento de secuencias Needleman-Wunsch

```
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <stdlib.h>
#include <math.h>
#include "sample.h"

#define MAX2(A, B) (((A) > (B)) ? (A) : (B))
#define MATCHING(A,B) (((A) == (B)) ? (2) : (-1))
#define gap -2

int Tactual= 0;
int Talinear= 0;
int *matriu= NULL;
char *alin= NULL;


void alinear(char *s1, char *s2)
{
      int i,j;
      int N,M;
      int match, opcio1,opcio2,opcio3;

      struct timespec t1,t2;

//Paso 1 : Inicialización

      N=strlen(s1)-1;
      M=strlen(s2)-1;

      if( ((N+1)*(M+1)) > Tactual)
      {
            int incremento = floor(((N+1)*(M+1))*0.5);
            int vmalloc = (((M+1)*(N+1)) + incremento);
            Tactual = ((N+1)*(M+1)) + incremento ;
            free(matriu);
            matriu = malloc ( sizeof(int) * vmalloc);
      }

      for ( j=0 ; j<N+1 ; j++ )
      {
            matriu[j]=0;
      }

      for ( i=0 ; i<M+1 ; i++ )
      {
            matriu[i*(N+1)] = 0;
      }


//Paso 2 : Calculo de la Matriz de Needleman-Wunsch

    Sample_ON();
      int c, c2, s2char, match1, match2, r1,r2,r3,resultat1, resultat2;
      for(i=1;i<M+1;i++){

            c= (i-1)*(N+1);
            c2 = i*(N+1);
            s2char = s2[i-1];
```

```
            r1 = matriu[c];
            r2 = matriu[c2];

            for(j=1;j<N+1;j=j+2){

                  r3 = matriu[c+ j];
                  match1 = MATCHING(s1[j-1],s2char);
                  match2 = MATCHING(s1[j],s2char);

                  opcio1 = r1 + match1;
                  opcio2 = r2 + gap;
                  opcio3 = r3 + gap;
                  resultat1 = MAX2(opcio2, MAX2(opcio1,opcio3));

                  r1 = matriu[c + j+1];
                  opcio1 = r3 + match2;
                  opcio2 = resultat1 + gap;
                  opcio3 = r1 + gap;
                  r2 = MAX2 (opcio2, MAX2(opcio1,opcio3));

                  matriu[c2 + j] = resultat1;
                  matriu[c2 + j+1] = r2;
            }
      }

    Sample_OFF();


// Paso 3: Traceback

      int x,y;
      int puntuacio,puntuaciod,puntuaciou,puntuaciol,pos;
      x=M;
      y=N;

      if((M+N) > Talinear)
      {

            int incremento2 = floor((2 * (M+N))*0.5);
            int vmalloc2 = ((2* (M+N)) + incremento2);
            Talinear= (N+M) + incremento2;
            free(alin);
            alin = malloc ( sizeof(char) * vmalloc2);
      }

      pos= 0;
      for(i=0;i<2;i++){

            for(j=0;j<M+N;j++){

                  alin[i*(M+N) + j]=' ';

            }

      while(y > 0 && x > 0)
      {

            puntuacio = matriu[x*(N+1) + y];
            puntuaciod = matriu[((x-1)*(N+1)) + y-1];
            puntuaciou = matriu [((x-1)*(N+1)) + y];
            puntuaciol = matriu [(x*(N+1)) + y-1];
```

```
            if ( puntuacio == puntuaciod + MATCHING(s1[y-1],s2[x-1]) )
            {

                  alin[0*(M+N) + pos] = s1[y-1];
                  alin[1*(M+N) + pos] = s2[x-1];
                  x--;
                  y--;
                  pos++;

            }
            else if (puntuacio == puntuaciol + gap)
            {

                  alin[0*(M+N) + pos] = s1[y-1];
                  alin[1*(M+N) + pos] = '-';
                  y--;
                  pos++;

            }


            else if ( puntuacio == puntuaciou + gap)
            {

                  alin[0*(M+N) + pos] = '-';
                  alin[1*(M+N) + pos] = s2[x-1];
                  x--;
                  pos++;

            }

      }

      if(y > 0 && matriu[x*(N+1) + y]==0 )
      {
            while(y > 0)
            {
                  alin[0*(M+N) + pos] = s1[y-1];
                  alin[1*(M+N) + pos] = '-';
                  y--;
                  pos++;
            }
      }

      if(x > 0 && matriu[x*(N+1) + y]==0 )
      {
            while(x > 0)
            {
                  alin[0*(M+N) + pos] = '-';
                  alin[1*(M+N) + pos] = s2[x-1];;
                  x--;
                  pos++;
            }
      }

      int contador=0;
      FILE *f;
      f=fopen("output.txt","a+");
      fprintf(f,"\n");
      fwrite(alin,sizeof(char)*(N+M),2,f);
      fprintf(f,"\n");
      fclose(f);

}
```

ANEXO 2:

CÓDIGO VERSIÓN PARALELA VARIANTE LARGE-GRAINED

- Función *multiplealign* : encargada de leer el fichero que contiene las secuencias, guardarlas en memoria y posteriormente irle pasando un par de secuencias a la función *alinear*, que es la encargada de realizar un alineamiento de 2 secuencias basándose en el algoritmo Needleman-Wunsch. Se paraleliza el bucle encargado de pasar las secuencias a la función alinear para que cada thread alinee secuencias diferentes.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <omp.h>
#include "alinear.c"
#include "sample.h"


void EliminarMemoria(char *memo)
{
	free(memo);
}


void FAlignmentParalel(char *memo, int tamany, int mida)
{

	int i,j;
	float s;
	char *seq1, *seq2, *inicio;

	Sample_ON();

	#pragma omp parallel for default (shared) private(i) schedule(static)
	for (i=0; i < (tamany*tamany); i++)
	{

		int s1 = i/tamany;
		int s2 = i%tamany;

		j = 0;
		inicio = memo;

		if(*inicio != '\n')
		{
			seq1 = memo + ((mida+1)*s1) + j;
			seq2 = memo + ((mida+1)*s2) + j;
			j++;
			inicio++;
		}
		alinear(seq1,seq2);

	}
	Sample_OFF();

}


int main(int argc, char** argv)
{

	FILE *reg = NULL;
	int mida,nseqs,t,numthreads;
	int i=0;
	char *p;
	struct timespec t1,t2;
```

```
	numthreads = atoi(argv[1]);
	reg = fopen(argv[2], "r");

	fscanf(reg,"%d",&nseqs);
	fseek(reg,1,SEEK_CUR);

	fscanf(reg,"%d",&mida);
	fseek(reg,1,SEEK_CUR);
	mida= mida +2;

	char buffer[mida];
	char *memo;

	memo = malloc(sizeof(char)*(mida+1)*nseqs);

	Sample_Init(0,0);

	while(fgets(buffer,mida,reg)!=NULL)
	{
		memcpy(memo+((mida+1)*i),buffer,mida);
		i++;
	}

	omp_set_num_threads(numthreads);
	FAlignmentParalel(memo,nseqs,mida);

	Sample_End();

	EliminarMemoria(memo);
}
```

- Función *alinear* : recibe 2 secuencias como parámetros y realiza un alineamiento de éstas 2 secuencias basándose en el algoritmo de alineamiento de secuencias Needleman-Wunsch.

```
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <stdlib.h>
#include <math.h>
#include "sample.h"

#define MAX2(A, B) (((A) > (B)) ? (A) : (B))
#define MATCHING(A,B) (((A) == (B)) ? (2) : (-1))
#define gap -2

int Tactual= 0;
int Talinear= 0;
int *matriu= NULL;
char *alin= NULL;


#pragma omp threadprivate(Tactual,Talinear,matriu,alin)

void alinear(char *s1, char *s2)
{

      int i,j;
      int N,M;
      int match, opcio1,opcio2,opcio3;

      struct timespec t1,t2;

//Paso 1 : Inicialización

      N=strlen(s1)-1;
      M=strlen(s2)-1;

      if( ((N+1)*(M+1)) > Tactual)
      {
            int incremento = floor(((N+1)*(M+1))*0.5);
            int vmalloc = (((M+1)*(N+1)) + incremento);

            Tactual = ((N+1)*(M+1)) + incremento ;
            free(matriu);
            matriu = malloc ( sizeof(int) * vmalloc);
      }


      for ( j=0 ; j<N+1 ; j++ )
      {

            matriu[j]=0;

      }

      for ( i=0 ; i<M+1 ; i++ )
      {
            matriu[i*(N+1)] = 0;

      }
```

```
//Paso 2 : Calculo de la Matriz de Needleman-Wunsch

      Sample_ON();

      int c, c2, s2char, match1, match2, r1,r2,r3,resultat1, resultat2;
      for(i=1;i<M+1;i++){

            c= (i-1)*(N+1);
            c2 = i*(N+1);
            s2char = s2[i-1];

            r1 = matriu[c];
            r2 = matriu[c2];

            for(j=1;j<N+1;j=j+2){

                  r3 = matriu[c+ j];
                  match1 = MATCHING(s1[j-1],s2char);
                  match2 = MATCHING(s1[j],s2char);

                  opcio1 = r1 + match1;
                  opcio2 = r2 + gap;
                  opcio3 = r3 + gap;
                  resultat1 = MAX2(opcio2, MAX2(opcio1,opcio3));

                  r1 = matriu[c + j+1];
                  opcio1 = r3 + match2;
                  opcio2 = resultat1 + gap;
                  opcio3 = r1 + gap;
                  r2 = MAX2 (opcio2, MAX2(opcio1,opcio3));

                  matriu[c2 + j] = resultat1;
                  matriu[c2 + j+1] = r2;

            }
      }
      Sample_OFF();

// Paso 3: Traceback

      int x,y;
      int puntuacio,puntuaciod,puntuaciou,puntuaciol,pos;
      x=M;
      y=N;

      if((M+N) > Talinear)
      {

            int incremento2 = floor((2 * (M+N))*0.5);
            int vmalloc2 = ((2* (M+N)) + incremento2);
            Talinear= (N+M) + incremento2;
            free(alin);
            alin = malloc ( sizeof(char) * vmalloc2);
      }

      pos= 0;
      for(i=0;i<2;i++){

            for(j=0;j<M+N;j++){

                  alin[i*(M+N) + j]=' ';
            }
      }
```

```
	while(y > 0 && x > 0)
	{
		puntuacio = matriu[x*(N+1) + y];
		puntuaciod = matriu[((x-1)*(N+1)) + y-1];
		puntuaciou = matriu [((x-1)*(N+1)) + y];
		puntuaciol = matriu [(x*(N+1)) + y-1];

		if ( puntuacio == puntuaciod + MATCHING(s1[y-1],s2[x-1]) )
		{

			alin[0*(M+N) + pos] = s1[y-1];
			alin[1*(M+N) + pos] = s2[x-1];
			x--;
			y--;
			pos++;

		}
		else if (puntuacio == puntuaciol + gap)
		{
			alin[0*(M+N) + pos] = s1[y-1];
			alin[1*(M+N) + pos] = '-';
			y--;
			pos++;
		}
		else if ( puntuacio == puntuaciou + gap)
		{
			alin[0*(M+N) + pos] = '-';
			alin[1*(M+N) + pos] = s2[x-1];
			x--;
			pos++;
		}

	}

	if(y > 0 && matriu[x*(N+1) + y]==0 )
	{
		while(y > 0)
		{
			alin[0*(M+N) + pos] = s1[y-1];
			alin[1*(M+N) + pos] = '-';
			y--;
			pos++;
		}
	}

	if(x > 0 && matriu[x*(N+1) + y]==0 )
	{
		while(x > 0)
		{
			alin[0*(M+N) + pos] = '-';
			alin[1*(M+N) + pos] = s2[x-1];;
			x--;
			pos++;
		}
	}

	int contador=0;
	FILE *f;
	f=fopen("output.txt","a+");
	fprintf(f,"\n");
	fwrite(alin,sizeof(char)*(N+M),2,f);
	fprintf(f,"\n");
	fclose(f);
}
```

ANEXO 3:

CÓDIGO VERSIÓN PARALELA VARIANTE FINE-GRAINED

- Función *multiplealign* : encargada de leer el fichero que contiene las secuencias, guardarlas en memoria y posteriormente irle pasando un par de secuencias a la función *alinear*, que es la encargada de realizar un alineamiento de 2 secuencias basándose en el algoritmo Needleman-Wunsch.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <omp.h>
#include "alinear.c"
#include "sample.h"

void EliminarMemoria(char *memo)
{

	free(memo);
}


void FAlignment(char *memo, int tamany, int mida, int numthreads,int T)
{
	int i,j;
	int s1,s2;
	float s;
	char *seq1, *seq2, *inicio;

	for (i=0; i < (tamany*tamany); i++)
	{
		s1 = i/tamany;
		s2 = i%tamany;

		j = 0;
		inicio = memo;

		if(*inicio != '\n')
		{
			seq1 = memo + ((mida+1)*s1) + j;
			seq2 = memo + ((mida+1)*s2) + j;
			j++;
			inicio++;
		}

		alinear(seq1,seq2,numthreads,T);

	}

}

int main(int argc, char** argv)
{

	FILE *reg = NULL;
	int mida,nseqs,t,numthreads;
	int i=0;
	int T=0;
	char *p;
	struct timespec t1,t2;

	numthreads = atoi(argv[1]);
	reg = fopen(argv[2], "r");
	T = atoi(argv[3]);

	fscanf(reg,"%d",&nseqs);
```

```
	fseek(reg,1,SEEK_CUR);

	fscanf(reg,"%d",&mida);
	fseek(reg,1,SEEK_CUR);
	mida= mida +2;

	char buffer[mida];
	char *memo;

	memo = malloc(sizeof(char)*(mida+1)*nseqs);

	  Sample_Init(0, 0);

	while(fgets(buffer,mida,reg)!=NULL)
	{
		memcpy(memo+((mida+1)*i),buffer,mida);
		i++;
	}

	FAlignment(memo,nseqs,mida,numthreads,T);

	Sample_End();

	EliminarMemoria(memo);
}
```

- Función *alinear* : recibe 2 secuencias como parámetros y realiza un alineamiento de éstas 2 secuencias basándose en el algoritmo de alineamiento de secuencias Needleman-Wunsch.

```
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <stdlib.h>
#include <math.h>

#include "sample.h"

#define MAX2(A, B) (((A) > (B)) ? (A) : (B))
#define MIN(A, B) (((A) > (B)) ? (B) : (A))
#define MATCHING(A,B) (((A) == (B)) ? (2) : (-1))
#define gap -2

int Tactual= 0;
int Talinear= 0;
int *matriu= NULL;
char *alin= NULL;


void alinear(char *s1, char *s2, int numthreads, int T)
{

      int i,j,k;
      int N,M;
      int match;

      struct timespec t1,t2;

//Paso 1 : Inicialización

      N=strlen(s1)-1;
      M=strlen(s2)-1;

      if( ((N+1)*(M+1)) > Tactual)
      {
            int incremento = floor(((N+1)*(M+1))*0.5);
            int vmalloc = (((M+1)*(N+1)) + incremento);
            Tactual = ((N+1)*(M+1)) + incremento ;
            free(matriu);
            matriu = malloc ( sizeof(int) * vmalloc);
      }


      for ( j=0 ; j<N+1 ; j++ )
      {

            matriu[j]=0;

      }



      for ( i=0 ; i<M+1 ; i++ )
      {
            matriu[i*(N+1)] = 0;

      }
```

```
      #pragma omp parallel
      {
        omp_set_num_threads(numthreads);
      }

//Paso 2 : Calculo de la Matriz de Needleman-Wunsch

      int particion;
      int flag = 0;
      int tparticion;
      float numerador = N;
      float denominador = numthreads;
      float resultat = numerador / denominador;
      particion=(int)ceil(resultat);
      tparticion = particion * 4;

      if(tparticion > 64 ) flag=1;

      if(flag ==0)
      {
            while( (64 % tparticion)!=0)
            {
                  particion++;
                  tparticion = particion * 4;
            }
      }
      else
      {
            while( (tparticion % 64) !=0)
            {
                  particion ++;
                  tparticion = particion * 4;
            }
      }


      Sample_ON();

      for( k = 2; k <= (numthreads+(M/T))*T; k=k+T )
      {

            int lsup = MIN(k-1,M-(T-1));
            int linf = MAX2(1,k-(T*(numthreads-1)+1));

            #pragma omp parallel for default (shared) private(i)
            schedule(static)

            for(i = lsup; i >= linf; i=i-T)
            {

                        int opcio1,opcio2,opcio3;
                        int j;
                        int c, c2, s2char, match1, match2, r1,r2,r3,resultat1,
                        resultat2;


                  for( j=0; j < T;j++)
                  {
                        c= (i-1)*(N+1)+(N+1)*j;
                        c2 = i*(N+1)+(N+1)*j;
                        s2char = s2[(i-1)+j];

                        int p = particion *((k-i)/T)+1;
                        int pfinal = MIN(p+particion,N);
```

```
                        r1=matriu[p+c-1];
                        r2=matriu[p+c2-1];

                        for(;p<pfinal;p=p+2)
                        {

                              r3 = matriu[p+c];
                              match1 = MATCHING(s1[p-1],s2char);
                              match2 = MATCHING(s1[p],s2char);
                              opcio1 = r1 + match1;
                              opcio2 = r2 + gap;
                              opcio3 = r3 + gap;
                              resultat1 = MAX2(opcio1, MAX2(opcio2,opcio3));

                              r1 = matriu[p+c +1];
                              opcio1 = r3 + match2;
                              opcio2 = resultat1 + gap;
                              opcio3 = r1 + gap;

                              r2 = MAX2 (opcio1, MAX2(opcio2,opcio3));

                              matriu[p+c2] = resultat1;
                              matriu[p+c2+1] = r2;

                        }
                  }

            }
      }
      Sample_OFF();

// Paso 3: Traceback

      int x,y;
      int puntuacio,puntuaciod,puntuaciou,puntuaciol,pos;
      x=M;
      y=N;

      if((M+N) > Talinear)
      {
            int incremento2 = floor((2 * (M+N))*0.5);
            int vmalloc2 = ((2* (M+N)) + incremento2);
            Talinear= (N+M) + incremento2;
            free(alin);
            alin = malloc ( sizeof(char) * vmalloc2);
      }

      pos= 0;


      for(i=0;i<2;i++){

            for(j=0;j<M+N;j++){

                  alin[i*(M+N) + j]=' ';
            }
      }

      while(y > 0 && x > 0)
      {
            puntuacio = matriu[x*(N+1) + y];
            puntuaciod = matriu[((x-1)*(N+1)) + y-1];
            puntuaciou = matriu [((x-1)*(N+1)) + y];
            puntuaciol = matriu [(x*(N+1)) + y-1];
```

```
		if ( puntuacio == puntuaciod + MATCHING(s1[y-1],s2[x-1]) )
		{
			alin[0*(M+N) + pos] = s1[y-1];
			alin[1*(M+N) + pos] = s2[x-1];
			x--;
			y--;
			pos++;
		}
		else if (puntuacio == puntuaciol + gap)
		{
			alin[0*(M+N) + pos] = s1[y-1];
			alin[1*(M+N) + pos] = '-';
			y--;
			pos++;
		}
		else if ( puntuacio == puntuaciou + gap)
		{
			alin[0*(M+N) + pos] = '-';
			alin[1*(M+N) + pos] = s2[x-1];
			x--;
			pos++;
		}

	}

	if(y > 0 && matriu[x*(N+1) + y]==0 )
	{
		while(y > 0)
		{
			alin[0*(M+N) + pos] = s1[y-1];
			alin[1*(M+N) + pos] = '-';
			y--;
			pos++;
		}
	}

	if(x > 0 && matriu[x*(N+1) + y]==0 )
	{
		while(x > 0)
		{
			alin[0*(M+N) + pos] = '-';
			alin[1*(M+N) + pos] = s2[x-1];;
			x--;
			pos++;
		}
	}

	int contador=0;
	FILE *f;
	f=fopen("output.txt","a+");
	fprintf(f,"\n");
	fwrite(alin,sizeof(char)*(N+M),2,f);
	fprintf(f,"\n");
	fclose(f);

}
```