Modelo de un algoritmo eficiente para el alineamiento de secuencias biomoleculares basado en programación paralela

Wilson Cesar Callisaya Choquecota, Rosalia Callohuari Quispe

nosliwsys@gmail.com, liabiol@yahoo.es

Escuela Académica Profesional de Ingeniería en Informática y Sistemas – Facultad de Ingeniería - Universidad Nacional Jorge Basadre Grohmann, Perú
Av. Miraflores S/N (Ciudad Universitaria)
Tacna - Perú

Resumen: *En la actualidad, se ha producido un considerable esfuerzo para desarrollar algoritmos que comparan las secuencias de macromoléculas biológicas (proteínas, ADN y ARN), cuyo objetivo es detectar las relaciones evolutivas tanto estructurales como funcionales. Éste es el principal problema de la biología computacional. Estas tareas se llevan a cabo actualmente por las herramientas de la bioinformática que han sido desarrollados con algoritmos secuenciales. La programación dinámica, tanto los algoritmos de alineamiento locales (Smith-Waterman) como de alineamiento global (Needleman-Wunsch) determinan el alineamiento óptimo de dos secuencias. Actualmente los ordenadores que tienen más de un núcleo están disponibles para el usuario común, y para usar los múltiples procesadores del ordenador es necesario conocer los paradigmas de programación paralela. Este trabajo presenta una nueva propuesta algorítmica para el alineamiento global usando la programación paralela. Esto requiere de una nueva reformulación del algoritmo de Needleman Wunsh. La implementación del Algoritmo Paralelo ha requerido hacer un llenado de la matriz de scores por sus antidiagonales con todos los procesadores disponibles. El software utilizado para ello fue el C# con la librería TPL ("Task Parallel Library"). La aplicación compara el algoritmo de Needleman-Wunsch con este nuevo algoritmo, comprobando los tiempos de respuesta. Los resultados muestran que el algoritmo paralelo propuesto reduce el tiempo de respuesta en comparación con el algoritmo de alineamiento global de Needleman-Wunsch.*

Palabras clave: Biología Computacional, Alineamiento Global, Needleman-Wunsh, Programación Paralela, Bioinformática.

Abstract: *At present there has been a considerable effort to develop algorithms that compare the sequences of biological macromolecules (proteins, DNA and RNA), which aims to detect evolutionary relationships both structural and functional. This is the main problem of computational biology. These tasks are currently performed bioinformatics tools that have been developed with sequential algorithms. Dynamic programming, both local alignment algorithms (Smith-Waterman) and global alignment (Needleman-Wunsch) determining optimal alignment of two sequences. Currently the computers that have more one core are available for the common user, and to use multiple computer processors need to know parallel programming paradigms. This paper presents a new algorithmic proposed for global alignment using parallel programming, this requires a new reformulation of the algorithm of Needleman Wunsh. The implementation of parallel algorithm has required to make a matrix filled with scores for their antidiagonales with all available processors. The software used for this was the C # with the library TPL (Task Parallel Library). The application compares Needleman-Wunsch algorithm with this new algorithm, checking the response time. The results show that the proposed parallel algorithm reduces the response time in comparison with the global alignment algorithm of Needleman-Wunsch.*

Keywords: Computational Biology, Global Alignment, Needleman-Wunsh, Parallel Programming, Bioinformatics.

1 Introducción

Uno de los principales problemas de la Biología Computacional es el de alineamiento de secuencias biomoleculares (ADN, ARN o secuencias de aminoácidos), ya que la similitud de 2 secuencias implica similitud funcional o estructural significativa [1].

Los métodos de alineamiento de secuencias más difundidos son: Análisis de matriz de puntos, algoritmo de programación dinámica, y el método Word o K-Tupla; métodos usados por los programas FASTA y BLAST [2]. Estos últimos métodos usan técnicas heurísticas para su desarrollo, obteniéndose un resultado probabilístico, el cual es muy cercano al verdadero. En el caso de la programación dinámica, su desarrollo hace posible encontrar el resultado exacto al buscar por todos los alineamientos existentes.

Varias investigaciones se han realizado para ayudar a resolver este problema eficientemente, pero poco se ha intentado usando el Paradigma Paralelo, esto debido a los costes que implicaba tener un ordenador paralelo y a su dificil implementación dado que se debía dar importancia a la comunicación entre los procesadores, pero en la actualidad ya existen librerías que nos apoyan a desarrollar en Paralelo, tales como el OPENMP disponible para C++, y Visual Studio con el Framework 4.0 nos ofrece la biblioteca procesamiento Paralelo basado en tareas TPL, dando la oportunidad a enfocarse solo en el problema de fondo.

Este trabajo es un producto transdisciplinario desarrollado por profesionales en Informática y Biología. Presenta una propuesta algorítmica para apoyar a la solución del alineamiento de secuencias usando la programación paralela, enfocándose en el llenado de la matriz de scores

(matriz de puntajes) descrito en la Sección 2.

El resto de este trabajo está organizado de la siguiente manera. En la Sección 2, se explica cómo es el procedimiento para realizar el alineamiento de secuencias con Programación Dinámica y se detalla cómo se mide el Factor de SpeedUp en Programación Paralela. La Sección 3 muestra los trabajos previos. La Sección 4 detalla la propuesta algorítmica para solucionar el problema usando el Paradigma Paralelo. La Sección 5 muestra los experimentos y resultados de la aplicación comparándolo con la implementación clásica del algoritmo de Programación Dinámica para alineamiento de secuencias y Resultados. Finalmente, se dan las conclusiones en la Sección 6.

2 Teoría del dominio

Para alinear las secuencias con programación dinámica se debe definir un valor para el "match" (*coincidencia*), "missmatch" (*no coincidencia*) o seleccionar una matriz de sustitución y el "gap" (puntaje cuando ocurre un "*Indel"*). El proceso se realiza en 3 pasos:

Paso 1 Inicialización: Ambas secuencias se ubican en una matriz F de m x n (m y n longitudes de ambas secuencias), luego el valor de la posición F(0,0) = 0 y se llena tanto la primera fila y la primera columna con múltiplos del valor del "gap", como lo describe el ejemplo en la Figura 1. Para el caso del Alineamiento Local tanto la primera fila y columna se llenan con 0.

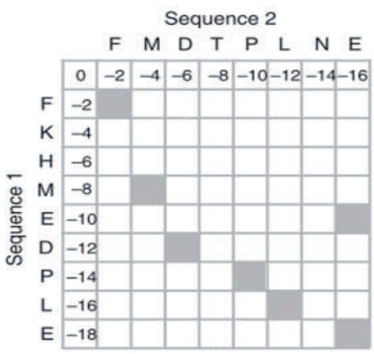

Figura 1. Inicialización con "gap".

Paso 2 Llenado de la matriz de scores (matriz de puntajes): Se procede a llenar todos los valores de la matriz según la función descrita en la Figura 2, si es para un Alineamiento Global y con la función descrita en la Figura 3 para un Alineamiento Local. El ejemplo de la Figura 4 describe el proceso del alineamiento global.

$$F(i,j) = \max \begin{cases} F(i-1,j-1) + s(x_i, y_j), \\ F(i-1,j) - d, \\ F(i,j-1) - d. \end{cases}$$

Figura 2. Función del Alineamiento Global

$$F(i,j) = \max \begin{cases} 0, \\ F(i-1,j-1) + s(x_i, y_j), \\ F(i-1,j) - d, \\ F(i,j-1) - d. \end{cases}$$

Figura 3. Función del Alineamiento Local.

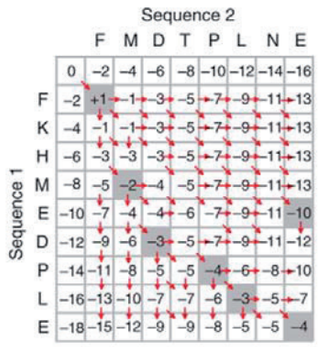

Figura 4. Llenado de la matriz de scores.

Paso 3 Identificación del alineamiento – Traceback: Este paso diverge según el tipo de alineamiento. En el caso de Alineamiento Global, inicia siempre en la posición (m,n) en el cual está el score (puntaje) del mejor alineamiento y se hace un recorrido hacia atrás para identificar el alineamiento, como se describe en la Figura 5. En el caso del Alineamiento Local inicia en el mayor valor de la matriz de scores hasta llegar a un valor 0 [3][4].

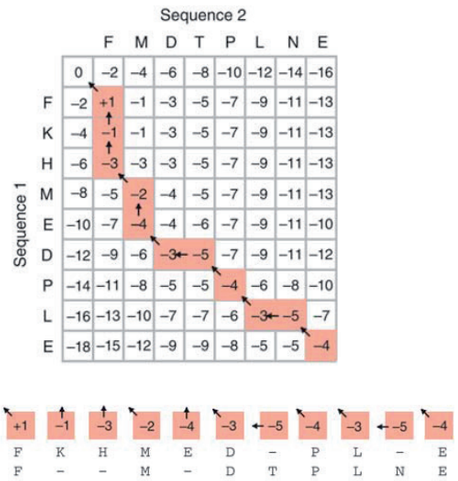

Figura 5. Traceback (recorrido hacia atrás).

En la actualidad, se pueden encontrar herramientas bioinformáticas que nos ayudan para este propósito, como por ejemplo el Needle, Stretcher para Alineamiento Global y Water, Matcher, LALIGN para Alineamiento Local.[5]. Estas herramientas han sido diseñadas con algoritmos secuenciales pudiéndose usar el procesamiento paralelo.

El propósito principal del procesamiento paralelo es realizar cálculos con menores tiempos de respuesta de los que se puede hacer en un ordenador con un único procesador, mediante el uso de varios procesadores al mismo tiempo. Algunos diseños informáticos permiten a un único procesador ejecutar varias secuencias de instrucciones de una manera intercalada con la programación concurrente, pero en el caso de la programación paralela cada uno de estos hilos se ejecuta en simultáneo.

El beneficio potencial de la computación en paralelo se mide típicamente por el tiempo que se necesita para completar una tarea en un único procesador, en comparación con el tiempo que se necesita para completar la misma tarea en N procesadores en paralelo. El aumento de velocidad S(N) debido a la utilización de procesadores paralelos N, llamado Factor de SpeedUp, se define en la ecuación (1).

En (1), el valor de Tp (1) es el tiempo de procesamiento de algoritmo en un único procesador y Tp (N) es el tiempo de procesamiento de los procesadores paralelos [6].

$$S(N) = \frac{T_p(1)}{T_p(N)} \qquad (1)$$

3 Trabajos previos

Hay diversos trabajos que tratan de incrementar el tiempo de respuesta alterando el algoritmo inicial de Needleman-Wunsh y el de Smith–Waterman, como el de Shehab, Keshk, & Mahgoub que pretende incrementar el tiempo de respuesta del paso 3 del algoritmo de Alineamiento de Secuencias, pero no deja de ser este una propuesta secuencial. [7]

La propuesta de una solución a este problema en forma paralela ha sido un reto para los investigadores del área de Biología Computacional. Se presentaron diversas propuestas para intentar hacer este análisis e incrementar el tiempo de respuesta. Una de las primeras menciones realizadas a la solución de éste y muchos otros problemas de la Biología Computacional se muestra en el libro de Zomaya en el 2006[8]. En el llenado de la matriz de scores, se puede observar que cada una de las celdas tiene una fuerte dependencia con las tres anteriores, es por ello que se considera una Paralelización de Grano fino. La estrategia usada para apoyar en el alineamiento de secuencias con programación dinámica usando la programación paralela, es realizar el llenado de la matriz de scores de antidiagonal en antidiagonal, pudiendo ser este trabajo distribuido en varios núcleos, como se indica en la Figura 6.

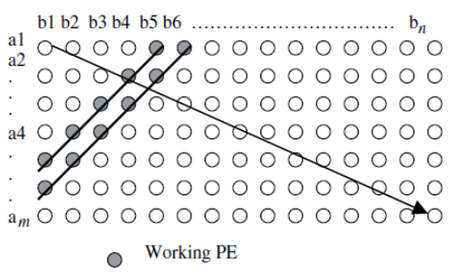

Figura 6. Barrido de antidiagonal en antidiagonal.

Propuesta similar se observa en el trabajo de Nawaz, Nadeem, Someren, & Bertels en 2010 para abordar el problema usando el procesamiento paralelo utilizando con un circuito integrado configurable, una FPGA (“Field Programmable Gate Array”) [9], en el cual también refiere al llenado de antidiagonal en antidiagonal, y propone realizarlo por bloques en cada ciclo como se muestra en la Figura 7.

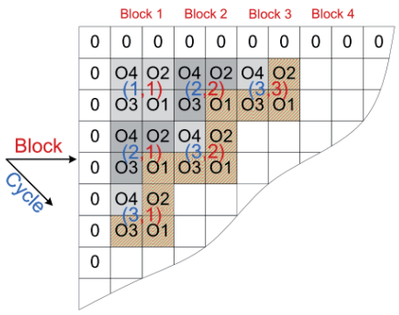

Figura 7. Llenado en Bloques.

Propuesta similar se realiza esta vez usando la GPU y CUDA utilizando los multiprocesadores de la tarjeta de video.[10]

Como se puede apreciar, estas ideas han sido desarrolladas con hardware especializado, pero en la actualidad contamos con librerías que nos pueden ayudar a resolver este problema, y hardware de propósito general con múltiples núcleos. No hay difusión de un algoritmo que realice tal recorrido de antidiagonal en antidiagonal para un proceso paralelo. La propuesta algorítmica se describe en la siguiente sección.

4 Diseño de algoritmos propuestos

Al existir una fuerte dependencia en el llenado de la matriz de scores, se tuvo que reformular el algoritmo para el llenado de antidiagonal en antidiagonal. Observemos cómo es el comportamiento de una matriz no cuadrada. En la Figura 8, se muestra una matriz de 5 filas y 7 columnas, con sus posiciones respectivas. Podemos observar que si recorremos las antidiagonales de la posición superior a la inferior notamos que mientras la posición de la fila i aumenta la posición de la columna j disminuye.

(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)
(2,1)	(2,2)	(2,3)	(2,4)	(2,5)	(2,6)	(2,7)
(3,1)	(3,2)	(3,3)	(3,4)	(3,5)	(3,6)	(3,7)
(4,1)	(4,2)	(4,3)	(4,4)	(4,5)	(4,6)	(4,7)
(5,1)	(5,2)	(5,3)	(5,4)	(5,5)	(5,6)	(5,7)

Figura 8. Matriz con la posición de sus celdas.

Esta idea se puede implementar con un procesador secuencial, pero al implementarlo en paralelo se podría encontrar grandes dificultades, puesto que, al usar un parallel.for se observa que los procesadores no tienen un orden específico al desarrollar una tarea, por ejemplo, si para decrementar el valor de la columna hemos introducido la instrucción j=j-1 dentro del bucle paralelo nos daría valores incongruentes porque cuando un procesador se ubique en una i-ésima fila, para calcular el valor j necesitara de un j anterior, pero este valor pudo haber sido alterado por otro procesador dando un resultado no deseado.

4.1 Idea para la paralelización del llenado de la matriz de scores

Si con una sola variable pudiéramos obtener ambos parámetros: La posición de la fila y la columna, para cualquier posición se podría distribuir el trabajo, para lograr ello primero se realizó un nuevo análisis de la matriz pero esta vez concatenando la posición i y la posición j como se muestra en la Figura 9.

11	12	13	14	15	16	17
21	22	23	24	25	26	27
31	32	33	34	35	36	37
41	42	43	44	45	46	47
51	52	53	54	55	56	57

Figura 9. Matriz con las posiciones i y j concatenadas.

Al analizar ahora cada antidiagonal notamos que se comportan como una progresión aritmética de razón 9, ahora con solo el valor de cualquiera de estas celdas se puede obtener su posición tanto para la fila i como para la columna j, basta solo dividir el número entre 10, el cociente será el valor de la fila y el residuo el valor de la columna. Pero aún existen problemas para esta idea tal es el caso que se muestra en la Figura 10.

11	12	13	14	15	16	17	18	19	110	111	112
21	22	23	24	25	26	27	28	29	210	211	212
31	32	33	34	35	36	37	38	39	310	311	312
41	42	43	44	45	46	47	48	49	410	411	412
51	52	53	54	55	56	57	58	59	510	511	512
61	62	63	64	65	66	67	68	69	610	611	612
71	72	73	74	75	76	77	78	79	710	711	712

Figura 10. Matriz con las posiciones i y j concatenadas, en amarillo donde se empieza a no cumplirse el criterio detallado.

Considerando esta situación y futuras situaciones se evalúa la longitud de la cadena más larga, definimos un N que es igual longitud de la cadena más larga más complemento aritmético (CA), y de acuerdo a ello se multiplica N al valor de la posición de la fila i y posterior a ello se suma con j, al realizar esto se puede observar que toda antidiagonal se comporta como una progresión aritmética de razón N-1, en la Figura 11 se observa esta situación.

Bastará conocer el inicio de la antidiagonal para poder calcular cualquier valor de la antidiagonal con la ecuación (2):

$$VCA = VIA+(N-1)*(i-1) \quad (2)$$

Siendo:

VCA = El valor de la celda actual

VIA = El valor del inicio de la antidiagonal

i = Posición i-ésima de la antidiagonal.

N = lsc + CA (lsc)

lsc = longitud de la secuencia más larga

El ejemplo de uso de (2) se puede observar en la Figura 11, asumamos que se está usando 3 procesadores entonces en un instante de tiempo se pueden llenar 3 valores de la matriz de scores, por ejemplo para el valor de VIA=312, con N=100, los valores de su antidiagonal dependerán de i, entonces si consideramos un i=5 se obtendrá:

VCA = 312+ (100-1)*(5-1)

VCA = 708

106	107	108	109	110	111	112	
206	207	208	209	210	211	212	
306	307	308	309	310	311	312	-->1
406	407	408	409	410			-->2
506	507	508	509	510			-->3
606	607	608					-->4
706	707	708					-->5

Figura 11. Matriz de scores con posiciones operadas en un instante de tiempo por 3 procesadores.

Para obtener el valor de la fila y la columna se dividirá el valor de VCA entre el valor de N, siendo su cociente el valor de la fila y el residuo el valor de la columna. Cabe resaltar que es importante que el bloque que está en la zona crítica donde todos los procesadores van a escribir sus variables deban ser protegidas, en su defecto darían valores sin sentido.

Se debe considerar que el tamaño de la antidiagonal va incrementándose hasta llegar a la longitud de la secuencia más corta, luego se mantendrá el tamaño de esa antidiagonal hasta llegar a la antidiagonal que coincida con el tamaño de la longitud de la secuencia más larga y a partir de ahí se reducirá hasta llegar a la última antidiagonal, estas tres situaciones se muestran en la Figura 12.

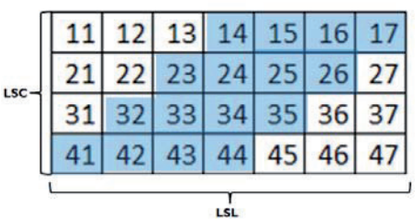

Figura 12. Matriz de scores con posiciones operadas en un instante de tiempo por 3 procesadores.

4.2 Elaboración de los Algoritmos para la solución propuesta

La obtención de la fila y la columna, junto con la operación de la ecuación (2) se observa en la función Almacenar descrita en el algoritmo 1.

En el algoritmo 1 se puede apreciar X y Y son respectivamente las posiciones de la matriz recuperadas de aplicar la ecuación (2), f es una matriz que almacena todos los valores de la matriz de scores, "max" es una función que recupera el mayor de 3 numeros, "s" es la función de similitud de ambas secuencias, "gap" es la penalidad asignada a los huecos.

Algoritmo 1 Almacenar(i, VIA, N)

Requiere:

La función max (Máximo de 3 números).

La función de similitud S.

El llenado de la primera fila y la columna de la matriz de scores.

El puntaje del gap

Asegurar:

VCA<=VIA+(N-1)x(i)

X <= VCA/N

Y <= VCA mod N

f(X,Y)<=max(f(X-1,Y)+gap, f(X-1,Y-1)+s(X,Y),f(X,Y-1)+ gap)

Para la comparación, se implementó el paso 2 (llenado de la matriz de scores) del Algoritmo de Alineamiento de Secuencias con Programación Dinámica, como lo describe el algoritmo 2.[11]

Algoritmo 2 Paso 2 para alineamiento de secuencias con Programación Dinámica

Requiere:

La función *max* (Máximo de 3 números).

La función de similitud *S*.

El puntaje del gap

Las longitudes de dos cadenas lsc (longitud de la secuencia más corta) y lsl (longitud de la secuencia más larga)

Asegurar:

Para i<=0 **Hasta** lsc **Hacer**

f(i,0)=i x gap

Fin Para

Para i<=0 **Hasta** lsl **Hacer**

f(0,i)=i x gap

Fin Para

Para i<=1 **Hasta** lsc **Hacer**

Para j<=1 **Hasta** lsl **Hacer**

f(i,j)=max(f(i-1,j)+gap, f(i-1,j-1)+s(i,j), f(i,j-1)+gap)

Fin Para

Fin Para

Escribir "El puntaje máximo se encuentra en:" f(lsc,lsl)

Este paso ha sido readaptado para que sea fácilmente paralelizable con la idea descrita en la Subsección 4.1, para lo cual se elaboró el algoritmo 3, en la cual se usará la instrucción parallel.for (descrita en pseudocódigo como *Hacer en Paralelo*). Esta instrucción es similar a la instrucción for (descrita en pseudocódigo como Hacer) con la diferencia que esta distribuye el recorrido del bucle a los diferentes procesadores disponibles del ordenador.

Se requerirá, para usar el algoritmo 3, un pre-procesamiento que es el desarrollo del paso 1 del Algoritmo de Alineamiento de Secuencias con Programación Dinámica. A continuación, se describe, en detalle, las variables usadas en el algoritmo 3:

da = Representa a la antidiagonal Actual

lsc = Longitud de la secuencia más corta

lsl = Longitud de la secuencia más larga

tda = Tamaño de la antidiagonal actual

centi = Centinela para identificar cuando se llega al tamaño de la antidiagonal mayor.

i = posición i-ésima de la antidiagonal.

VIA = Valor inicial de la antidiagonal inicializando en N+1

N = lsl + CA(lsl)

Algoritmo 3 Llenado de la Matriz de Scores usando varios procesadores.

Requiere:

La función *Almacenar*

Las longitudes de dos cadenas lsc (longitud de la secuencia más corta) y lsl (longitud de la secuencia más larga)

El valor de N

Asegurar:

VIA<=N+1

centi<=0

Para da <=1 **Hasta** lsc + lsl -1 **Hacer**

Para i<=0 **Hasta** tda-1 **Hacer en Paralelo**

Almacenar(i,VIA,N)

Fin Para

Si tda < lsc **y** centi=0 **entonces**

tda<=tda+1

VIA <=VIA +1

Sino

centi<=1

Si da<lsl **entonces**

VIA=VIA+1

Sino

tda=tda-1

VIA=VIA+N

Fin si

Fin si

Fin Para

Escribir "El puntaje máximo se encuentra en:" f(lsc,lsl)

Como se apreció, es posible realizar la paralelización recorriendo sus antidiagonales, pero hay un coste de comunicación alto al finalizar cada antidiagonal, como se aprecia en el algoritmo 3. El número de veces que se usará la instrucción parallel.for (descrita en pseudocódigo como *Hacer en Paralelo*) será el número total de antidiagonales de la matriz de scores. En la siguiente subsección, se optimizará el algoritmo.

4.3 Optimización trabajando en bloques

Con el anterior algoritmo, cada procesador hace el llenado de una celda cualquiera dentro de una antidiagonal en una iteración que usa el parallel.for, en la optimización presente en lugar de ello cada procesador hará un llenado de todo un bloque horizontal de celdas en una iteración

que usa el parallel.for dentro de la matriz de scores. Para ello primero calcularemos el tamaño de cada bloque realizando una división por exceso entre la longitud de la secuencia más larga y el número de procesadores.

tb = divExceso(lsl,npn)

siendo :

tb = tamaño del bloque

npn = Número de procesadores necesarios

divExceso = Función de la división por exceso o equivalente a añadir la unidad al resultado de la división si existe residuo.

El objetivo de esto es lograr que el llenado de la matriz de scores del paso 2 del algoritmo de Programación Dinámica se realice en bloques de antidiagonales, como se observa en la Figura 13.

		A	A	U	G	C	C	A	U	U	G	A	C
	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12
C	-1												
A	-2												
G	-3												
C	-4												
C	-5												

Figura 13. Distribución de los bloques de antidiagonales de la matriz de scores para cuatro procesadores.

En la Figura 13, los colores idénticos nos indican los bloques de antidiagonales que realizarán la tarea en paralelo, en este caso la división fue exacta entre el número de procesadores considerados (para el ejemplo cuatro procesadores), como se aprecia también la lsl se la posiciona en forma horizontal. Existen casos en los cuales la división no será exacta por lo cual usaremos la división por exceso y completaremos la secuencia de lsl con caracteres adicionales no propias de la secuencia, por ejemplo, una X, como se aprecia en la Figura 14.

A	A	U	G	C	C	A	U	U	G	A	C	G	G	X	X
-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14	-15	-16
101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116
201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216
301	302	303	304	305	306	307	308	309	310	311	312	313	314	315	316
401	402	403	404	405	406	407	408	409	410	411	412	413	414	415	416
501	502	503	504	505	506	507	508	509	510	511	512	513	514	515	516
601	602	603	604	605	606	607	608	609	610	611	612	613	614	615	616

Figura 14. Distribución de los bloques de antidiagonales para la matriz de scores siendo una división inexacta.

El score seguirá ubicado en la posición f(lsc,lsl), en la Figura 15, se observa que un procesador tendrá necesariamente que realizar operaciones que no son parte de la matriz de scores verdadera, pero la cantidad de procesos puede ser despreciable, ya que las secuencias a analizar normalmente son entre 10 a 100 millones de caracteres (ADN, ARN y Proteínas).

Al realizar el proceso de esta forma, el número de veces que se usa el parallel.for es igual a longitud de la secuencia más corta (lsc) más el número de procesadores necesarios (npn) menos la unidad.

Para calcular el valor inicial de cada bloque, necesitaremos conocer el valor del inicio del Bloque Superior de la antidiagonal actual (VIBSA) y aplicar la ecuación:

VCA = VIBSA+(N-tb)*(i-1) (3)

Siendo:

VCA = El valor de la celda actual

VIBSA = El valor del inicio del Bloque Superior de la antidiagonal actual

N = lsl + CA(lsl)

lsl = Longitud de la secuencia más larga.

I = Posición i-ésima de la antidiagonal.

tb = Tamaño del bloque

La aplicación de (3) se puede observar en la Figura 16. En este ejemplo, podríamos asumir que se está usando 5 procesadores en un instante de tiempo, se pueden llenar 5 valores de la matriz de scores, por ejemplo para el valor de VIBSA=213, con N=100, el tb=3, los valores de su antidiagonal dependerán de i. Entonces, si asumimos un i=4, se obtendrá:

VCA = 213+ (100-3)*(4-1)

VCA = 504

Entonces, el llenado de un bloque se haría del valor de VCA hasta el valor de VCA más tb (tamaño de bloque) menos la unidad. Para obtener el valor de la fila y la columna se dividirá el valor de VCA entre el valor de N, siendo su cociente el valor de la fila y el residuo el valor de la columna.

		A	A	U	G	C	C	A	U	U	G	A	C	G	G	X	
	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14	-15	
C	-1	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	
A	-2	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	-->1
G	-3	301	302	303	304	305	306	307	308	309	310	311	312				-->2
C	-4	401	402	403	404	405	406	407	408	409							-->3
C	-5	501	502	503	504	505	506										-->4
U	-6	601	602	603													-->5

Figura 15. Proceso de llenado en bloques, en verde el valor inicial de cada bloque de la antidiagonal.

Hay que tener en cuenta que si el número de procesadores es mayor a la secuencia más corta se deberá usar el algoritmo de la subsección 4.1.

4.4. Elaboración de algoritmos para la optimización trabajando en bloques

La obtención de la fila y la columna, junto con la operación de la ecuación (3) se observa en la función AlmacenarB descrita en el algoritmo 4.

En el algoritmo 4, se puede apreciar X y Y que son respectivamente las posiciones de la matriz recuperadas de aplicar la ecuación (3). Se ha añadido un bucle a diferencia de la función Almacenar mostrado en la subsección anterior para recorrer el bloque actual.

Algoritmo 4 AlmacenarB(i, VIBSA, N)

Requiere:

La función max (Máximo de 3 números)..

La función de similitud S.

El llenado de la primera fila y la columna de la matriz de scores.

El puntaje del gap

El tamaño del bloque tb

Asegurar:

VCA<=VIBSA+(N-tb) x (i)

Para val<=VCA **Hasta** VCA + tb -1 **Hacer**

X <= val/N

Y <= val mod N

f(X,Y)<=max(f(X-1,Y)+gap, f(X-1,Y-1)+s(X,Y),f(X,Y-1)+ gap)

Fin Para

Se requeriría para usar el algoritmo 5 un pre-procesamiento como en el algoritmo 3. A su vez se obtiene el total de cifras de la secuencia más larga y se hace la añadidura de las "X" si fuera una secuencia de longitud no divisible entre el número de procesadores.

Algoritmo 5 Llenado de la Matriz de Scores usando varios procesadores en bloques

Requiere:

La función *Almacenar*

Las longitudes de dos cadenas lsc (longitud de la secuencia más corta) y lsl (longitud de la secuencia más larga)

El valor de N

Número de procesadores necesarios npn.

Tamaño de Bloque tb.

Asegurar:

VIBSA<=N+1

centi<=0

Para da <=1 **Hasta** lsc + npn - 1 **Hacer**

Para i<=0 **Hasta** tda-1 **Hacer en Paralelo**

AlmacenarB(i,VIBSA,N)

Fin Para

Si tda < npn **y** centi=0 **entonces**

tda<=tda+1

VIBSA <=VIBSA +tb

Sino

centi<=1

Si da<lsc **entonces**

VIBSA=VIBSA+N

Sino

tda=tda-1

VIBSA=VIBSA+N

Fin si

Fin si

Fin Para

Escribir "El puntaje máximo se encuentra en:" f(lsc,lsl)

Las variables usadas en el algoritmo 5 son:

da = Representa a la antidiagonal de bloques actual.

tda = Tamaño de la antidiagonal de bloques actual.

centi = Centinela para identificar cuando se llega al tamaño de la antidiagonal mayor.

i = posición i-ésima de la antidiagonal.

VIBSA = El valor del inicio del Bloque Superior de la antidiagonal actual. Inicializado en N+1.

N = lsl + CA(lsl).

5 Experimentos y resultados

La herramienta de software ha sido implementada en C# usando el *Visual Studio* y el *Framework 4.0*, ya que este *Framework* provee la librería TPL y esta librería nos da soporte para bucles paralelos, es decir, iteraciones que se realizan con un cierto grado de concurrencia. Para ello usaremos la clase *System.Threading.Task.Parallel* [12], ya que nos permitirá usar la instrucción Parallel.for.

Se implementa también en C# el algoritmo de Programación Dinámica para alineamiento de secuencias para realizar la comparación de tiempos respectiva.

Para realizar la medición de tiempos, se ha usado la clase *StopWatch* disponible en la librería "System.Diagnostic" de C#. Así podemos obtener un cronómetro de gran precisión.

El Experimento fue realizado en un ordenador de procesador "*Intel (R) Core (TM) i5-3210M de 2.5 Ghz*", el cual tiene 4 núcleos con memoria de 8Gb. En la Figura 16, se observa la interfaz de la aplicación final con los tiempos obtenidos usando *StopWatch*.

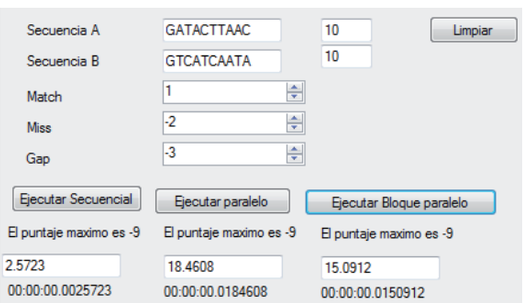

Figura 16. Interfaz de la aplicación de alineamiento de secuencias con los tiempos empleados.

En el experimento, se hace un estudio comparativo del paso 2 del algoritmo de programación dinámica clásico con los algoritmos paralelos implementados.

En el estudio del rendimiento de los algoritmos, se seleccionaron un total de 46 pares de secuencias obtenidas de la Base de Datos del GenBank [13], de tamaños aproximados entre 500 a 11500 pb. Para efectos del estudio, se seleccionaron secuencias de tamaños que no difieran en más de 500 pares de bases (pb) a los diferentes tamaños referenciales mostrados en la Tabla I, los Tiempos de ejecución en milisegundos (ms) obtenidos por el *StopWatch* del Procesamiento Secuencial y los Algoritmos Paralelos, así como el resultado del *SpeedUp* de los dos algoritmos paralelos propuestos en la Sección 4

se describen en la Tabla 1.

Tabla 1: Tiempos de Ejecución para los Procesamientos Secuencial y Paralelo.

Tamaño Referencial de las Secuencias (pb)	Tiempos de Ejecución (ms)			SpeedUp Paralelo sin Bloques	SpeedUp Paralelo en Bloques
	Secuencial	Paralelo sin Bloques	Paralelo en Bloques		
500	56.529775	69.42105	53.139175	0.8143031	1.063806
1000	197.0521	219.6907	159.3638	0.8969523	1.2364923
1500	576.3760	537.6962	407.2332	1.0719362	1.4153464
2000	970.8381	875.2200	727.5966	1.1092503	1.3343082
2500	1488.6238	1289.1062	982.8921	1.154772	1.5145343
3000	2141.0394	1640.1192	1368.7319	1.305417	1.5642504
3500	2539.8440	2034.0661	1642.3078	1.2486537	1.5465092
4000	3591.4311	2665.6282	2123.3163	1.3473114	1.6914254
4500	4726.1362	3213.7794	2654.5898	1.4705851	1.780364
5000	6490.0966	3797.4614	3133.6398	1.7090619	2.0711048
5500	7238.0394	4197.1809	3452.0844	1.7245002	2.0967156
6000	8216.0436	4542.9273	3691.6468	1.8085351	2.2255768
6500	9485.07555	5101.46473	4222.01398	1.8592847	2.2465761
7000	11299.3159	5860.0234	4829.4209	1.9282032	2.3396834
7500	13247.2562	6436.6551	5279.4590	2.0580963	2.5092072
8000	15360.1171	7290.6919	5855.7580	2.106812	2.6230792
8500	16712.3168	7744.8956	6406.9469	2.1578492	2.6084682
9000	18480.6478	8516.7067	6767.5236	2.1699289	2.7307844
9500	20911.3996	9442.8793	7559.5228	2.2145152	2.7662328
10000	23387.5019	10406.1922	8308.2868	2.2474601	2.8149608
10500	26183.7990	11610.8662	9007.0977	2.2551116	2.9070184
11000	29651.8950	12804.4591	9992.8335	2.3157476	2.967316
11500	31810.9826	13670.9042	10658.6492	2.3269114	2.9845229

Se puede apreciar que, para secuencias cortas, el procesamiento secuencial tiene un menor tiempo de ejecución, pero a medida que va incrementándose el tamaño de las secuencias el tiempo de ejecución del procesamiento paralelo es menor que el secuencial, como se aprecia en la Figura 17.

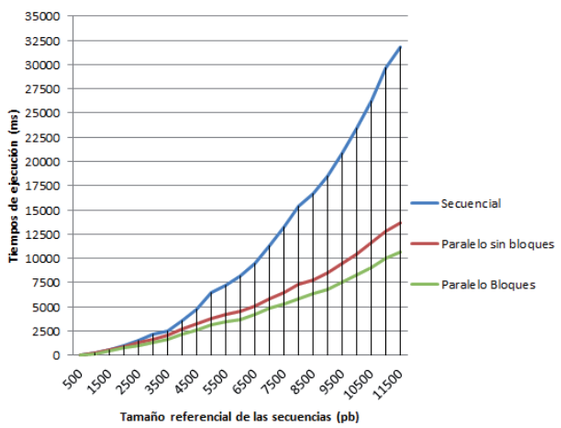

Figura 17. Tiempos de ejecución para el alineamiento de diferentes secuencias con los procesamientos Secuencial y Paralelo.

Se puede apreciar, en la Figura 18, que se obtiene un incremento del tiempo de respuesta con el procesamiento paralelo hasta el doble que el procesamiento secuencial. Esto se aprecia al evaluar el *SpeedUp* de ambos Algoritmos Paralelos presentados.

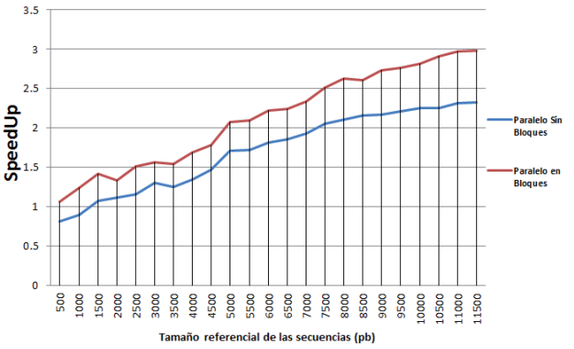

Figura 18. *SpeedUp* de la implementación paralela para los diferentes tamaños referenciales de secuencias.

6 Conclusiones y trabajos futuros

En este trabajo, se ha presentado una nueva implementación del algoritmo de Programación Dinámica para el alineamiento de secuencias reformulando el algoritmo en el paso 2 de la matriz de scores usando la Programación Paralela. Como muestran los resultados, ambas propuestas logran disminuir el tiempo de respuesta a medida que el tamaño de las secuencias se incrementa. El Tiempo de respuesta alcanzado hasta secuencias de tamaño 11500 llega a ser el doble que en el procesamiento secuencial.

Los resultados indican que el Algoritmo Paralelo en Bloques es un 20% más eficiente que la primera propuesta. Esta ventaja es debido a la gran cantidad de veces que se usa la instrucción parallel.for esta cantidad de veces es disminuida en la propuesta dada en bloques.

Como trabajo a futuro, se podría dar la aplicación del paradigma paralelo a diferentes ámbitos dentro de la Biología Computacional, como, por ejemplo, al alineamiento múltiple de Secuencias o a la construcción de árboles filogenéticos.

Referencias bibliográficas

[1] D. Gusfield, *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*, 1st editio. New York, United States, 1997, p. 534.

[2] D. Mount, *Bioinformatics: Sequence and Genome Analysis*. New York, United States, 2001, p. 564.

[3] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison, *Biological Sequence Analysis: Probabilistic models of proteins and nucleic acids*. New York, United States, 1998, p. 356.

[4] J. Pevsner, *Bioinformatics and Functional Genomics*, 2nd Editio. New York, United States, 2009, p. 897.

[5] The European Bioinformatics Institute EMBL-EBI, 2013 Http://www.ebi.ac.uk/Tools/psa/.

[6] F. Gebali, *Algorithms and Parallel Computing*. New Jersey, United States of America, 2011, p. 364.

[7] S. A. Shehab, A. Keshk, and H. Mahgoub, "Fast Dynamic Algorithm for Sequence Alignment based on Bioinformatics," *International Journal of*

Computer Applications, vol. 37, no. 7, pp. 54–61, 2012.

[8] A. Y. Zomaya, *Parallel Computing for Bioinformatics and Computational Biology*. New Jersey, United States of America, 2006, p. 1000.

[9] Z. Nawaz, M. Nadeem, H. Van Someren, and K. Bertels, "A parallel FPGA design of the Smith-Waterman traceback," in *Proc. International Conference on Field-Programmable Technology*, 2010, pp. 454–459.

[10] A. Khajeh-Saeed, S. Poole, and J. Blair Perot, "Acceleration of the Smith–Waterman algorithm using single and multiple graphics processors," *Journal of Computational Physics*, vol. 229, no. 11, pp. 4247–4258, Jun. 2010.

[11] J. Setubal and J. Meidanis, *Introduction to Computational Molecular Biology*. Boston, United States, 1997, p. 296.

[12] A. Freeman, *Pro .NET 4 Parallel Programming in C#*. Berkeley, CA: Apress, 2010, p. 328.

[13] The National Center for Biotechnology Information, 2013. http://www.ncbi.nlm.nih.gov/genbank/.