Ciencia & Desarrollo, Vol. 16, 21 (2) 55 - 64 (2017)
ISSN 2304 – 8891 Versión impresa
Depósito legal N° 2017-16148

PROPUESTA DE UN ALGORITMO PARALELO PARA EL PROCESO DE ALINEAMIENTO DE PARES DE SECUENCIAS BIOMOLECULARES

PROPOSAL FOR A PARALLEL ALGORITHM FOR THE BIOMOLECULAR SEQUENCE PAIR ALIGNMENT PROCESS

[1]Wilson Cesar Callisaya Choquecota, [2]Hugo Manuel Barraza Vizcarra

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. Este es el principal problema de la biología computacional, estas tareas se llevan a cabo, actualmente, por las herramientas de la bioinformática que han sido desarrolladas con algoritmos secuenciales. La programación dinámica, tanto los algoritmos de alineamiento locales (Smith-Waterman) y alineamiento global (Needleman-Wunsch) determinan el alineamiento óptimo de dos secuencias. Actualmente, las computadoras que tienen más de un núcleo están disponibles para el usuario común, y para usar los múltiples procesadores de la computadora, 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: Alineamiento global, Bioinformática, Biología Computacional, Needleman-Wunsh, Programación paralela.

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 anti-diagonals 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: Global Alignment, Bioinformatics, Computational Biology, Needleman-Wunsch, parallel Programming.

[1] Escuela de Posgrado, Universidad Nacional Jorge Basadre Grohmann. Tacna – Perú. Email: nosliwsys@gmail.com
[2] Escuela de Ingeniería en Informática y Sistemas, Universidad Nacional Jorge Basadre Grohmann. Tacna – Perú. Email: hmbarrazav@unjbg.edu.pe

Presentado: 27/02/17 Aprobado: 31/07/17

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 (Gusfield, 1997).

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 (Mount, 2001), 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 costos que implicaba tener un computador paralelo y a su dificil implementación dado, se tenía que dar importancia a la comunicación entre los procesadores. Sin embargo, en la actualidad ya existen librerias que apoyan a desarrollar en paralelo, tales como el OpenMP disponible para C++, y Visual Studio con en el Framework 4.0 que 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 (figura 4).

El resto de este trabajo está organizado de la siguiente manera. En Teoría del Dominio se explica el procedimiento para realizar el alineamiento de secuencias con programación dinámica y se detalla cómo se mide el Factor de *SpeedUp* en Programación Paralela. Seguidamente se muestra los trabajos previos y el diseño de Algoritmos Propuestos. Posteriormente se presentan los experimentos y resultados de la aplicación comparándolo con la implementación clásica del algoritmo de programación dinámica para alineamiento de secuencias y resultados. Para finalizar se dan las conclusiones y recomendaciones para trabajos futuros.

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.

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.

		Secuencia 2							
		F	M	D	T	P	L	N	E
	0	-2	-4	-6	-8	-10	-12	-14	-16
F	-2								
K	-4								
H	-6								
M	-8								
E	-10								
D	-12								
P	-14								
L	-16								
E	-18								

Secuencia 1

Figura 1. Inicialización con "*gap*"
Fuente: Elaboración propia

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

Figura 2. Función del Alineamiento Global
Fuente: Elaboración propia

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

Figura 3. Función del Alineamiento Local
Fuente: Elaboración propia

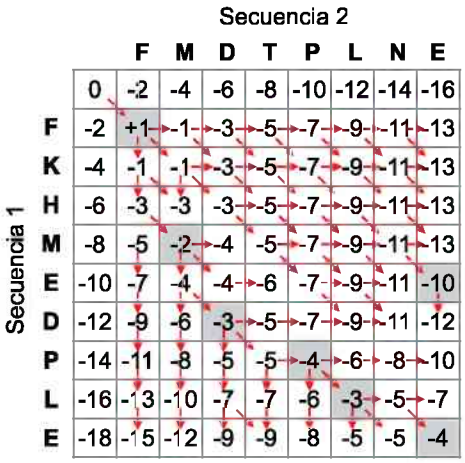

Figura 4. Llenado de la matriz de scores
Fuente: Elaboración propia

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 (Durbin *et al.*, 1998) (Pevsner, 2009).

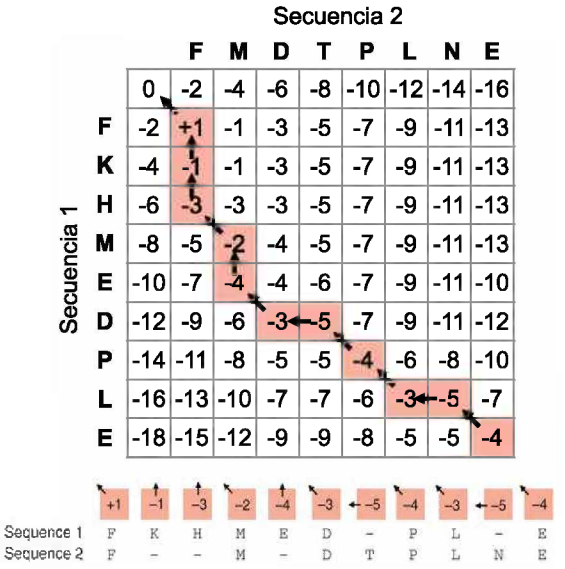

Figura 5. Traceback (recorrido hacia atrás)
Fuente: Elaboración propia

En la actualidad se pueden encontrar herramientas bioinformáticas que ayudan para este propósito como por ejemplo el Needle, Stretcher para alineamiento global y Water, Matcher, LALIGN para alineamiento local (The European Bioinformatics Institute EMBL-EBI, 2013). 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 computador 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 T_p (1) es el tiempo de procesamiento de algoritmo en un único procesador y T_p (N) es el tiempo de procesamiento de los procesadores paralelos (Gebali, 2011).

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

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 y 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 (Shehab *et al.* 2012).

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 este y muchos otros problemas de la Biología Computacional se muestra en el libro de Zomaya (Zomaya, 2006). 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.

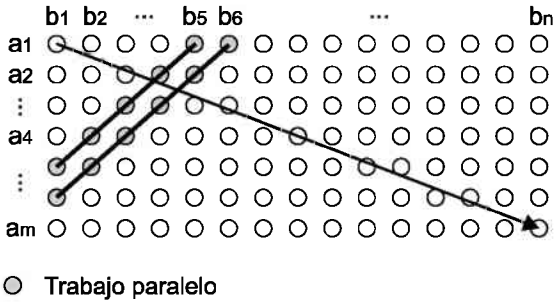

Figura 6. Barrido de antidiagonal en antidiagonal
Fuente: Elaboración propia

Propuesta similar se observa en el trabajo de Nawaz *et al.* (2010) para abordar el problema usando el procesamiento paralelo utilizando con un circuito integrado configurable, una FPGA ("*Field Programmable Gate Array*") (Nawaz *et al.*, 2010), 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.

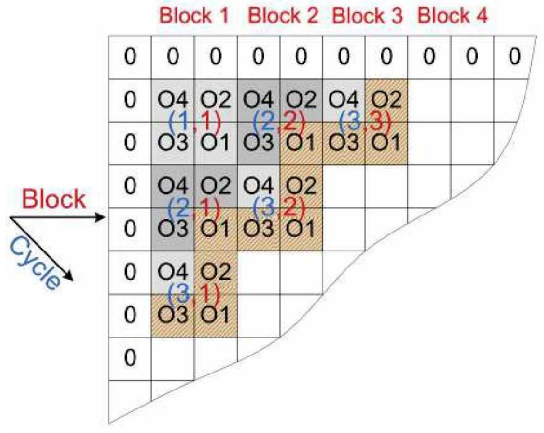

Figura 7. Llenado en Bloques
Fuente: Elaboración propia

Propuesta similar se realiza esta vez usando la GPU (*Graphics Processor Unit*) y CUDA (*Compute Unified Device Architecture*) utilizando los multiprocesadores de la tarjeta de video (Khajeh-Saeed *et al.*, 2010).

Como se puede apreciar, estas ideas han sido desarrolladas con hardware especializado pero en la actualidad contamos con librerías que 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.

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; se puede observar que si se recorre las antidiagonales de la posición superior a la inferior, se nota 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
Fuente: Elaboración propia

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 se ha introducido la instrucción j=j-1 dentro del bucle paralelo, daría valores incongruentes, porque cuando un procesador se ubique en una i-ésima fila para calcular el valor j necesitará de un j anterior; pero este valor pudo haber sido alterado por otro procesador dando un resultado no deseado.

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
Fuente: Elaboración propia

Al analizar ahora cada antidiagonal, se nota 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.
Fuente: Elaboración propia

Considerando esta situación y futuros escenarios, se evalúa la longitud de la cadena más larga; se define que N es igual a la 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 12, 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 se considera 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
Fuente: Elaboración propia

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.

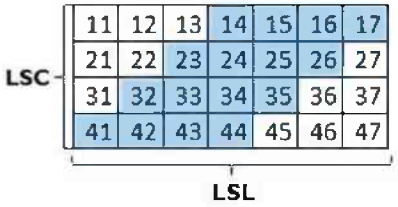

Figura 12. Matriz de scores con posiciones operadas en un instante de tiempo por 3 procesadores
Fuente: Elaboración propia

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 que 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 números, "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 dinamica como lo describe el algoritmo 2 (Setubal & Meidanis, 1997).

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)+g
   Fin Para
Fin Para
Escribir "El puntaje máximo se encuentra en:"
f(lsc,lsl)
```

Este paso ha sido readaptado para que sea fácilmente paralelizable; 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 computador.

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:

```
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)
```

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)

Como se apreció es posible realizar la paralelización recorriendo sus antidiagonales, pero hay un costo 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.

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 se calculará 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 : Numero 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
Fuente: Elaboración propia

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 le posiciona en forma horizontal, existen casos en los

cuales la división no será exacta; por lo cual, se usará 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
Fuente: Elaboración propia

El score seguirá ubicado en la posición $f(lsc, lsl)$. En la figura 14, 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 100 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 se necesitará conocer el valor del inicio del Bloque Superior de la antidiagonal actual (VIBSA) y aplicar la ecuación:

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

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 15. En este ejemplo, se podría 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 se asume 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.
Fuente: Elaboración propia

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 aprecia 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).

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 da soporte para bucles paralelos, es decir, iteraciones que se realizan con un cierto grado de concurrencia; para ello, se usa la clase *System.Threading.Task.Parallel* (Freeman, 2010), ya que permite 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 se puede obtener un cronómetro de gran precisión.

El Experimento fue realizado en un computador de procesador "Intel (R) Core (TM) i7-4771 de 3.5 Ghz", el cual tiene 8 núcleos, con memoria de 8Gb. En la figura 16 se observa la interfaz de la aplicación final con los tiempos obtenidos usando *StopWatch*.

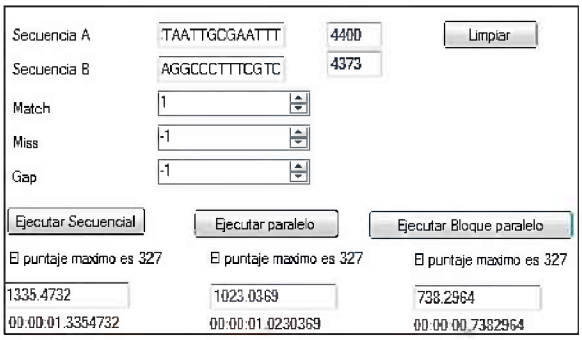

Figura 16. Interfaz de la aplicación de alineamiento de secuencias con los tiempos empleados
Fuente: Elaboración propia

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 60 pares de secuencias obtenidas de la base de datos del GenBank (The National Center for Biotechnology Information, 2013), de tamaños aproximados entre 400 a 12000 pb. Para efectos del estudio, se seleccionaron secuencias de tamaños que no difieran en más de 400 pares de bases (pb) a los diferentes tamaños referenciales mostrados en la tabla 1, 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 IV 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 en Bloques	Paralelo en Bloques		
400	7.6345	48.4126	22.4560	0.1573	0.3391
800	33.8420	43.8774	23.0776	0.7713	1.4664
1200	92.5139	84.6062	94.8874	1.0935	0.9750
1600	128.4414	163.3348	69.5710	0.7854	1.8462
2000	220.0160	223.0678	118.7799	0.9863	1.8525
2400	350.2218	305.6356	201.7435	1.1456	1.7355
2800	488.9285	360.6928	314.2978	1.3417	1.5397
3200	641.2658	504.6594	461.6450	1.2707	1.3891
3600	825.3279	700.5314	545.3521	1.1781	1.5139
4000	1025.6548	740.7942	658.3504	1.3845	1.5579
4400	1261.6876	974.3385	688.2793	1.2948	1.8331
4800	1486.6998	1004.0549	862.3963	1.4807	1.7245
5200	1656.3646	1038.4770	938.9873	1.5950	1.7640
5600	2004.2554	1289.0782	1031.0385	1.5548	1.9439
6000	2417.4324	1442.9822	1146.6902	1.6753	2.1082
6400	2656.6239	1657.0179	1338.0745	1.6033	1.9854
6800	3100.5804	1828.7607	1513.7795	1.6955	2.0482
7200	3505.9656	1883.9729	1573.5864	1.7548	2.1009
7600	3794.2351	2116.3246	1768.1329	1.7928	2.1459
8000	4128.6867	2351.4847	1892.0541	1.7558	2.1821
8400	4494.9350	2502.7684	2128.3266	1.7960	2.1120
8800	5154.6810	2861.1794	2404.4640	1.8016	2.1438
9200	5560.8041	3087.1790	2424.0842	1.8013	2.2940
9600	6133.2640	3309.1439	2699.9253	1.8534	2.2716
10000	6570.8878	3544.7961	2940.8088	1.8537	2.2344
10400	7269.1134	3871.2756	3115.9154	1.8777	2.3329
10800	7789.6954	4131.5046	3497.6190	1.8854	2.2271
11200	8282.1328	4380.2445	3624.7942	1.8908	2.2849
11600	8881.3759	4617.8285	3824.8078	1.9233	2.3220
12000	9494.8506	4979.7026	4044.6334	1.9067	2.3475

Fuente: Elaboración propia

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.

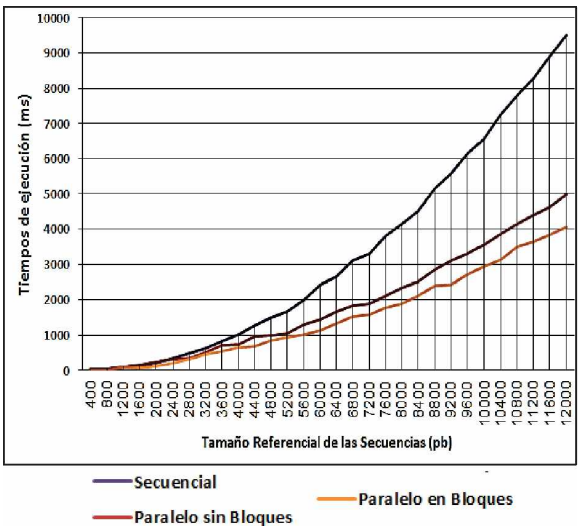

Figura 17. Tiempos de ejecución para el alineamiento de diferentes secuencias con los procesamientos Secuencial y Paralelo.
Fuente: Elaboración propia

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.

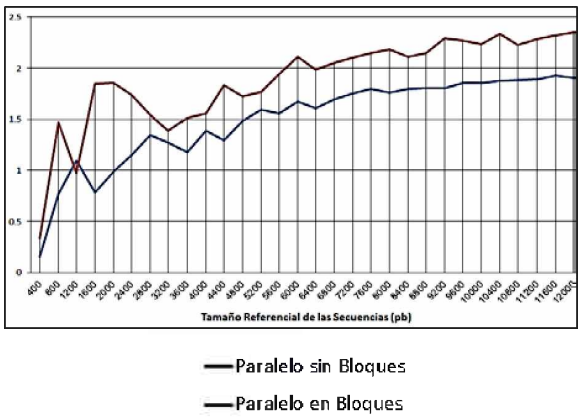

Figura 18. *SpeedUp* de la implementación paralela para los diferentes tamaños referenciales de secuencias.
Fuente: Elaboración propia

CONCLUSIONES

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 12000 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.

El procedimiento algorítmico propuesto difiere al presentado por de Nawaz, Nadeem, Someren, y Bertels en el 2010, dado que su trabajo fue diseñado para el paso 3 (traceback) del algoritmo de Smith Watterman, realizando su implementación en un hardware especializado como FPGA. El trabajo presentado reformula el paso 2 del algoritmo de programación dinámica para alineamiento de pares de secuencias de Needleman-Wunsch.

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

Durbin, R., Eddy, S., Krogh, A. y Mitchison, G. (1998). *Biological Sequence Analysis: Probabilistic models of proteins and nucleic acids.* New York.

Freeman, A. (2010). *Pro .NET 4 Parallel Programming in C#.* Berkeley: CA: Apress.

Gebali, F. (2011). *Algorithms and Parallel Computing.* New Jersey.

Gusfield, D. (1997). *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology.* New York.

Khajeh-Saeed, A., Poole, S. y Perot, J. B. (2010). Acceleration of the Smith–Waterman algorithm using single and multiple graphics processors. *Journal of Computational Physics, 229*(11), 4247–4258.

Mount, D. (2001). *Bioinformatics: Sequence and Genome Analysis.* New York.

Nawaz, Z., Nadeem, M., Someren, H. V., & Bertels, K. (2010). A parallel FPGA design of the Smith-Waterman traceback. *Proc. International Conference on Field-Programmable Technology*, 454–459.

Pevsner, J. (2009). *Bioinformatics and Functional Genomics.* New York.

Setubal, J. y Meidanis, J. (1997). *Introduction to Computational Molecular Biology.* Boston.

Shehab, S. A. Keshk, A. y Mahgoub, H. (2012). Fast Dynamic Algorithm for Sequence Alignment based on Bioinformatics. *International Journal of Computer Applications, 37*(7), 54–61.

The European Bioinformatics Institute EMBL-EBI. (2013). Recuperado de: http://www.ebi.ac.uk/Tools/psa/

The National Center for Biotechnology Information. (2013). Recuperado de: http://www.ncbi.nlm.nih.gov/genbank/

Zomaya, A. Y. (2006). *Parallel Computing for Bioinformatics and Computational Biology.* New Jersey.