UNIVERSIDAD NACIONAL JORGE BASADRE GROHMANN - TACNA

Facultad de Ingeniería

Escuela Académico Profesional de Ingeniería en Informática y Sistemas

"PROPUESTA DE UN ALGORITMO PARALELO PARA EL PROCESO DE ALINEAMIENTO DE PAREADO DE SECUENCIAS BIOMOLECULARES"

TESIS

Presentada por:

Bach. Wilson Cesar Callisaya Choquecota

Para optar el Título Profesional de:

INGENIERO EN INFORMÁTICA Y SISTEMAS

TACNA – PERÚ

2015

UNIVERSIDAD NACIONAL JORGE BASADRE GROHMANN, TACNA

FACULTAD DE INGENIERIA

JURADO CALIFICADOR Y CALIFICACIÓN DE LA SUSTENTACIÓN DE TESIS

"PROPUESTA DE UN ALGORITMO PARALELO PARA EL PROCESO DE ALINEAMIENTO DE PAREADO DE SECUENCIAS BIOMOLECULARES"

TESIS SUSTENTADA Y APROBADA EL 17 DE JULIO DEL 2015
ESTANDO EL JURADO CALIFICADOR INTEGRADO POR:

Presidente: .....
Dr. Julio Miguel Fernández Prado
Presidente

Presidente-Accesitario: .....
Msc. Erbert Francisco Osco Mamani
Presidente-Accesitario

Secretario: .....
Msc. Edgar Aurelio Taya Acosta
Secretario

Vocal: .....
Ing. Gianfranco Alexey Malaga Tejada
Vocal

Universidad Nacional Jorge Basadre Grohmann, TACNA

FACULTAD DE INGENIERIA

TESIS Nro.....

TITULO PROFESIONAL DE

Ingeniero en Informática y Sistemas

La Secretaría Academica de la Facultad de Ingeniería, por resolución de Facultad Nro. 02928-2015-FAIN/UNJBG designo Jurado Calificador Dictaminador para la sustentación de la Tesis titulada: "Propuesta de un algoritmo paralelo para el proceso de alineamiento de pareado de secuencias biomoleculares".

El mismo que está conformado por:

Presidente: Dr. Julio Miguel Fernández Prado

Presidente-Accesitario: Msc. Erbert Francisco Osco Mamani

Secretario: Msc. Edgar Aurelio Taya Acosta

Vocal: Ing. Gianfranco Alexey Malaga Tejada

Para calificar la sustentación de la tesis en acto público el día 17 de julio del 2015 presentado por el Bachiller Wilson Cesar Callisaya Choquecota, de la Escuela Académico Profesional de Ingeniería en Informática y Sistemas.

El Jurado Calificador en forma secreta e individual emitió su calificativo sobre el trabajo expuesto y procedió a obtener el promedio que arrojo el calificativo de Aprobado por Unanimidad con cuatro votos a favor con la nota de Quince.

Para ratificar lo detallado firma:

.....

Dr. Julio Miguel Fernández Prado

PRESIDENTE

.....

Msc. Erbert Francisco Osco Mamani

PRESIDENTE-ACCESITARIO

.....

Msc. Edgar Aurelio Taya Acosta

SECRETARIO

.....

Ing. Gianfranco Alexey Malaga Tejada

VOCAL

DEDICATORIA

A Dios

Por haber puesto en mi camino, a las personas que sin dudarlo me ayudaron para consolidar este logro importante.

A mi familia

A mi madre Manuela y mi padre Isidro que han estado conmigo en todos los momentos de mi formación, mis hermanos por sus consejos y palabras de aliento.

A mis profesores

Por darme siempre consejos e ideas para enrumbarme en el correcto camino para ser un profesional.

CONTENIDO

ÍNDICE DE FIGURAS

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 herramientas de la bioinformática que han sido desarrolladas con algoritmos secuenciales. 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. 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 más de un 50% en comparación con el algoritmo de alineamiento global de Needleman-Wunsch.

INTRODUCCIÓN

Desde la difusión de computadores de múltiples núcleos, se ha vuelto más accesible desarrollar aplicaciones que aprovechen los núcleos que estos disponen, muchos algoritmos actuales tienen que volverse a reformular para aprovechar esta tecnología. El alineamiento de secuencias es usado para la identificación de las diferentes secuencias genómicas. Siendo el principal problema de la biología computacional el tiempo de respuesta, pero los algoritmos tradicionales han sido diseñados para computadores de un solo núcleo.

La presente tesis propone un algoritmo que pueda usar los múltiples núcleos para el alineamiento de pares de secuencias, y así reducir el tiempo de respuesta con respecto al algoritmo secuencial usado actualmente.

En la presente tesis se describe en el capítulo I el planteamiento de la investigación definiendo detalles de la problemática y las variables, seguidamente el capítulo II nos muestra el marco teórico de la investigación, después tenemos en el capítulo III el desarrollo de la investigación, luego en el capítulo IV encontramos los resultados, posteriormente en el capítulo V la discusión respectiva de estos resultados, seguidamente en el capítulo VI vemos las conclusiones y recomendaciones y finalmente en el capítulo VII la bibliografía utilizada.

I. PLANTEAMIENTO DE INVESTIGACIÓN

1.1. Descripción del problema

1.1.1. Antecedentes del problema

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 de la fase 3 del algoritmo de alineamiento de secuencias pero no deja de ser este una propuesta secuencial. (Shehab, Keshk, & Mahgoub, 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 a este y muchos otros problemas de la Biología Computacional se muestra en el libro de Zomaya en el 2006, el cual nos da una propuesta de cómo podría ser el algoritmo paralelo, dando ideas iniciales de su granularidad fina y gruesa.

Paralelismo de Grano Fino: Los típicos algoritmos de programación basados en programación dinámica, como algoritmo de Smith-Waterman, calculan una matriz de mxn (m y m es la longitud de la secuencia) en función de las tres entradas H_{I-1J}, H_{IJ-1} y H_{I-1J-1}. Grano fino significa que los procesadores trabajarán conjuntamente en el cálculo de la matriz H, celda por celda. Algunos investigadores organizaron la máquina paralela como una matriz de procesadores para computar de manera diagonal de barrido de la matriz H como se muestra en la Fig. 1. Una ventaja es que esta estrategia sólo requiere comunicaciones locales en cada paso.

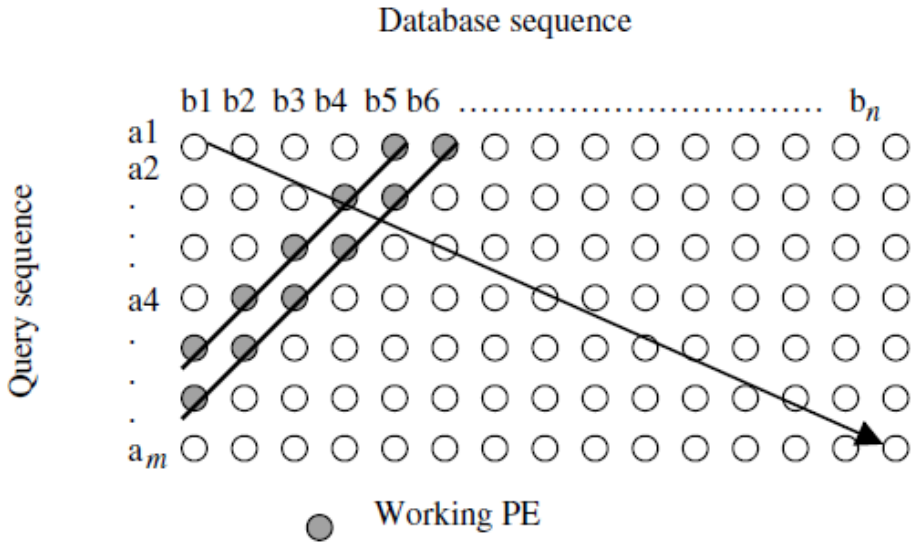

Figura 01: Diagonal-barrido de grano fino, distribución de la carga de trabajo para evitar dependencias de datos en el algoritmo de Smith-Waterman.

Fuente: (Zomaya, 2006, p. 242)

El paralelismo de grano grueso: Hay varias estrategias propuestas para lograr el paralelismo de grano grueso en aplicaciones de secuencia de alineación. En primer lugar, el programa establece la etapa inicial del algoritmo. A continuación, se administra el algoritmo de extensión, que funciona hasta que se agote el número de secuencias de base de datos, a continuación, obtiene la siguiente secuencia de base de datos para ser comparado contra la secuencia de consulta.

Como debe tenerse en cuenta, el algoritmo tiene una forma muy simple en lo que respecta al flujo de datos. La secuencia de bases de datos corresponde con el conjunto de datos que se debe buscar, lo que es un conjunto de secuencias de diferentes longitudes. En esencia, en una implementación paralela de grano grueso típica, uno de los procesadores actúa como un "maestro", el envío de bloques de secuencias a los "esclavos", que, a su vez, llevan a cabo los cálculos del algoritmo. Cuando los esclavos reportan resultados para un bloque, el maestro envía un nuevo bloque. Esta estrategia es posible porque los resultados de la comparación entre las secuencias de consulta y de bases de datos son independientes de los resultados anteriores se derivan de la comparación de la consulta con otras secuencias.

Sin embargo, el tiempo necesario en el procesamiento de cualquier secuencia dada depende no sólo de la longitud de la secuencia,

sino también en su composición. Por lo tanto, es necesario el uso de una estrategia de equilibrio de carga dinámica. La forma más sencilla es modificar la forma en que el procesador maestro distribuye la carga en la demanda de los esclavos. Obviamente, el envío de mensajes de una sola secuencia de tiempo introduce sobrecarga adicional cara debido al alto número de mensajes intercambiados. Por lo tanto, en lugar de la distribución de secuencia de mensajes por secuencia, se obtienen mejores resultados mediante el envío de bloques de secuencias. (Zomaya, 2006, pp. 242–244)

Ya en el año 2010 se han presentado publicaciones para solucionar este problema con procesamiento paralelo, una de estas propuestas fue realizarlo con un hardware especializado como lo es el paper presentado por Nawaz, Nadeem, Someren, & Bertels que presenta una solución en procesamiento paralelo con Field Programmable Gate Array (FPGA).

Hay muchas implementaciones Smith-Watterman(SW) sobre FPGA, que muestran aceleraciones de hasta 100 veces en comparación con un procesador de propósito-general (GPP). En ese trabajo, se propuso un diseño del Smith-Wattermann de rastreo, que se realiza en paralelo con la etapa de llenado de la matriz y que da la alineación óptima después de una exploración a través de toda la base de datos.

Paralelización y relleno del algoritmo SW: Tenemos que mirar en su grafo de dependencias de datos, como se muestra en la Fig. 2 (diferentes tonos de gris en los círculos muestran los elementos que se pueden ejecutar en paralelo). Círculos en blanco son los elementos después de la inicialización de las condiciones de contorno. Cualquier iteración (ij) no se puede ejecutar hasta que iteraciones (i - 1, j), (i - 1, j - 1) y (i, j - 1) sean ejecutadas primero, debido a las dependencias de datos. Sin embargo, si se recorre los elementos de una forma de frente de onda a partir de la esquina superior izquierda como se muestra en la Fig. 2, todos los elementos de la diagonal se pueden ejecutar en paralelo. Esta ejecución paralela se llama aplicación de flujo de datos, ya que todos los cálculos se realizan cuando es sus datos disponibles.

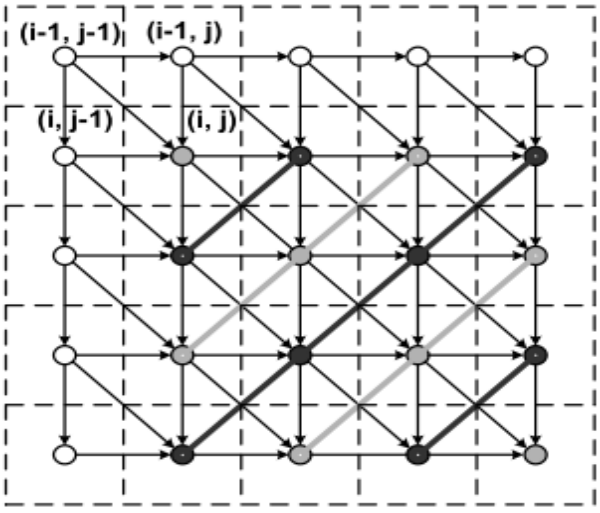

Figura 02: Grafo de dependencias de datos

Fuente: (Nawaz, Nadeem, Someren, & Bertels, 2010)

Smith-Waterman RVEP aplicación: RVEP calcula anti-diagonales de bloques en forma de flujo de datos en lugar de anti-diagonales de elementos de flujo de datos clásico. Esto se muestra en la Fig. 3, que muestra la secuencia de relleno para los bloques con factor de bloqueo B = 2. El tamaño del bloque es b = B x B. Cada bloque contiene cuatro elementos O1, O2, O3 y O4. En la Fig. 3, primero se calcula todos los elementos en el bloque con el ciclo 1, entonces se calcula todos los elementos en la lucha contra la diagonal de bloques con ciclo de 2 y así sucesivamente. Esta estructura sugiere que utilicemos una matriz sistólica lineal para hacer el cálculo, donde se utiliza un mismo circuito bloque sistólicamente para el cálculo de los bloques en la misma columna que el anti-diagonal de bloques progresa.

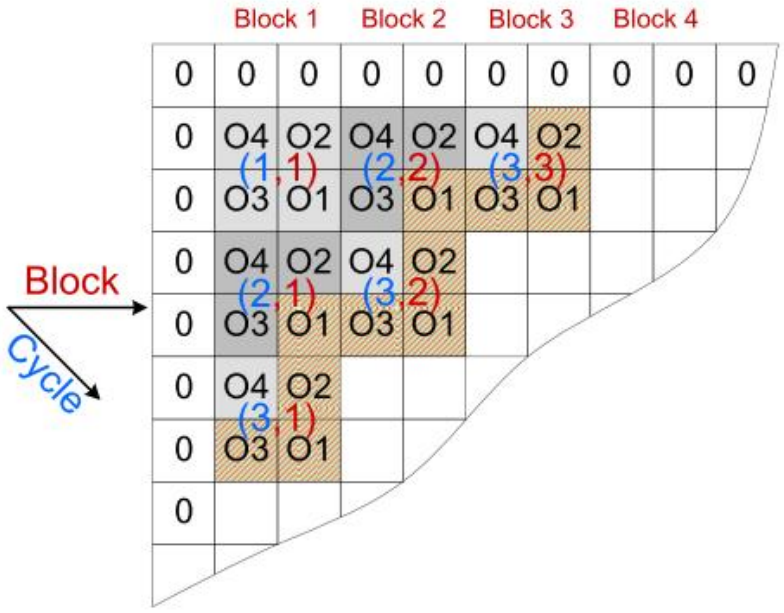

Figura 03: Secuencia de relleno de la matriz de scores para Bloques de tamaño 2.

Fuente: (Nawaz et al., 2010)

En la Fig. 3. se muestra en bloques en la parte superior el hardware para ser utilizado en cada columna. Los Números en rojo representan el bloque utilizado y los números en azul representan el ciclo en el que se utiliza.

En ese trabajo, se ha propuesto un diseño de FPGA en paralelo del algoritmo de Smith Waterman de rastreo, que construye la mejor alineación entre la secuencia desconocida y su secuencia conocida más cercana genéticamente de la base de datos. (Nawaz et al., 2010)

La aparición de los multiprocesadores en las tarjetas de video y tener la posibilidad de programar en ellos hace que apliquemos la programación paralela en esta Tecnología llamada CUDA (Compute Unified Device Architecture), el paper presentado por (Khajeh-Saeed, Poole, & Blair Perot, nos muestra otra propuesta para el problema del alineamiento de secuencias en CUDA usando la GPU (Graphics Processor Unit).

Estos autores indican que al igual que muchas aplicaciones en la ciencia computacional, el algoritmo de Smith-Waterman se ve limitada por la velocidad de acceso a memoria y se puede acelerar de manera significativa mediante el uso de procesadores de gráficos (GPU) como el motor de cálculo.

En su trabajo se muestra que el uso efectivo de la GPU requiere una nueva reformulación del algoritmo de Smith-Waterman. El rendimiento demostró que una GPU es hasta 45 veces más rápida que una CPU para esta aplicación, y la aplicación paralela muestra la velocidad lineal hasta en un máximo de 4 GPUs.

Afortunadamente los costos y precios de los productos básicos ya están disponibles, tales como procesadores de gráficos (GPUs) y procesadores de juego, que son más adecuadas para el acceso de datos. Los procesadores gráficos (que ahora son estándar en la mayoría de los

PCs) se convirtieron programables en el 2005 y más amigable programar en ellos a través de una interfaz de tipo C (CUDA 2007/0penCL 2008) a principios de 2008. Cuando se utiliza como un acelerador de matemáticas para los algoritmos científicos resultados GPU en más o menos un orden de magnitud mejor rendimiento que la que puede obtenerse con una CPU de múltiples núcleos.

La compensación implica el uso de un procesador de gráfico en lugar de una CPU de propósito general es que una GPU está diseñado para un propósito específico, y no computación de propósito general. Mientras que el hardware de la GPU es muy adecuado para la informática científica, todavía es necesario entender el hardware y sus limitaciones si el tratado para el rendimiento es a alcanzar.

Ese trabajo se centra en la aceleración de hardware del algoritmo de Smith-Waterman clásico (sin heurística) esto lo hace principalmente debido a que la heurística es algo específico para el dominio de su aplicación.

La velocidad relativamente lenta de las conexiones de red relativas a velocidades de memoria significa que la paralelización de las GPU múltiple (más de 4) requiere de grano grueso como una descomposición del problema como sea posible.(Khajeh-Saeed, Poole, & Blair Perot, 2010)

La idea se muestra en la Fig. 4.

Figura 04: Diagrama esquemático de la superposición para el uso de múltiples GPU

Fuente: (Khajeh-Saeed et al., 2010)

Como se puede apreciar estas implementaciones están más vinculadas al hardware, procesando las instrucciones con equipo especializado siendo CUDA la opción más cercana al usuario común, pero ya han aparecido librerías que nos permiten realizar el trabajo paralelo con lenguajes de programación de alto nivel.

Ya en el año 2013 realice dos trabajos referidos a este tema pero en aquellas oportunidades el experimento se realizó en un computador con 4 núcleos aprovechando la librería TPL (Task parallel library) que nos ofrece el C# con el Framework 4.0 de Visual Studio. (Callisaya, Callohuari, Jimenez, 2013a)(Callisaya, Callohuari 2013b).

1.1.2. Problemática de la investigación

En el mundo de la bioinformática el alineamiento de secuencias es usado para la identificación de las diferentes secuencias genómicas. Este es el principal problema de la biología computacional, el tiempo de respuesta para el alineamiento de pareado de secuencias ha sido un problema mayor dando en la actualidad soluciones heurísticas, puesto la solución exacta realizada con programación dinámica era muy lenta. Aun con la aparición del paradigma paralelo aún no se implementan soluciones paralelas en lenguajes de alto nivel que apoyen a los diferentes algoritmos de la biología computacional.

En el Perú aun la bioinformática está en sus comienzos y en lo que respecta en biología computacional poco se hace, hay un desconocimiento de los profesionales de esta área de la utilidad de la programación paralela que pudiera apoyar a los diferentes proyectos que estos realizan como “Algoritmos de reconocimiento de patrones MODS para el diagnóstico de tuberculosis” desarrollándose en la UPCH, actualmente hay dos sitios donde se practica la bioinformática propiamente dicha la Unidad de Bioinformática / UPCH , a cargo del Dr. Mirco Zimic y el laboratorio de Bioinformática / IPEN, a cargo de la Dra. Silvia Vásquez.

En la facultad de ingeniería en informática y sistemas de la UNJBG (Universidad Nacional Jorge Basadre Grohmann) de Tacna recientemente han impulsado los cursos de algoritmos paralelos y programación paralela, conocimientos cuales podrían apoyar en la mejora de los algoritmos secuenciales de la biología computacional.

En la Biología Computacional se observa diversas posibilidades de aplicación de los conocimientos de la carrera de ingeniería en informática y sistemas, en especial sobre este nuevo paradigma de la programación paralela y esto ha motivado a la realización del presente trabajo denominado: “Propuesta de un algoritmo paralelo para el proceso de alineamiento de pareado de secuencias biomoleculares”

1.2. Formulación del problema

Problema General

¿En qué medida un algoritmo paralelo influye en el desempeño del alineamiento de pares de secuencias biomoleculares?

Problemas Específicos

- ¿Cómo funciona el algoritmo que se usa para el alineamiento de pares de secuencias biomoleculares tradicional?

- ¿Cómo diseñar un algoritmo paralelo para el alineamiento de secuencias biomoleculares?

1.3. Justificación

La importancia de la alineación de secuencias para la bioingeniería tanto para poder realizar estudios moleculares es muy requerido para el desarrollo de nuevos fármacos, para estudios genéticos, el genoma humano tiene 3 200 000 000 millones de pares de bases haciendo actualmente imposible un alineamiento en un tiempo aceptable es por ello que se requiere de nuevos métodos que puedan incrementar el tiempo de respuesta la alternativa paralela puede reducir grandemente el tiempo de respuesta de este problema.

Para poder realizar este proyecto será necesario usar tecnologías ya diseñadas previamente tanto la librería OPENMP (Open MultiProcessing) como la librería TPL, con el cual se desarrollara el problema, con ello difundiendo el uso de estas herramientas de tecnologías de información.

Existe necesidad de documentación tanto de biología computacional como de programación paralela, siendo este proyecto de

gran valor académico para futuras investigaciones en esta línea, para nuevas propuestas de trabajo.

1.4. Alcances y limitaciones

1.4.1. Alcances

La presente tesis se enfocará en el diseño del algoritmo paralelo para el paso 2 del algoritmo de alineamiento de pares de secuencia global, tratado en el 2.2.1.8.

1.4.2. Limitaciones

- Bibliográfica: Escasa bibliografía en las bibliotecas locales. Se superó esta limitación recurriendo a bibliotecas virtuales
- Área geográfica: Los centros de investigación de bioinformática no están en la localidad, impidiendo realizar consultas a investigadores relacionados en el tema, se solucionó haciendo consultas en línea y asistiendo a congresos donde participaron estos especialistas.

- Financiamiento: Es necesario contar con computadores de alto rendimiento para efectuar las pruebas necesarias. Fue necesario la compra de un computador core i7 para el desarrollo de la Tesis.

1.5. Objetivos

1.5.1. Objetivo general

Diseñar un algoritmo paralelo para el alineamiento de pares de secuencias biomoleculares.

1.5.2. Objetivos específicos

- Conocer el funcionamiento del algoritmo de alineamiento de pares de secuencias biomoleculares que se usa actualmente.
- Proponer el diseño de un algoritmo paralelo para alineamiento de secuencias biomoleculares.

1.6. Hipótesis

1.6.1. Hipótesis global

- H1: Con el algoritmo paralelo propuesto para alinear secuencias biomoleculares se reduce el tiempo de respuesta respecto al secuencial
- Ho: Con el algoritmo paralelo propuesto para alinear secuencias biomoleculares no se reduce el tiempo de respuesta respecto al secuencial.

1.7. Variables

1.7.1. Identificación de variables

- Algoritmo paralelo.
- Proceso de alineamiento de pareado de secuencias biomoleculares.

1.7.2. Definición de variables

- Algoritmo paralelo: Algoritmo diseñado con la técnica de programación que utiliza a los múltiples procesadores del computador para realizar una o varias tareas.
- Proceso de alineamiento de pareado de secuencias biomoleculares: Proceso en el que se usa un algoritmo secuencial de programación dinámica para identificar similitudes entre dos secuencias de ADN, ARN o aminoácidos.

1.7.3. Operacionalización de variables

Cuadro 1: Operacionalización de las variables

Variable	Definición conceptual	Definición operacional	Indicadores
Variable independiente: Algoritmo paralelo	Algoritmo diseñado con la técnica de programación que utiliza a los múltiples procesadores del computador para realizar una o varias tareas.	Tiempo de respuesta.	Tiempo de ejecución.
		Incremento de aceleración.	SpeedUp
Variable dependiente Proceso de alineamiento de pareado de secuencias biomoleculares.	Proceso en el que se usa un algoritmo secuencial de programación dinámica para identificar similitudes entre dos secuencias de ADN, ARN o aminoácidos.	Tiempo de respuesta	Tiempo de ejecución

Fuente: Elaboración propia

1.7.4. Clasificación de las variables

- Variable independiente: Algoritmo paralelo.
- Variable dependiente: Proceso de alineamiento de pareado de secuencias biomoleculares.

1.8. Diseño de la investigación

1.8.1. Diseño experimental o no experimental

Se realizó un diseño experimental con post-prueba únicamente y un grupo de control. Al ser el proceso de alineamiento de secuencias no influyente en los datos iniciales de las secuencias el grupo de control y el grupo experimental son iguales, para así medir exclusivamente los tiempos en ambos escenarios.

Grupos	*Proceso experimental*	*Prueba de salida*
G.C	*X1*	*O1*
G.E	*X2*	*O2*

Dónde:

GC = Grupo de control.

GE = Grupo experimental.

O1; O2 = Prueba de salida.

X1 = Proceso de alineamiento tradicional (Secuencial).

X2 = Proceso de alineamiento con programación paralela.

1.8.2. Población y muestra

1.8.2.1. Población

Las secuencias que se seleccionaron fueron extraídas del GENBANK (Base de Secuencias Biomoleculares), teniendo en cuenta que esta población es desconocida en número de secuencias actualmente almacenadas, y de diferentes tamaños de 20 hasta 20 millones de pb (pares de bases).

También se consideró a subgrupos para cada medición para longitudes de secuencias que no se diferencien en más de 400 pb a diferentes tamaños referenciales que estarán en longitudes de 400 pb a 12 000 pb que se usara para observar los tiempos de ejecución de los

algoritmos secuencial y paralelo a medida que las longitudes de secuencia cambian.

1.8.2.2. Muestra

Al ser el proceso de alineamiento un proceso en el cual participan dos secuencias para posteriormente mostrar el alineamiento ideal, se seleccionaran 60 secuencias para posteriormente realizar 30 alineamientos. Siendo los grupos experimental y de control iguales para medir solo el tiempo que demoran en realizar el mismo proceso ambos algoritmos.

Para el cálculo del tamaño de muestra se hizo en base a la fórmula clásica de Freeman: n=10*(k+1), donde k es el número de variables independientes, para usar la regresión lineal sin problemas (Freeman, 1987).

Dado que el número de variables independientes son 2 mi k=2, entonces reemplazando en la formula n=10*(2+1), un total de n=30, entonces realizaremos 30 alineamientos para el experimento y se seleccionaran un total de 60 secuencias para realizar estas pruebas de tamaños entre 400 pb hasta 12 000 pb.

Siendo los tamaños de nuestras muestras.

n1 (tamaño de la muestra del grupo 1) = 30

n2 (tamaño de la muestra del grupo 2) = 30

1.8.3. Técnicas e instrumentos para recolección de datos

Como técnica de recolección de datos se usó la observación, medición directa de los datos y como instrumento de medición guía de observación en una hoja de anotaciones donde se apuntó los datos obtenidos por el cronómetro digital de gran precisión que encontramos en la clase StopWatch disponible en la librería “System.Diagnostic” de C#, dicha herramienta brindada por la Microsoft en el Visual Studio 2010, el cual midió el tiempo en que se demoran en realizar el alineamiento de secuencias ambas técnicas con el algoritmo tradicional (secuencial) y el algoritmo paralelo propuesto.

1.8.4. Análisis de datos

Para el análisis de los datos, se usaran gráficos comparativos y se calculara el SpeedUp (Ver Sub Apartado 2.2.2.4.) para las diferentes

longitudes de secuencias, en la cual se realizara una regresión lineal para identificar como se comportaría a longitudes más grandes.

II. MARCO TEÓRICO

2.1. Bases teóricas de biologia computacional

Entre las herramientas más útiles basados en computadoras en la biología moderna son aquellos que involucran sí secuencia de alineaciones de proteínas, ya que estas alineaciones a menudo proporcionan importantes conocimientos sobre genética y función de las proteínas. Hay varios tipos diferentes de alineaciones: alineamientos globales de pares de proteínas relacionadas por una ascendencia común a lo largo de sus longitudes, alineamientos locales que implican segmentos relacionados de proteínas, múltiples alineaciones de los miembros de las familias de proteínas, y alineaciones realizadas durante búsquedas en bases de datos para detectar homologías. (Henikoff & Henikoff, 1992)

2.1.1. La importancia de la comparación de secuencias

El primer hecho de análisis de secuencias biológicas En las secuencias biomoleculares (ADN, ARN, o secuencias de aminoácidos), alta similitud

de secuencia por lo general implica similitud funcional o estructural significativo.

La vida se basa en un repertorio de bloques moleculares estructurados e interrelacionados que se comparten y se pasan alrededor. Los mismos y relacionados con estructuras y mecanismos moleculares que aparecen varias veces en el genoma de una sola especie y a través de un espectro muy amplio de especies divergentes. "Duplicación con modificación" es el paradigma central de la evolución de proteínas, en el que nuevas proteínas o nuevas funciones biológicas se forman a partir de las anteriores.

Hoy en día, el método más poderoso para inferir la función biológica de un gen (o la proteína que codifica) es por la secuencia de búsqueda de similitud en bases de datos de proteínas y ADN de secuencia. Con el desarrollo de métodos rápidos para la comparación de secuencias, tanto con algoritmos heurísticos y potentes ordenadores paralelos, descubrimientos basados únicamente en la homología de secuencia se han convertido en rutina.

La comparación de secuencias a gran escala, por lo general organizada como búsqueda de base de datos, es una herramienta muy poderosa para la inferencia biológica en la biología molecular moderna. Y esa herramienta es usada casi universalmente por los biólogos

moleculares. Ahora es una práctica estándar, cada vez que un nuevo gen se clona y se secuencia, para traducir su secuencia de ADN en una secuencia de aminoácidos y, a continuación buscar similitudes entre él y los miembros de las bases de datos de proteínas. Hoy a nadie se le ocurriría publicar la secuencia de un gen clonado nuevo sin hacer una búsqueda en las de bases de datos.(Gusfield, 1997, pp. 213–214)

El propósito de los algoritmos de alineamiento de secuencias es detectar las relaciones evolutivas, y por lo tanto estructurales y funcionales, entre las secuencias. Una exitosa comparación de secuencias nos permitiría inferir las propiedades biológicas de nuevas secuencias de datos acumulados en los genes relacionados. Por ejemplo, una similitud entre una secuencia de nucleótidos traducida y una secuencia de la proteína conocida sugiere una región de codificación homóloga en la secuencia de nucleótidos correspondiente. Significativa similitud de secuencia entre las proteínas puede implicar que las proteínas de compartir las mismas estructuras secundarias y terciarias, y tienen funciones biológicas cerca. La predicción de estructuras de proteínas desconocidas se basa a menudo en el estudio de las estructuras conocidas de proteínas homólogas.

Hoy en día, el procedimiento de rutina para el análisis de una nueva secuencia de la proteína casi siempre comienza con una

comparación de la secuencia de la mano con las secuencias en una o más de las principales bases de datos de secuencias. Una nueva secuencia se analiza mediante la extrapolación de las propiedades de sus 'Vecinos' en una búsqueda de base de datos. Estos métodos se han aplicado durante las últimas tres décadas con mucho éxito y han ayudado a identificar la función biológica de muchas secuencias de proteínas, así como para revelar muchas relaciones distantes e interesantes entre las familias de proteínas. En realidad, más secuencias han sido supuestamente caracterizadas por la búsqueda de bases de datos que los de cualquier otra tecnología. (Higgins & Taylor, 2000, pp. 167–168)

2.1.2. Elementos Clave a tomar en cuenta al Alinear

La tarea más elemental análisis de la secuencia es preguntar si dos secuencias están relacionadas. Esto se realiza por lo general primero mediante la alineación de las secuencias (o partes de ellos) y a continuación, decidir si es más probable que haya ocurrido debido a que las secuencias están relacionadas, o simplemente por casualidad que la alineación. Las cuestiones clave son:

(1) ¿Qué tipo de alineación se deben considerar?

(2) El sistema de puntuación utilizado para clasificar las alineaciones

(3) El algoritmo usado para encontrar una alineación con puntuación óptima.

(4) Los métodos estadísticos utilizado para evaluar la importancia de una puntuación de alineamiento.

(Durbin, Eddy, Krogh, & Mitchison, 1998, p. 12)

2.1.3. Alineamiento de Secuencias:

El Análisis comparativo de secuencias es el primer paso para estudiar la relación de secuencia estructura-función de proteínas y secuencias de nucleótidos. La comparación de la secuencia de una determinada proteína con los de las bases de datos de proteínas anotadas a menudo da pistas muy importantes sobre la estructura 3D de la proteína y su posible función. Estructura y función de proteínas siguen siendo dos grandes problemas no resueltos de la biología molecular. Las proteínas tienen una gran variedad de estructuras en 3D, y dar sentido a estas estructuras observadas es un desafío: ¿cómo podemos reconocer las semejanzas y diferencias importantes entre las estructuras de las proteínas y cómo funciona la estructura se relacionan con la función de proteínas?

El esquema de alineamiento global es muy bueno para la comparación y el análisis de la relación entre dos secuencias seleccionadas. Las proteínas, sin embargo, a menudo se componen de diferentes dominios, en los que cada dominio puede estar relacionado con una determinada función. Por lo tanto cuando se trata de la búsqueda de secuencias relacionadas funcionalmente a menudo es beneficioso para buscar pequeñas partes de las secuencias que son similares entre sí. Esta búsqueda consiste en hacer un alineamiento por pares de la secuencia de consulta a todas las secuencias en la base de datos, y ordenar las alineaciones resultantes de la puntuación de alineamiento. Los éxitos son algunas de las mejores secuencias de la base de datos que tienen una alineación de puntuación con la secuencia de consulta. (Kesmir, 2013, pp. 33–34)

Si logramos generar un algoritmo eficiente, vamos a estar en una buena posición para ir a pescar en los bancos de datos de secuencias relacionadas. Una aplicación importante es para la anotación de los genomas, que implica la asignación de la estructura y función a la mayor cantidad posible de genes.

¿Cómo podemos definir una medida cuantitativa de similitud de secuencia? Para comparar los nucleótidos o aminoácidos que aparecen en las posiciones correspondientes de dos o más secuencias, primero

tenemos que asignar dichas correspondencias. El alineamiento de secuencias es la identificación de las correspondencias de residuos a residuos.

Cualquier asignación de correspondencias que conserva el orden de los residuos dentro de las secuencias es una alineación. Este análisis se complica al considerar los GAPS (huecos). Ejemplos de Alineamiento se observan en la Fig. 5.

- - - - - - - g c t g a a c g c t a t a a t c - - - - - - - -	Un alineamiento informativo
g c t g a a c g c t a t a a t c	Un alineamiento sin huecos
g c t g a - a - - c g - - c t - a t a a t c	Una alineación con huecos
g c t g - a a - c g - c t a t a a t c -	y otro

Figura 05: Ejemplos de Alineamiento

Fuente: (Lesk, 2002, p. 154)

Podríamos considerar que en la Fig. 5 la última de estas alineaciones es la mejor de las cuatro. Pero para decidir si es la mejor de todas las posibilidades, necesitamos una forma de examinar todas las alineaciones posibles sistemática. Entonces tenemos que calcular una

puntuación que refleja la calidad de cada alineación posible, y para identificar la alineación con la puntuación óptima. La alineación óptima puede no ser único: varias alineaciones diferentes pueden dar la misma mejor puntuación. Por otra parte, incluso pequeñas variaciones en el sistema de puntuación puede cambiar la clasificación de las alineaciones, causando una diferente para emerger como la mejor.(Lesk, 2002, p. 154).

Como se observó en la figura 5 dos secuencias se alinean por escrito ellos a través de una página en dos filas. Caracteres idénticos o similares se colocan en la misma columna, y los caracteres no idénticos o bien se pueden colocar en la misma columna que una falta de coincidencia o delante de un hueco en la otra secuencia. En una alineación óptima, caracteres no idénticos y vacíos son colocados para que los caracteres idénticos o similares como sea posible en el registro vertical. Secuencias que pueden ser fácilmente alineados de esta manera se dice que son similares.

Hay dos tipos de alineamiento de secuencias: las globales y locales. En el alineamiento global, se hace un intento de alinear la secuencia completa, usando tantos caracteres como sea posible, hasta ambos extremos de cada secuencia. Las secuencias que son muy similares y aproximadamente la misma longitud son candidatos adecuados para la alineación global.

2.1.4. Alineando con Sistema de puntuación:

Estos cambios se dan por eventos evolutivos, las secuencias han cambiado por inserciones, deleciones y mutaciones. Estos eventos evolutivos los desvelaremos al alinear las secuencias. Supongamos que una secuencia de ADN a ha evolucionado a la secuencia de B a través de sustituciones, inserciones y deleciones. Esta transformación puede ser representada por una alineación donde a es escrita por encima de b con el común (conservada) alineado apropiadamente. Por ejemplo, dicen que a = ACTTGA y b se obtiene mediante la sustitución de la segunda de base de C a G, la inserción de una A, entre la segunda y la tercera bases, y mediante la supresión de la quinta base (G). La alineación correspondiente será:

a	=	A	C	-	T	T	G
b	=	A	G	A	T	T	-
score	=	1	0	-1	1	1	-1

Figura 06: Ejemplo de alineamiento con puntuación

Fuente: (Higgins & Taylor, 2000, p. 167)

Por lo general, en realidad no sabemos qué secuencia se desarrolló de la otra. Por lo tanto, los hechos no son direccionales y la inserción de A en B podría haber sido una deleción de una en una.

En una aplicación típica se nos dan dos secuencias relacionadas y que queremos recuperar los eventos evolutivos que transformaron una a la otra. El objetivo de la alineación de secuencias es encontrar la alineación correcta que codifica la verdadera serie de eventos evolutivos que se han producido. La alineación se puede asignar una puntuación que representa el número de identidades (dos letras idénticas), el número de sustituciones (dos letras diferentes), y el número de huecos (inserciones / deleciones). Por ejemplo, en la alineación anterior, una puntuación de 1 se le dio a cada identidad, una puntuación de 0 se le dio a cada sustitución, y una puntuación negativa de -1 fue dada para cada espacio. En general, la alineación anotó 2, que es la suma de todas las puntuaciones par y las puntuaciones de la brecha. En general, las calificaciones de las identidades y las sustituciones que se utilizan para marcar la alineación se llaman la matriz de puntuación, y las calificaciones de los huecos (Gaps) se llaman puntuación de sanciones. En conjunto se les llama el sistema de puntuación. Con altas calificaciones (positivo) para las identidades, y las puntuaciones bajas o negativas para las sustituciones y los huecos, la estrategia básica para trazar la alineación

correcta busca la alineación que obtuvo el mejor resultado.(Higgins & Taylor, 2000, pp. 167–168)

Las inserciones y deleciones se denominan generalmente gaps. La selección natural tiene un efecto sobre este proceso mediante el filtrado de las mutaciones, de manera que algunos tipos de cambio pueden ser vistos más que otros.

La puntuación total se asigna a una alineación será una suma para cada par de residuos alineados, sumado con el puntaje de cada gap. Informalmente, se espera que las identidades y sustituciones conservadas sea más probable en las alineaciones que se espera al azar, y así contribuimos términos puntuación positiva, y se esperan cambios no conservativos que observar con menos frecuencia en las alineaciones reales de lo que esperamos por casualidad, por lo que éstos contribuyen términos puntuación negativa. (Durbin et al., 1998, pp. 13–14)

2.1.5. Métodos de alineamiento de pares de secuencias:

La alineación de dos secuencias se realiza utilizando los siguientes métodos:

1. Análisis de matriz de puntos
2. El algoritmo de programación dinámica (o DP)

3. Word o k-tupla métodos, tales como el utilizado por los programas FASTA y BLAST.

Si no se conocen el parecido de las secuencias, el método de matriz de puntos se debe utilizar en primer lugar, porque este método muestra los posibles alineamientos de secuencias como diagonales de la matriz. Análisis de matriz de puntos puede revelar fácilmente la presencia de inserciones / deleciones y repite directos e invertida que son más difíciles de encontrar en los otros métodos, más automatizados. La principal limitación de este método es que la mayoría de los programas de ordenador de matriz de puntos no muestran una alineación real.

El método de programación dinámica, se utiliza para alineación global de secuencias de Needleman y Wunsch (1970) y para la adaptación local de Smith y Waterman (1981), proporciona una o más alineaciones de las secuencias. Un alineamiento es generado por a partir de los extremos de las dos secuencias y el intento de coincidir con todos los posibles pares de caracteres entre las secuencias y, siguiendo un esquema de puntuación para los partidos, desajustes, y lagunas. Este procedimiento genera una matriz de números que representa todas las posibles alineaciones entre las secuencias. El conjunto más alta de las puntuaciones secuenciales en la matriz define una alineación óptima.

El método de programación dinámica se garantiza en un sentido matemático para proporcionar la alineación óptima (mejor o más alta puntuación) para un conjunto dado de variables definidas por el usuario, incluyendo la elección de la matriz de puntuación y la brecha de sanciones. Afortunadamente, la experiencia con el método de programación dinámica ha proporcionado mucha ayuda para la toma de las mejores decisiones, y la programación dinámica ha sido ampliamente utilizada. El método de programación dinámica también puede ser lento debido al muy gran número de etapas de cálculo, que aumentan aproximadamente como el cuadrado o el cubo de las longitudes de secuencia. El requisito de memoria de la computadora también aumenta con el cuadrado de la longitud de la secuencia. Por lo tanto, es difícil utilizar el método de secuencias muy largas. Afortunadamente, los informáticos han reducido en gran medida estos requisitos de espacio y tiempo a cerca de las relaciones lineales sin comprometer la fiabilidad del método de programación dinámica, y estos métodos son ampliamente utilizados en las aplicaciones de programación dinámicos disponibles a la alineación de secuencias.

El método de la palabra o K-tuplas métodos que son utilizados por el FASTA y los algoritmos BLAST. Alinean dos secuencias muy rápidamente, por primera búsqueda de tramos cortos de secuencias

idénticas (llamadas palabras o K-tuplas) y para entonces unirse a estas palabras en una alineación por el método de programación dinámica. Estos métodos son lo suficientemente rápido para ser adecuados para la búsqueda de una base de datos completa de las secuencias que se alinean mejor con una secuencia de prueba de entrada. Los métodos FASTA y BLAST son heurísticos, es decir, un método empírico de programación de computadora en el que las reglas generales se utilizan para encontrar soluciones y la retroalimentación se utiliza para mejorar el rendimiento. Sin embargo, estos métodos son fiables en un sentido estadístico, y por lo general proporcionan una alineación fiable. (Mount, 2001, pp. 53–57)

2.1.6. Alineamientos Globales y Locales

Alineamiento Global: Para las dos proteínas hipotéticas fragmentos de la secuencia en la Fig. 7, el alineamiento global se extiende sobre toda la longitud de la secuencia para incluir tantos aminoácidos coincidentes como posibles hasta e incluyendo los extremos de la secuencia. Las barras verticales entre las secuencias indican la presencia de aminoácidos idénticos. Aunque no es obvia una región de identidad en este ejemplo (la secuencia de GKG precedida por una sustitución

comúnmente observado de T para A), una alineación global no puede alinear dichas regiones de manera que más aminoácidos a lo largo de la totalidad de las longitudes de secuencia pueden ser emparejados.

Alineamiento Local: En un alineamiento local, la alineación se detiene en los extremos de las regiones de identidad o fuerte similitud, y se le da una prioridad mucho mayor a la búsqueda de estas regiones locales como se observa en la Fig. 7 que se extiende a la alineación para incluir pares de aminoácidos más vecinos. Los guiones indican la secuencia no incluido en la alineación. Este tipo de alineación favorece la búsqueda de patrones conservados de nucleótidos, secuencias de ADN, o los patrones de aminoácidos en las secuencias de proteínas. (Mount, 2001, pp. 53–54)

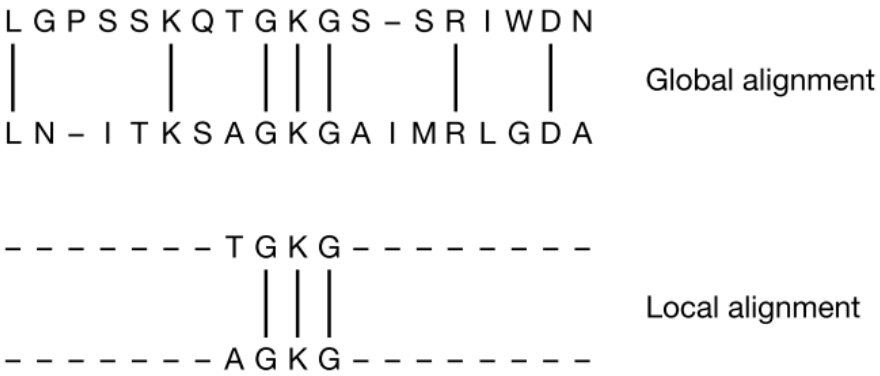

Figura 07: Alineamiento Global y Local

Fuente: (Mount, 2001, p. 53)

2.1.7. Programación Dinámica para el Alineamiento de Secuencias

Los Métodos de programación Dinámica son una clase general de algoritmos que se ven a menudo tanto en el alineamiento de secuencias y otros problemas de cálculo. Fueron descritos por primera vez en 1950 por Richard Bellman de la Universidad de Princeton como una técnica de optimización general. La programación dinámica parece haber sido introducido en la comparación de secuencias biológicas por Saul Needleman y Christian Wunsch, que al parecer no tenían conocimiento de la similitud entre su método y de Bellman.

Los algoritmos de programación dinámica nos ayudan a resolver problemas de optimización, los problemas en los que hay un gran número de posibles soluciones, pero sólo uno o un pequeño número de las mejores soluciones. Un algoritmo de programación dinámica encuentra la mejor solución por primera romper el problema original en subproblemas más pequeños y, a continuación la solución. Estas piezas del problema más grande tienen una dependencia secuencial, es decir, la cuarta pieza puede ser resuelta sólo con la respuesta de la tercera pieza, el tercero se puede resolver sólo con la respuesta a la segunda, y así sucesivamente. La programación dinámica trabaja primero solucionando todos estos subproblemas, almacenando cada solución intermedia en una tabla junto

con una puntuación, y, finalmente se selecciona la secuencia de soluciones en la que se obtiene la puntuación más alta. El objetivo del algoritmo de programación dinámica es maximizar la puntuación total de la alineación. El objetivo es que el número de pares de residuos de alta puntuación debe ser maximizado y el número de huecos y pares de baja puntuación debe ser minimizado.

El algoritmo utiliza una matriz de puntuaciones (mxn) en la que m y n son las longitudes de las secuencias que están siendo alineadas. A partir de la alineación de un gap contra sí misma (que se le otorga la puntuación inicial cero), el algoritmo se llena en la matriz de una fila a la vez. En cada posición en la matriz el algoritmo calcula las puntuaciones que dan como resultado para cada uno de sus tres opciones, selecciona la que proporciona la puntuación más alta, a continuación, almacena un puntero en la posición actual a la posición anterior que se utilizó para llegar a la alta puntuación. Cuando todas las posiciones en la matriz ha sido rellenado, se realiza una etapa de rastreo, y la ruta de acceso de mayor puntuación a lo largo de los punteros es seguida de nuevo al principio de la alineación.(Gibas & Reilly, 2001, pp. 157–158)

2.1.8. Algoritmo de Alineamiento

La Alineación global implica una disposición de dos secuencias con sus posiciones de residuos conservados evolutivamente equivalentes a lo largo de toda la longitud. Needleman-Wunsch desarrolló un método de alineación global utilizando programación dinámica (Needleman y Wunsch, 1970). En este método, la alineación de dos secuencias de Xi= i…m y Y j = j.. n de longitud m y n, respectivamente, implican una inicialización de la matriz de tamaño de 1 m y n 1. La primera fila y la columna son cada uno llenos de penalizaciones por hueco. Cada celda de la matriz tiene una puntuación, que se llena usando la función de la Fig. 8.

Fij es la puntuación en una celda de la matriz (i, j) es el partido en la parte superior celda diagonal. S(xi,yj) es la puntuación de aminoácidos Xi y Yj de la sustitución de la matriz como PAM o BLOSUM, u otra matriz de sustitución. d es una penalización por hueco de apertura. Los huecos se indican con "-" en una alineación y representan la inserción o deleción (in-del) caso ocurrido en una de secuencia. Ejemplo de ello se muestra en la Fig. 9, y resultado de la alineación en la Fig. 10. (Venkatarajan & Pandjassarame, 2009, pp. 51–52)

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

Figura 08: Función del Alineamiento Global

Fuente: (Durbin et al., 1998, p. 20)

		R	H	E	E	I	I	I	K	V	F	F	I
	0	-8	-16	-24	-32	-40	-48	-56	-64	-72	-80	-88	-96
H	-8	0	0	-8	-16	-24	-32	-40	-48	-56	-64	-72	-80
H	-16	-8	8	0	-8	-16	-24	-32	-40	-48	-56	-64	-72
Q	-24	-15	0	10	2	-6	-14	-22	-30	-38	-46	-54	-62
K	-32	-22	-8	2	11	3	-5	-13	-17	-25	-33	-41	-49
L	-40	-30	-16	-6	3	13	5	-3	-11	-16	-24	-32	-39
V	-48	-38	-24	-14	-5	6	16	8	0	-7	-15	-23	-29
F	-56	-46	-32	-22	-13	-2	8	16	8	0	-1	-9	-17
F	-64	-54	-40	-30	-21	-10	0	8	13	7	6	5	-3

Figura 09: Alineamiento Global de dos secuencias (RHEEIIIKVFI, HHQKLVFF) y su respectivo Traceback mostrado en oscuro.

Fuente: (Venkatarajan & Pandjassarame, 2009, p. 53)

```
RHEEIIIKVFFI
HHQKL---VFF-
```

Figura 10: Resultado del alineamiento de dos secuencias (RHEEIIIKVFI, HHQKLVFF)

Fuente: (Venkatarajan & Pandjassarame, 2009, p. 52)

Por lo tanto los datos de entrada y salida para el Algoritmo corresponden a:

Entrada: Cadenas v y w, y una matriz de puntuación δ

Salida: Un alineamiento de v y w cuyo puntaje es definido por la matriz de puntuación δ, es el máximo de todos los posibles alineamientos de v y w. (Jones & Pevzner, 2004, p. 177)

El alineamiento local es similar al alineamiento local siendo la principal diferencia en su función para el llenado de la matriz de puntuación

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

Figura 11: Función del Alineamiento Local

Fuente: (Durbin et al., 1998, p. 22)

Por lo tanto los datos de entrada y salida para el algoritmo de alineamiento local corresponden a:

Entrada: Cadenas v y w, y una matriz de puntuación δ

Salida: Un alineamiento de v y w cuyo puntaje es definido por la matriz de puntuación δ, es el máximo de todos los alineamientos globales de todas las subcadenas de v y w. (Jones & Pevzner, 2004, p. 181)

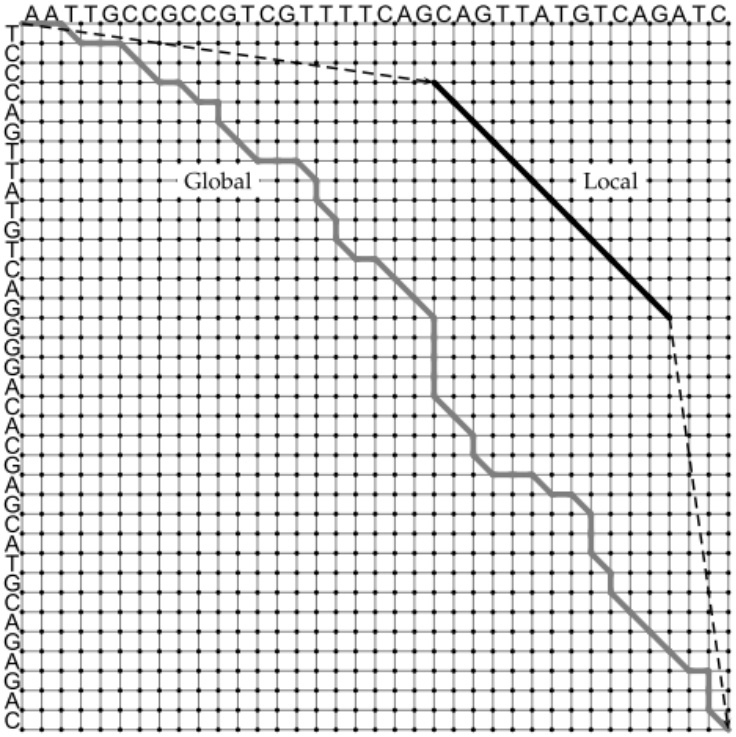

Figura 12: Alineamiento Global y Local en la matriz de scores

Fuente: (Jones & Pevzner, 2004, p. 182)

El alineamiento se realiza en 3 pasos como indica Pevsner 1.Inicialización de la matriz, 2. Llenado de la matriz de Scores, 3.Identificación del Alineamiento (Traceback)

Las Ilustraciones 13 y 14 detallan todo este proceso paso a paso. (Pevsner, 2009, pp. 76–80)

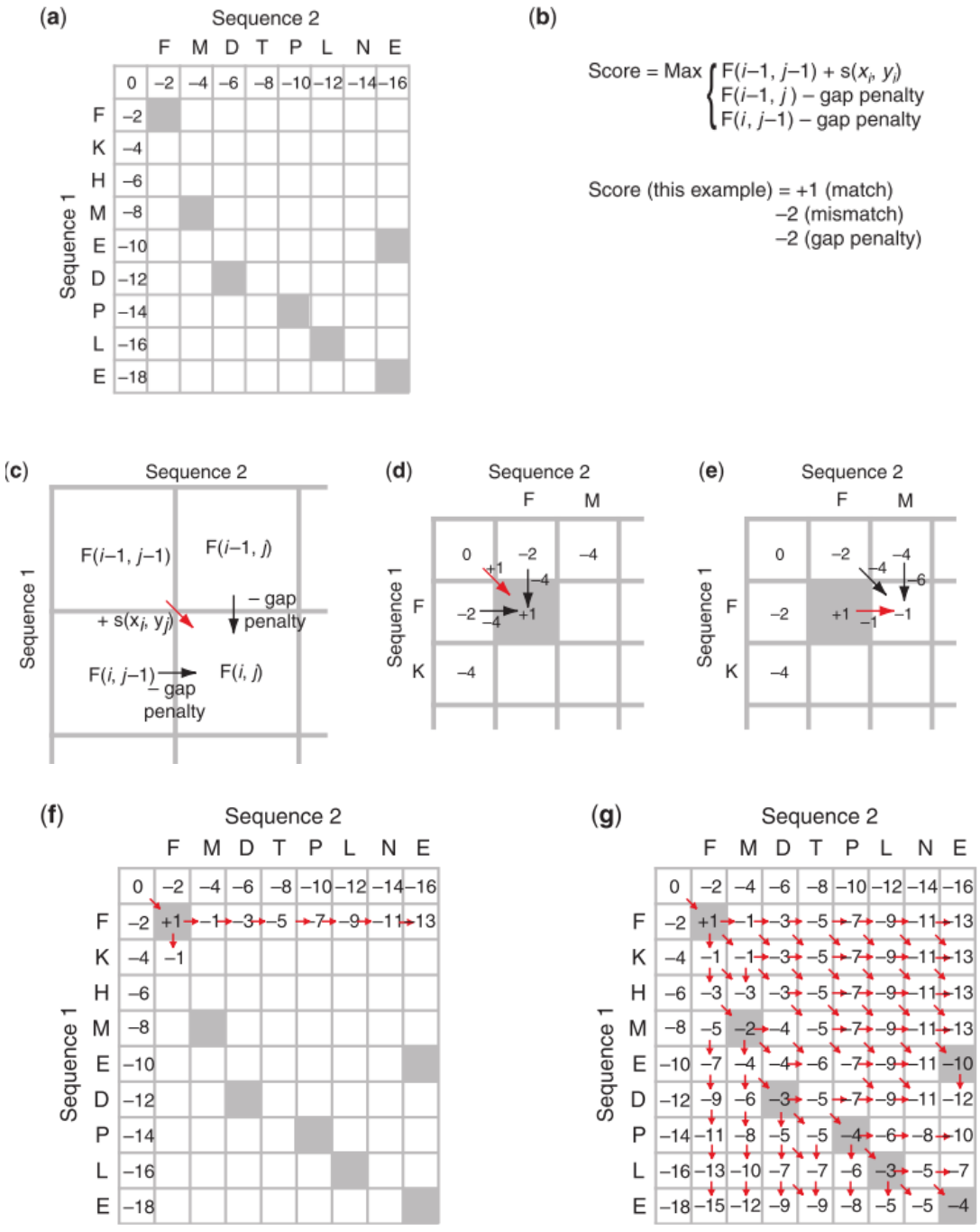

Figura 13: Procedimiento para los pasos de Inicialización y llenado de la matriz

Fuente: (Pevsner, 2009, p. 78)

En la Fig. 13 se muestra en (a) inicialización en (b) función a usar en la matriz y el puntaje respectivo de su funcion de similitud en (c) el llenado de cada celda depende de las posiciones superior, izquierda y diagonal, en (d) cálculo de la celda de la segunda fila y segunda columna en (e) Cálculo de la celda de segunda fila y tercera columna en (f) llenado de la segunda fila y en (g) matriz de scores llenas con la funcion de similitud respectiva.

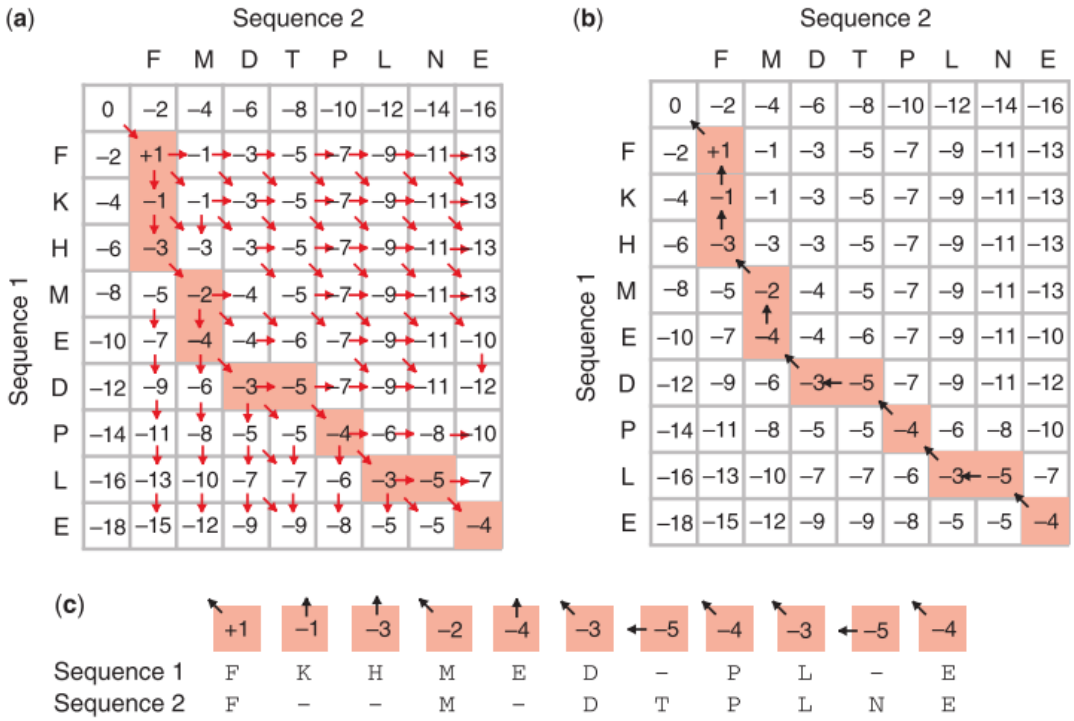

Figura 14: Paso numero tres identificación del alineamiento (Traceback)

Fuente: Fuente: (Pevsner, 2009, p. 80)

En la Fig. 14 (a) Nos muestra el origen de los optimos puntajes de cada celda (b) El recorrido del optimo puntaje para el alineamiento global (c) Las secuencias alineadas.

El pseudocódigo respectivo para los pasos 1 y 2 del alineamiento global se observa en seguidamente, y a continuación el paso 3 de Traceback.

```
Algorithm Similarity
    input: sequences s and t
    output: similarity between s and t
    m ← |s|
    n ← |t|
    for i←0 m do
    a[0,j] ← j x g
    for i ← 1 to m do
    for j ← 1 to n do
            a[i,j] ← max(a[i –I, j] + g,
                         a[i-I,j-1]+p(i,j),
                         a[i, j-1] + g)
    return a[m,n]
```

Pseudocódigo básico de la programación dinamica para la comparación de dos secuencias

Fuente: (Setubal & Meidanis, 1997, p. 52)

Algorithm Align

input: indices I, j, array a given by algorithm Similarity

output: aligment in align-s, align-t, and length in len

if i=0 **and** j=0 then

len ← 0

else if > 0 **and** a[i,j] =a[i-1,j] + g **then**

Align(i-1, j, len)

len ← len +1

align-s[len] ←s[i]

align-t[len] ← -

else if > 0 **and** j > 0 **and** a[i, j] = a[i-1, j-1] + p(i,j) **then**

Align(i-1,j-1,len)

len ← len +1

align-s[len] ← s[i]

align-t[len] ← t[j]

else // has to be j >0 **and** a[i,j] = a[I, j-1] + g

Align(i, j-1, len)

len ← len +1

align-s[len] ← -

align-t[len] ← r[j]

Pseudocódigo del algoritmo de traceback (Identificación del alineamiento).

Fuente: (Setubal & Meidanis, 1997, p. 53)

2.1.9. Software usado actualmente:

Actualmente se usan diversas herramientas informaticas para lograr el alineamiento de Secuencias como son:

SIM es un programa basado en la web para la alineación por parejas utilizando el algoritmo de Smith-Waterman que encuentra las mejores anotó nonoverlapping locales alineaciones entre dos secuencias. Es capaz de manejar decenas de kilobases de secuencia genómica. El usuario tiene la opción de establecer una matriz de puntuación para marcar la penalidad. Se producen un número especificado de alineamientos mejores.

SSEARCH es un medio simple de programas basados en la web que utiliza el algoritmo de Smith-Waterman para pairwise alineación de secuencias. Sólo se da un mejor alineamiento anotó. No hay opción al eliminar matrices o la penalización de puntuaciones.

LALIGN es un programa basado en la web que utiliza una variante del algoritmo de Smith-Waterman para alinear dos secuencias. A diferencia de SSEARCH, que devuelve la sola alineación mejor anotó, LALIGN da un número especificado de las mejores alineaciones anotadas. El usuario tiene la opción de configurar la matriz de puntuación para marcar brecha pena. La misma interfaz web también proporciona

una opción para la alineación global realizado por el programa ALIGN. (Xiong, 2006, p. 41)

2.2. Bases Teóricas del Procesamiento Paralelo

2.2.1. Importancia del procesamiento Paralelo

El propósito principal de procesamiento en paralelo es para realizar cálculos más rápidos que se puedan hacer con un único procesador mediante el uso de un número de procesadores al mismo tiempo. La consecución de este objetivo ha tenido una enorme influencia en casi todas las actividades relacionadas con la informática. La necesidad de soluciones más rápidas y para resolver los problemas de mayor tamaño se presenta en una amplia variedad de aplicaciones. Estos incluyen dinámica de fluidos, la predicción meteorológica, el modelado y simulación de sistemas grandes, el procesamiento de información y la extracción, procesamiento de imágenes, inteligencia artificial, y la fabricación automatizada.

Tres factores han contribuido a la fuerte tendencia actual a favor del procesamiento paralelo. En primer lugar, el costo de hardware ha ido disminuyendo de manera constante, por lo que ahora es posible construir

sistemas con muchos procesadores a un costo razonable. En segundo lugar, la integración de la tecnología de circuito a escala muy grande ha avanzado hasta el punto en que es posible diseñar sistemas complejos que requieren millones de transistores en un solo chip. En tercer lugar, el tiempo de ciclo más rápido de un procesador de tipo Von Neumann parece estar llegando más allá de las limitaciones físicas fundamentales que ninguna mejora es posible, además, como un mayor rendimiento es expulsado de un procesador secuencial, los aumentos de los costos asociados dramáticamente. Todos estos factores han llevado a los investigadores a explorar el paralelismo y su uso potencial en aplicaciones importantes.

Un computador paralelo es simplemente una colección de procesadores, normalmente del mismo tipo, interconectados de cierta manera para que la coordinación de sus actividades y el intercambio de datos. Los procesadores se supone que se encuentra dentro de una pequeña distancia uno de otro, y se utilizan principalmente para resolver un problema dado conjuntamente. Contraste dichos ordenadores con sistemas distribuidos, en los que un conjunto de la posibilidad de diferentes tipos de procesadores se distribuye en una amplia zona geográfica, y donde los objetivos principales son el uso de los recursos

distribuidos disponibles, y para recopilar información y transmitirla a través de una red que conecta los varios procesadores.

Los ordenadores paralelos pueden ser clasificados de acuerdo a una variedad de características arquitectónicas y modos de operaciones. En particular, estos criterios incluyen el tipo y el número de procesadores, las interconexiones entre los procesadores y los dispositivos de comunicación correspondientes, el control general y la sincronización, y las operaciones de entrada y salida.

¿Cómo se debe evaluar un algoritmo para su conveniencia para el procesamiento en paralelo? Al igual que en el caso de los algoritmos secuenciales, hay varios criterios importantes, como el rendimiento del tiempo, la utilización del espacio, y capacidad de programación. La situación de los algoritmos paralelos es más complicada debido a la presencia de parámetros adicionales, tales como el número de procesadores, las capacidades de las memorias locales, el esquema de comunicación, y los protocolos de sincronización. (JáJá, 1997, pp. 2–3)

No importa cómo son, la tecnología está desarrollando ordenadores rápidos para que sean aún más rápido. Nuestro apetito de poder de cómputo y memoria parece insaciable. Una máquina más potente lleva a nuevos tipos de aplicaciones, lo que a su vez alimenta nuestra demanda de sistemas aún más potentes. El resultado de este

continuo progreso tecnológico no es nada menos que impresionante: las computadoras portátiles un par de nosotros están utilizando para escribir este guión habría sido una de las máquinas más rápidas del planeta hace sólo una década, de haber estado presente en el momento.

Con el fin de alcanzar su velocidad de vértigo, los sistemas informáticos de hoy en día son muy complejos. Ellos se componen de múltiples componentes, o unidades funcionales, que puede ser capaz de operar simultáneamente y tienen tareas específicas, tales como la adición de dos números enteros o determinar si un valor es mayor que cero. Como resultado, un ordenador puede ser capaz de recuperar un dato de la memoria, multiplicar dos números de coma flotante, y evaluar una condición de bifurcación, todo al mismo tiempo. Se trata de un muy bajo nivel de procesamiento en paralelo y se conoce como "paralelismo a nivel de instrucción," o ILP a menudo. Un procesador que soporte este se dice que tiene una arquitectura superescalar. Hoy en día es una característica común en los microprocesadores de propósito general, incluso los que se utilizan en los ordenadores portátiles y PC.

Dadas las limitaciones de las estrategias alternativas para la creación de más potentes computadores, es probable que sea más pronunciado en el futuro próximo el uso de paralelismo en el hardware de propósito general. Algunos PCs y portátiles ya están multinúcleo o

multiproceso. Pronto, será rutinario tener procesadores con muchos núcleos y, posiblemente, la capacidad de ejecutar múltiples flujos de instrucciones dentro de cada núcleo. En otras palabras, la tecnología multinúcleo será la corriente principal. Es de vital importancia que las futuras aplicaciones puedan hacer uso efectivo del paralelismo que está presente en nuestro hardware. Pero a pesar de importantes avances en la tecnología de compilación, tendrá el programador que ayudar, mediante la descripción de la concurrencia que se encuentra en los códigos de la aplicación. (Chapman, Jost, & Van der Pas, 2008, pp. 1–3)

2.2.2. La Clase Parallel

El “For y bucles foreach" son piedras angulares del desarrollo C #, y no toma mucho tiempo para un nuevo programador para llegar al punto donde se convierten en una herramienta de uso frecuente. Las palabras clave foreach y para crear bucles secuenciales, llamada así porque una iteración no se inicia hasta que la iteración anterior se ha completado.

El TPL contiene soporte para bucles paralelos, es decir, bucles en la que las iteraciones se realizan con un cierto grado de concurrencia. Se puede ver un ejemplo sencillo e que calcula los cuadrados de 100 valores enteros. Debido a que este es un bucle paralelo, algunos de los cálculos se llevarán a cabo en paralelo.

Comenzando con bucles básicos y trabajar hasta más complejas permutaciones. Este tema es extraño, porque los bucles paralelos son una abstracción de conveniencia, construido con las características que hemos explorado en los capítulos anteriores. Una vez que se haya dominado, se encuentra bucles paralelos convenientes, fáciles de usar y, a menudo más simple que las tareas para algunos tipos de problemas de programación.

La clase Parallel: Bucles paralelos son posibles gracias a la clase System.Threading.Tasks.Parallel. No hay equivalentes en paralelo a la de

foreach y palabras clave, por lo que los bucles paralelos se realizan utilizando métodos de la clase Parallel.

Bucles paralelos, crear y administrar tareas en su nombre. Saber cómo crear y coordinar tareas, puede crear sus propios bucles paralelos para procesar los datos. Pero no es necesario, la clase System.Threading.Tasks.Parallel proporciona algunos métodos de conveniencia muy útiles que se ocupan de esto para usted.(Freeman, 2010, pp. 173–175)

2.2.3. Factor de SpeedUp

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 se define por:

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

Donde 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. En una situación ideal, para un algoritmo completamente paralelizables, y cuando el tiempo de comunicación entre

procesadores y la memoria se desprecia, tenemos Tp (N) = Tp (1) / N, y la ecuación anterior es

$$S(N) = \mathrm{N}$$

Es raro para obtener este aumento lineal en el dominio de cálculo debido a varios factores.(Gebali, 2011, pp. 15–16)

El aumento de velocidad de la definición dada anteriormente requiere una comparación con el algoritmo secuencial más rápido. Este algoritmo puede ser difícil de determinar o construir. Las posibles razones pueden ser las siguientes:

- El mejor algoritmo secuencial no se conozca. Podría ser el caso de que un límite inferior para el tiempo de ejecución de un método de solución para un problema dado se puede determinar, pero hasta ahora, ningún algoritmo con este tiempo de ejecución asintótica aún no se ha construido.
- Existe un algoritmo con el tiempo de ejecución asintótica óptima, pero dependiendo del tamaño y las características de un conjunto de entrada específica, otros algoritmos conducen a reducir los tiempos de ejecución en la práctica.
- El algoritmo secuencial que lleva a los tiempos de ejecución más pequeños requiere de un gran esfuerzo para ser implementado.

Debido a estas razones, el aumento de velocidad se calcula a menudo mediante el uso de una versión secuencial de la ejecución en paralelo en lugar del mejor algoritmo secuencial. (Rauber & Rünger, 2010, p. 160)

2.2.4. Niveles de paralelismo

Algoritmos y arquitecturas multiprocesador están estrechamente ligados. No podemos pensar en un algoritmo paralelo sin pensar en el hardware paralelo que va a apoyar. Por el contrario, no podemos pensar en hardware paralelo sin pensar en el software paralelo que conducirlo. El paralelismo se puede implementar en diferentes niveles en un sistema informático usando técnicas de hardware y software:

1. Paralelismo a nivel de datos, en los que operamos simultáneamente en varios trozos de un dato o en varios datos. Ejemplos de esto son la adición de bits en paralelo, multiplicación y división de números binarios, procesadores de vectores y matrices sistólicas para hacer frente a varias muestras de datos.
2. Paralelismo a nivel de instrucción (ILP), en la que al mismo tiempo ejecutar más de una instrucción por parte del procesador. Un ejemplo de esto es el uso de la canalización de instrucciones.

3. Paralelismo a nivel de hilo (TLP). Un hilo es una parte de un programa que comparte recursos de procesador con otros hilos. Un hilo a veces se llama un proceso ligero. En TLP, múltiples subprocesos de software se ejecutan simultáneamente en un procesador o varios procesadores.
4. Paralelismo a nivel de proceso. Un proceso es un programa que se ejecuta en el equipo. Un proceso de reserva de sus propios recursos de la computadora, como espacio de memoria y registros. Esto es, por supuesto, multitarea lo clásico y tiempo compartido de computación donde varios programas se están ejecutando de forma simultánea sobre uno máquina o en varias máquinas.

La más famosa taxonomía procesador fue propuesto por Flynn sobre la base de los datos y las operaciones realizadas en estos datos:

1. Único flujo de datos simple instrucción (SISD). Este es el caso del procesador único.
2. Instrucción simple flujo de datos múltiples (SIMD). Todos los procesadores ejecutan la misma instrucción en diferentes datos. Cada procesador tiene sus propios datos en una memoria local, y los datos de cambio procesadores entre sí a través de esquemas de comunicación típicamente simples. Muchas aplicaciones científicas y de ingeniería se prestan para el procesamiento en paralelo con este esquema. Ejemplos de tales aplicaciones incluyen el procesamiento de

gráficos, compresión de vídeo, análisis de imágenes médicas, y así sucesivamente.

3. Único flujo de datos múltiple instrucción (MISD). Se podría argumentar que las redes neuronales y máquinas de flujo de datos son ejemplos de este tipo de procesadores en paralelo.
4. Instrucción de flujo de datos múltiple Múltiple (MIMD). Cada procesador está ejecutando sus propias instrucciones sobre sus datos locales. Ejemplos de tales procesadores paralelos son procesadores multinúcleo y multihilo multiprocesadores en general.

Explorando el espacio de los ordenadores paralelos, que comprende las categorías de SIMD y MIMD, con más detalle. El tema de la sincronización entre procesadores no formaba parte de los criterios de clasificación utilizados por Flynn. En lugar de explorar esquemas de clasificación alternativos, se discute en este capítulo las diferentes arquitecturas de computadoras paralelas más comúnmente utilizados. (Smith, 1993, pp. 14–16)

Cabe destacar que el último tipo de procesador es el que se está convirtiendo en un sistema de procesamiento popular:

• Multiprocesadores de memoria compartida

• Multiprocesadores de memoria distribuida

• Los procesadores SIMD

• Procesadores sistólicos

• Computación Cluster

• La computación grid

• Los procesadores multinúcleo

• Transmisión multiprocesador

(Gebali, 2011, pp. 53–54)

2.2.5. Reglas para programar en paralelo para Multinúcleo

Programación de procesadores multinúcleo plantea nuevos desafíos. Aquí hay ocho reglas para la programación multinúcleo para ayudarle a tener éxito:

- Pensar en paralelo. Enfocar todos los problemas buscando el paralelismo. Entender dónde está el paralelismo, y organizar la forma de pensar para expresarlo. Decidir el mejor enfoque paralelo antes de tomar otras decisiones de diseño o implementación. Aprender a “Pensar en Paralelo”
- Programar usando la abstracción. Concentrarse en escribir código para expresar el paralelismo, pero evitar escribir código para gestionar hilos o núcleos de procesador. Las bibliotecas, OpenMP, e Intel Threading Building Blocks son todos ejemplos del uso de

abstracciones. No usar hilos nativos en bruto (pthreads, Windows threads, Boost threads, y cosas por el estilo). Los hilos y MPI son los lenguajes de ensamblaje para el paralelismo. Ofrecen máxima flexibilidad, pero requieren demasiado tiempo para escribirlos, depurarlos y mantenerlos. Nuestra programación debería tener el suficiente nivel como para que nuestro código sea sobre nuestro problema y no sobre la gestión de los hilos o de los núcleos.

- Programar pensando en tareas (cometidos), y no en hilos (núcleos). Dejar el mapeado de tareas a hilos o a núcleos de procesador como una operación claramente separada en nuestro programa, preferentemente como una abstracción en uso, que se encarga de controlar la gestión de hilos/núcleo. Crear una abundancia de tareas en el programa, o una tarea que pueda ser extendida por los núcleos del procesador automáticamente (como por ejemplo un bucle OpenMP). Al crear tareas, se pueden crear cuantas se puedan sin tener que preocuparse de la suscripción excesiva.
- Diseñar con la opción de desconectar la concurrencia. Para hacer más sencillo el depurado, crear programas que puedan funcionar sin concurrencia. De esta forma, mientras se depura, es posible ejecutar programas primero con concurrencia y después sin ella

para ver si las dos ejecuciones fallan o no. El depurado de cuestiones comunes es más sencillo cuando el programa no está funcionando de manera concurrente, porque resulta más familiar y está mejor soportado por las herramientas actuales. Saber que algo falla sólo cuando se ejecuta de forma concurrente indica el tipo de fallo técnico que se busca. Si se ignora esta regla y no se puede obligar al programa a funcionar en sólo un hilo, se pasará demasiado tiempo depurando. Puesto que se quiere tener la capacidad de ejecutar en un solo hilo específicamente para el depurado, no hay necesidad de que sea eficiente. Sólo hay que evitar crear programas paralelos que necesiten de la concurrencia para funcionar correctamente, como muchos modelos de productor-consumidor. Los programas MPI a menudo violan esta regla, que es parte del motivo por el que los programas MPI pueden ser problemáticos de implementar y de depurar.

- Evitar el uso de bloqueos. Sencillamente hay que decir “no” a los bloqueos. Los bloqueos ralentizan los programas, reducen su escalabilidad, y son fuente de fallos técnicos en los programas paralelos. Hacer de la sincronización implícita la solución al programa. Cuando se necesite aún la sincronización explicita, hay que usar operaciones atómicas. Usar bloqueos sólo como el último

recurso. Hacer todo lo posible por diseñar que la necesidad de los bloqueos quede totalmente fuera del programa.

- Usar herramientas y bibliotecas diseñadas para ayudar con la concurrencia. No “aguantar” con herramientas viejas. Ser críticos con el soporte de herramientas en lo que respecta a cómo presenta e interactúa con el paralelismo. La mayoría de las herramientas no están todavía preparadas para el paralelismo. Buscar bibliotecas seguras en los hilos – idealmente las que están diseñadas para utilizar ellas mismas el paralelismo.
- Utilizar asignadores escalables de memoria . Los programas hilados necesitan usar asignadores escalables de memoria. Período. Hay una serie de soluciones y me da la impresión de que todas ellas son mejores que malloc(). El uso de asignadores escalables de memoria acelera las aplicaciones al eliminar los cuellos de botella globales, al reutilizar la memoria dentro de los hilos para usar mejor los caché y al dividir de la forma adecuada para evitar compartir línea de caché.
- Diseñar para escalar a través de mayores cargas de trabajo. La cantidad de trabajo que el programa necesita gestionar aumenta con el tiempo. Hay que planificar para ello. Si se diseña con la escalada en mente, el programa gestionará más trabajo según

aumenta el número de núcleos de procesador. Cada año, pedimos que nuestros ordenadores hagan cada vez más cosas. Los diseños deberían favorecer el uso de aumentos en el paralelismo para ofrecer ventajas en la gestión de mayores cantidades de trabajo en el futuro. (Hillar, 2010, p. 22)

III. DESARROLLO

3.1. Diseño de Algoritmos Propuestos

Al existir una fuerte dependencia en el llenado de la matriz de scores (ver 2.1.8), se tuvo que reformular el algoritmo para el llenado de antidiagonal en antidiagonal, observemos como es el comportamiento de una matriz no cuadrada. En la Fig. 15 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 15: 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 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.

3.2. 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 Fig. 16.

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 16: Matriz con las posiciones i y j concatenadas

Fuente: Elaboración propia.

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 Fig. 17.

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 17: 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 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 Fig. 20 se observa esta situación.

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

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

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 Fig. 18, 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

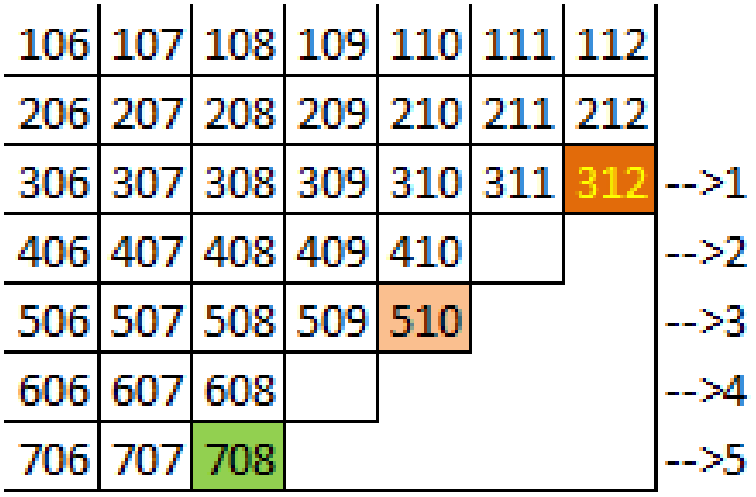

Figura 18: Matriz de scores con posiciones operadas en un instante de tiempo por 3 procesadores.

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 Fig. 19.

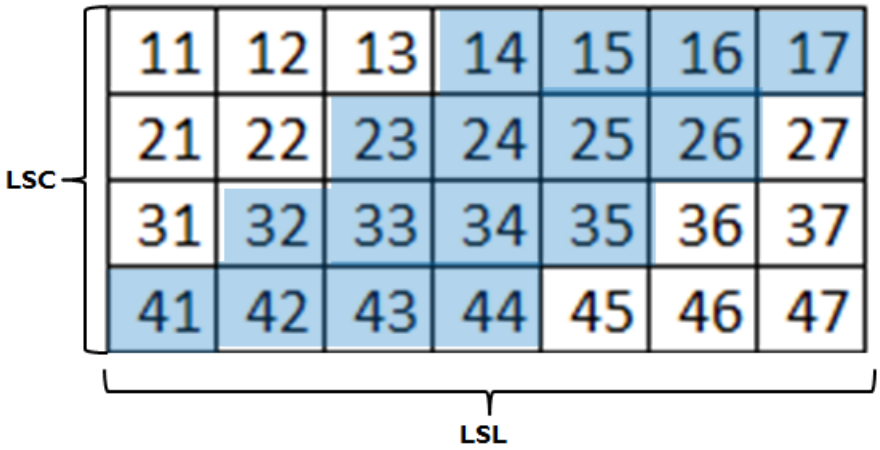

Figura 19: Matriz de scores con posiciones operadas en un instante de tiempo por 3 procesadores.

Fuente: Elaboración propia.

3.3. 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.

Algoritmo 1 Almacenar (i, VIA, N)
Fuente: Elaboración propia
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)

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 números, “s” es la función de similitud de ambas secuencias, “gap” es la penalidad asignada a los huecos.

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.

Algoritmo 2 Paso 2 para alineamiento de secuencias con programación Dinámica
Fuente: (Setubal & Meidanis, 1997, p. 52)

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 H**acer**

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 el apartado 3.2, para lo cual se elaboró el algoritmo 3 en la cual se usara 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:

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.

Fuente: Elaboración Propia

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 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.

3.4. 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=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 Fig. 20.

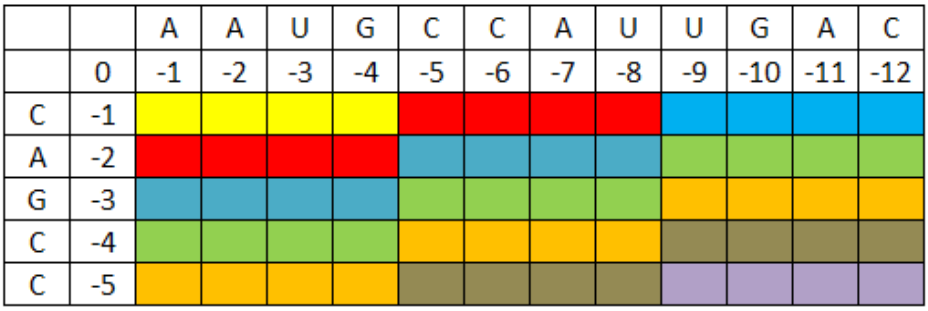

Figura 20: Distribución de los bloques de antidiagonales de la matriz de scores para cuatro procesadores.

Fuente: Elaboración propia.

En la Fig. 21 los colores idénticos nos indican los bloques de antidiagonales que realizaran 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 Fig. 21

	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 21: 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 Fig. 22 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 o 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 Fig. 22, 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.

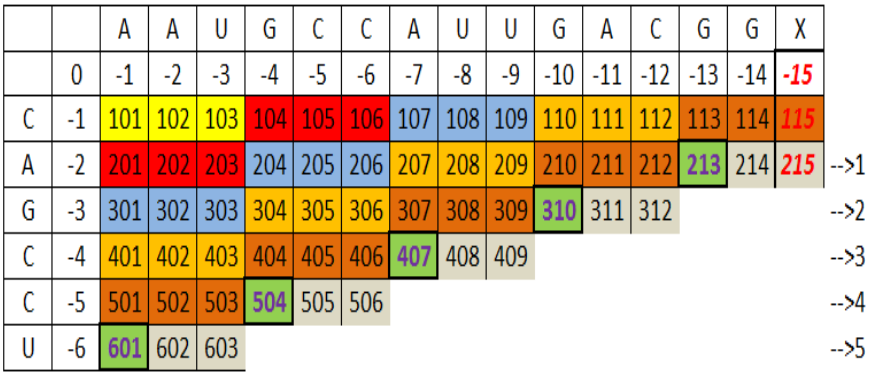

Figura 22: Proceso de llenado en bloques, en verde el valor inicial de cada bloque de la antidiagonal.

Fuente: Elaboración propia.

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 3.2.

3.5. 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)

Fuente: Elaboración propia

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.

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).

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

Fuente: Elaboración propia.

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)

3.6. Implementación

El algoritmo propuesto ha sido implementado 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, 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.

En la Fig. 23 se observa la interfaz de la aplicación final con los tiempos obtenidos usando StopWatch.

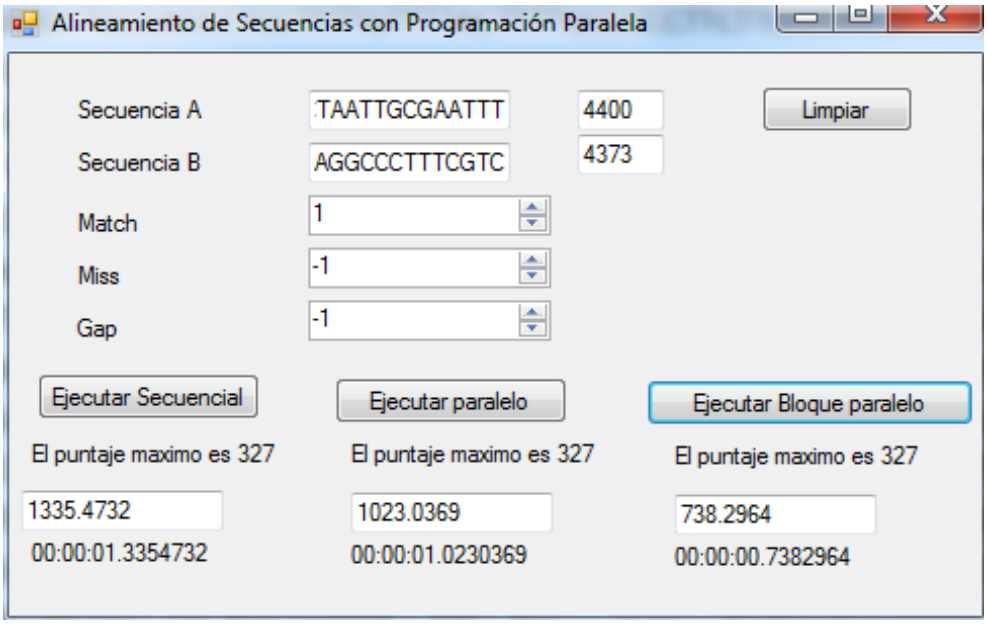

Figura 23: Interfaz de la apicación de alineamiento de secuencias con los tiempos empleados.

Fuente: Elaboración propia.

IV. RESULTADOS

4.1. Descripción de la obtención de los resultados

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 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, de tamaños aproximados entre 400 a 12 000 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 el Cuadro 2, 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 el Capítulo III se describen en el Cuadro 2.

Cuadro 2: 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		
400	7.6145	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	0.7864	1.8462
2000	220.0160	223.0678	118.7799	0.9863	1.8523
2400	350.1218	305.6356	201.7435	1.1456	1.7355
2800	483.9285	360.6928	314.2978	1.3417	1.5397
3200	641.2658	504.6694	461.6450	1.2707	1.3891
3600	825.3279	700.5314	545.1521	1.1781	1.5139
4000	1025.6548	740.7942	658.3604	1.3845	1.5579
4400	1261.6876	974.3385	688.2793	1.2949	1.8331
4800	1486.6998	1004.0549	862.1963	1.4807	1.7243
5200	1656.3646	1038.4770	938.9871	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	3305.9656	1883.9729	1573.5864	1.7548	2.1009
7600	3794.2351	2116.3246	1768.1129	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	6570.8378	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	4617.8235	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 el Gráfico 1.

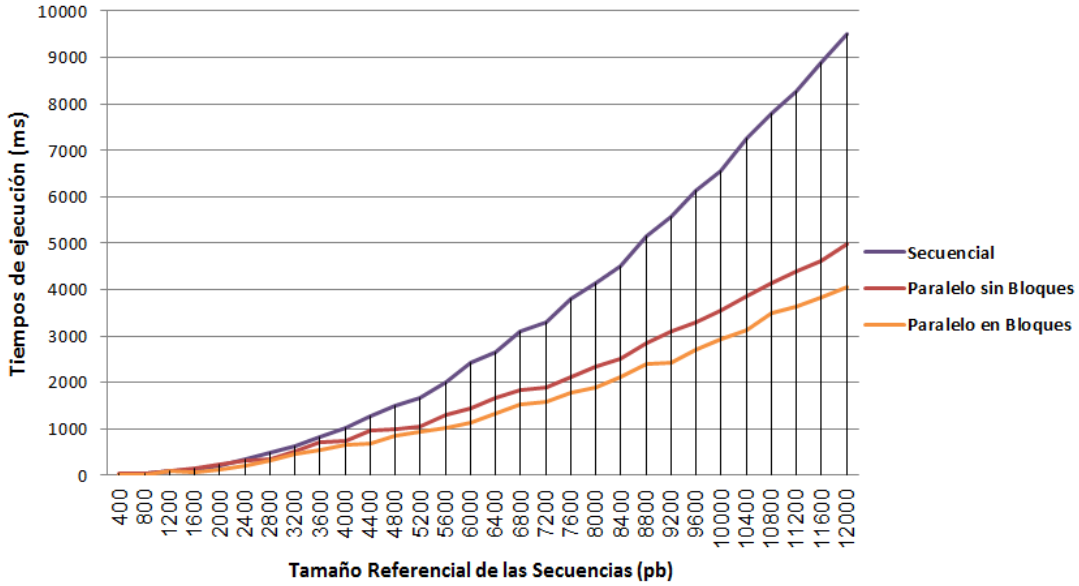

Gráfico 1 Tiempos de ejecución para el alineamiento de diferentes secuencias con los procesamientos secuencial y paralelo.

Fuente. Elaboración Propia.

Se puede apreciar en el Gráfico 2 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.

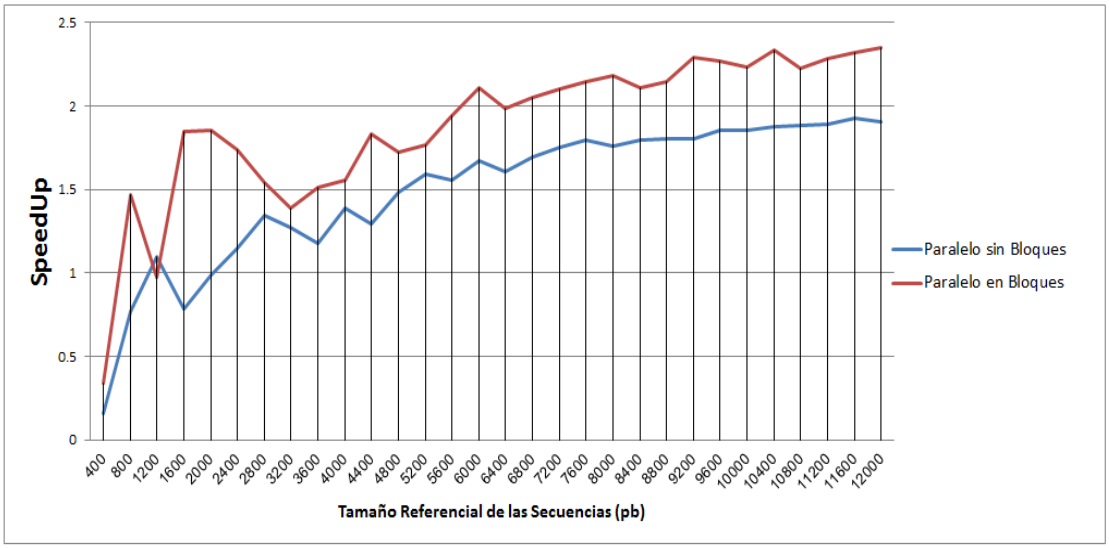

Gráfico 2 SpeedUp de la implementación paralela para los diferentes tamaños referenciales de secuencias.

Fuente. Elaboración propia

4.2. Contraste de Hipótesis

Para el contraste de Hipótesis se ha usado el análisis de regresión lineal. El análisis de regresión es una técnica para investigar y modelar la relación entre variables. Aplicaciones de regresión son numerosas y ocurren en casi todos los campos, incluyendo ingeniería, la física, ciencias económicas, ciencias biológicas y de la salud, como también ciencias sociales.

4.2.1. Realizando Cálculo de la línea de mejor ajuste

Se ha procedido ajustando los datos obtenidos a una recta, para esto se ha usado la Hoja de Cálculo de Microsoft Excel 2010 y obtención de la línea de mejor ajuste (Llamada en Excel línea de tendencia) como se aprecia en el Gráfico 3.

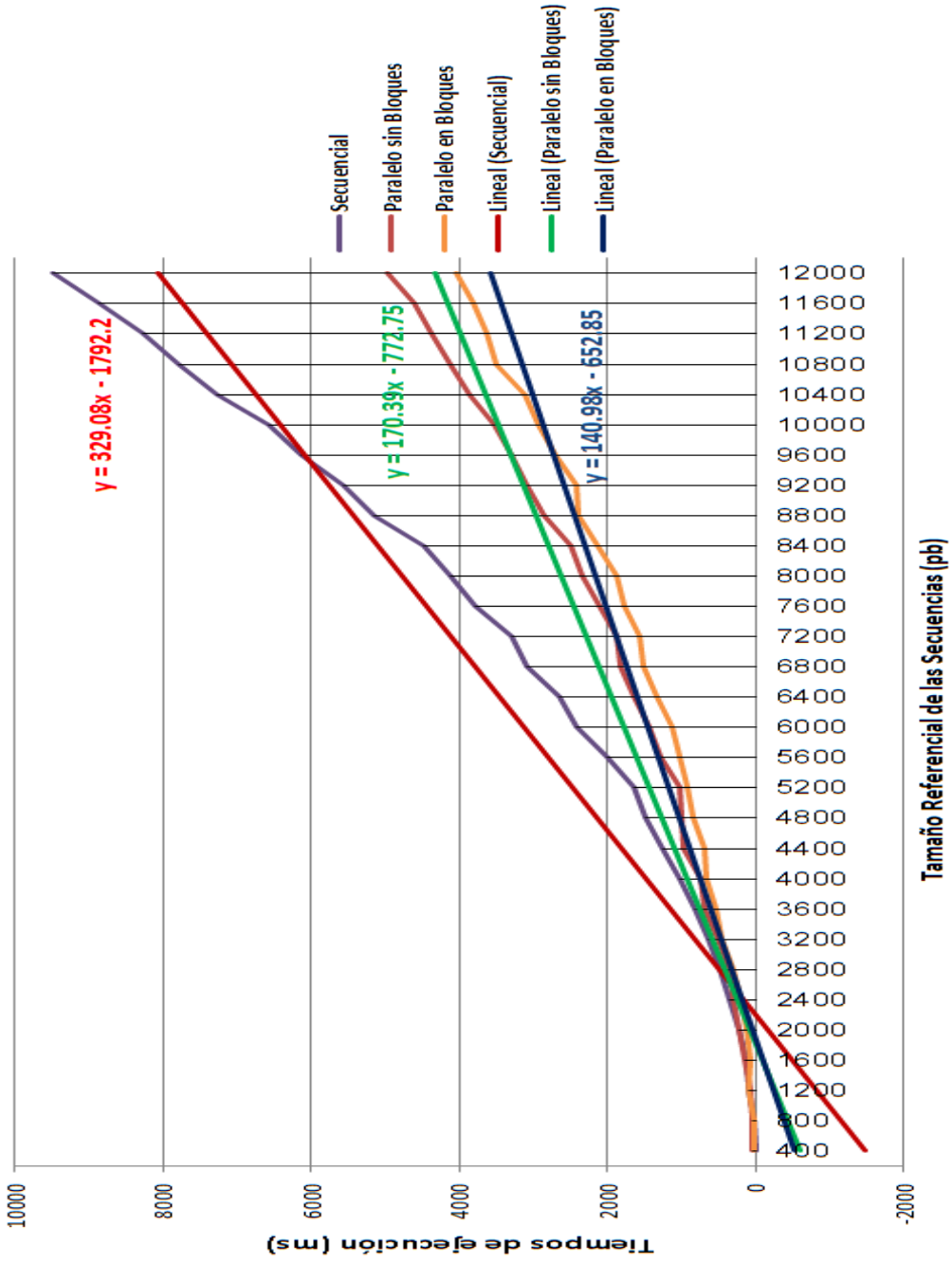

Gráfico 3 Comparación de los tiempos de ejecución con sus respectivas líneas de tendencia.

Fuente: Elaboración propia

Siendo las ecuaciones de las rectas (línea de tendencia) que mejor se ajustan para cada algoritmo datos obtenidos por el Microsoft Excel 2010 los siguientes:

Para el algoritmo secuencial:

Y1 = 329.08x - 1792.2

Y su pendiente m_1=329.08

Para el algoritmo paralelo sin bloques:

Y2 = 170.39x - 772.75

Y su pendiente m_2=170.39

Para el algoritmo paralelo con bloques:

Y3 = 140.98x - 652.85

Y su pendiente m_3=140.98

Por lo tanto m1>m2>m3

4.2.2. Interpretación

Recordemos la definición de la pendiente:

"La pendiente m de una recta no vertical L mide el cambio en y cuando x cambia de x1 a x2. Esto se llama razón de cambio promedio de y respecto a x." (Sulivan & Hernández, 2006, p.182)

“La pendiente es en realidad una razón de cambio. Cuando nos movemos de un punto en una recta a otro, la pendiente nos dice cuánto cambia y en comparación con cuánto cambia x”. (Goodman & Hirsch, 1996 , p. 108)

Siendo X la longitud de la secuencia y Y el tiempo de respuesta, y conociendo que m1>m2>m3, siendo m1 la pendiente del algoritmo secuencial, m2 la pendiente del algoritmo paralelo sin bloques, y m3 algoritmo paralelo con bloques, se observa que en el algoritmo Secuencial tiene hay una mayor razón de cambio con respecto a los algoritmos paralelos propuestos, entonces se concluye que ambos algoritmos propuestos tanto el algoritmo paralelo con bloques y el sin bloques presentan un menor tiempo de respuesta a medida que las longitudes de las secuencias se incrementan dado que presentan una menor pendiente con respecto al algoritmo secuencial.

V. DISCUSIONES

PRIMERA

Se observó el problema que existe al realizar un alineamiento de pareado de secuencias biomoleculares ya que se usa algoritmos secuenciales desaprovechando el potencial de todos los núcleos para realizar esta tarea, autores anteriores referían a máquinas especializadas para realizar esta tarea, se realizó el experimento en un computador personal haciendo uso de sus 8 núcleos para el experimento.

SEGUNDA

El manejo algorítmico del paradigma paralelo es poco difundido por diversos autores, el presente trabajo muestra un desarrollo detallado algorítmico del paradigma paralelo, desde su diseño hasta su implementación, mostrando la ventaja del paradigma paralelo.

TERCERA

Para comprobar si la propuesta del algoritmo paralelo reduce los tiempos, se compararon los tiempos con el algoritmo secuencial y luego se usó un análisis de regresión, calculando la línea de mejor ajuste y comparando las pendientes respectivas, verificando que la pendiente del algoritmo secuencial es mayor que la del algoritmo paralelo propuesto, teniendo el tiempo una mayor razón de cambio a medida que el tamaño de secuencia se incrementa.

CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES

PRIMERA

Se ha determinado que la propuesta del algoritmo paralelo para el alineamiento pareado de secuencias biomoleculares, si reduce el tiempo de respuesta, disminuyendo los tiempos en más de un 50%, utilizando todos los núcleos del computador.

SEGUNDA

Se implementó la propuesta del algoritmo paralelo para el alineamiento de pareado de secuencias biomoleculares para la fase 2 del alineamiento de secuencias, usando el paradigma paralelo implementándolo en un computador multinúcleo, los resultados de los tiempos de respuesta los podemos apreciar en el CAPITULO IV.

TERCERA

Se ha determinado que la razón de cambio del tiempo con respecto al tamaño de la secuencia en el algoritmo secuencial es mayor con respecto a la propuesta del algoritmo paralelo estos resultados encontrados en el CAPITULO IV.

RECOMENDACIONES

PRIMERA

Se recomienda realizar más propuestas con el paradigma paralelo para los diferentes algoritmos secuenciales que aún se usan en la biología computacional, para reducir los tiempos de respuesta y usando todos los núcleos que ahora podemos contar.

SEGUNDA

Se recomienda usar esta propuesta algorítmica para el alineamiento múltiple de secuencias biomoleculares o para la construcción de árboles filogenéticos.

TERCERA

Se recomienda usar la GPGPU para usar los multiprocesadores gráficos que ahora se cuentan para realizar implementación de esta propuesta algorítmica.

REFERENCIAS BIBLIOGRÁFICAS

Callisaya, W., Callohuari, R., & Jimenez, J (2013a). Programación Paralela para el Alineamiento de Secuencias de ADN usando Task Parallel Library (TPL). Proceedings of the 20th Intercon, pp. 156-162.

Callisaya, W.,& Callohuari, R. (2013b). Modelo de un algoritmo eficiente para el alineamiento de secuencias biomoleculares basado en programación paralela. Memoria COMTEL 2013, pp. 158-166.

Chapman, B., Jost, G., & Van der Pas, R. (2008). *Using OpenMP: portable shared memory parallel programming*. (M. Press, Ed.) (p. 353). London, England.

Durbin, R., Eddy, S., Krogh, A., & Mitchison, G. (1998). *Biological Sequence Analysis: Probabilistic models of proteins and nucleic acids*. (C. U. Press, Ed.) (p. 356). New York, United States.

Freeman, A. (2010). *Pro .NET 4 Parallel Programming in C#*. (Apress, Ed.) (p. 328). Berkeley, CA: Apress. doi:10.1007/978-1-4302-2968-1

Freeman, DH (1987) Applied categorical data analysis. New York. Marcel Dekker Inc.

Gebali, F. (2011). *Algorithms and Parallel Computing*. (J. Wiley & Sons, Ed.) (p. 364). New Jersey, United States of America.

Gibas, C., & Reilly, P. O. (2001). *Developing Bioinformatics Computer Skills*. (O'Reilly, Ed.) (1ra Editio., p. 446). United States Of America.
Goodman, Arthur, Hirsch, Lewis (1996). Algebra y trigonometría con geometría analítica. Pearson Educación.

Grama, A., Gupta, A., Karypis, G., & Kumar, V. (2003). *Introduction to Parallel Computing* (2nd Editio., p. 856). United States of America: Addison Wesley.

Gusfield, D. (1997). *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. (C. U. Press, Ed.) (1st editio., p. 534). New York, United States.

Henikoff, S., & Henikoff, J. G. (1992). Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences of*

the United States of America, *89*(22), 10915–9. Retrieved from http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=50453&tool=pmcentrez&rendertype=abstract

Higgins, D., & Taylor, W. (2000). *Bioinformatics: Sequence, Structure and Databanks*. (O. U. Press, Ed.)*Briefings in Bioinformatics* (Vol. 2, p. 249). New York, United States. doi:10.1093/bib/2.2.202

Hillar, G. C. (2010). *Professional Parallel Programming with C#: Master Parallel Extensions with .NET 4*. (J. W. & Sons, Ed.) (p. 576). United States of America.

JáJá, J. (1997). *An Introduction to Parallel Algorithms*. (A. Wesley, Ed.) (p. 566). United States of America.

Jones, N. C., & Pevzner, P. A. (2004). *An Introduction to bioinformatics algorithms*. (M. Press, Ed.) (p. 435). London, England.

Kesmir, C. (2013). Bioinformatics. Retrieved from http://theory.bio.uu.nl/BPA/tb.pdf

Khajeh-Saeed, A., Poole, S., & Blair Perot, J. (2010). Acceleration of the Smith–Waterman algorithm using single and multiple graphics processors. *Journal of Computational Physics*, *229*(11), 4247–4258. doi:10.1016/j.jcp.2010.02.009

Lesk, A. M. (2002). *Introduction to Bioinformatics*. (O. U. Press, Ed.) (p. 283). New York, United States: Oxford University Press.

Mount, D. (2001). *Bioinformatics: Sequence and Genome Analysis*. (C. S. H. L. Press, Ed.) (p. 564). New York, United States.

Nawaz, Z., Nadeem, M., Someren, H. Van, & Bertels, K. (2010). A parallel FPGA design of the Smith-Waterman traceback. In J. Bian, Z. Qiang, P. Athanas, Y. Ha, & K. Zhao (Eds.), *Proc. International Conference on Field-Programmable Technology* (pp. 454–459). Beijing, China: IEEE. Retrieved from http://ebookbrowse.com/220-a-parallel-fpga-design-of-the-smithwaterman-traceback-pdf-d407783147

Pevsner, J. (2009). *Bioinformatics and Functional Genomics*. (r J. W. & Sons, Ed.) (2nd Editio., p. 897). New York, United States.

Rauber, T., & Rünger, G. (2010). *Parallel Programming: For Multicore and Cluster Systems*. (Springer, Ed.) (p. 455). Berlin, Germany.

Setubal, J., & Meidanis, J. (1997). *Introduction to Computational Molecular Biology*. (P.-K. P. Company, Ed.) (p. 296). Boston, United States.

Shehab, S. A., Keshk, A., & Mahgoub, H. (2012). Fast Dynamic Algorithm for Sequence Alignment based on Bioinformatics. *International Journal of Computer Applications*, *37*(7), 54–61. Retrieved from http://research.ijcaonline.org/volume37/number7/pxc3876636.pdf

Smith, J. R. (1993). *The Design and Analysis of Parallel Algorithms*. (O. U. Press, Ed.) (p. 510). New York, United States.

Sullivan J., Hernández Garciadiego Carlos (2006). Algebra y Trigonometria. Pearson Educación.

Venkatarajan, S. M., & Pandjassarame, K. (2009). *Bioinformatics: A Concept-Based Introduction*. (Springer, Ed.) (p. 184). New York, United States.

Xiong, J. (2006). *Essential Bioinformatics*. (C. U. Press, Ed.) (p. 339). New York, United States.

Zomaya, A. Y. (2006). *Parallel Computing for Bioinformatics and Computational Biology*. (J. Wiley & Sons, Ed.) (p. 1000). New Jersey, United States of America.

VI. ANEXOS

ANEXO 1 Matriz de Consistencia

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

<table>
<tr><td>Problema</td><td>Objetivo</td><td>Hipótesis</td><td>Variables</td></tr>
<tr><td>Problema General</td><td>Objetivo General</td><td>Hipótesis General</td><td>V Independiente</td></tr>
<tr><td>¿En qué medida un algoritmo paralelo mejora el desempeño del alineamiento de pares de secuencias biomoleculares?</td><td>Diseñar un algoritmo paralelo para el alineamiento de pares de secuencias biomoleculares.</td><td rowspan="3">H1: Con el algoritmo paralelo propuesto para alinear secuencias biomoleculares se reduce el tiempo de respuesta respecto al secuencial

Ho: Con el algoritmo paralelo para alinear secuencias biomoleculares no se reduce el tiempo de respuesta respecto al secuencial.</td><td>Algoritmo paralelo.

Indicadores

Tiempo de ejecución
Speed Up</td></tr>
<tr><td>Problema Especifico</td><td>Objetivo Especifico</td><td>Variable Dependiente</td></tr>
<tr><td>• ¿Cómo funciona el Algoritmo que se usa para el alineamiento de pares de secuencias biomoleculares tradicional?
• ¿Cómo diseñar un algoritmo paralelo para el alineamiento de secuencias biomoleculares?</td><td>• Conocer el funcionamiento del algoritmo de alineamiento pares de secuencias biomoleculares que se usa actualmente.
• Proponer el diseño de un algoritmo paralelo para alineamiento de secuencias biomoleculares.</td><td>Proceso de alineamiento de pareado de secuencias biomoleculares.

Indicadores

Tiempo de ejecución.</td></tr>
</table>

ANEXO 2: Glosario de Términos

ADN: Acido Desoxirribunucleico, lo conforman los nucleótidos A, C, T, G correspondientes a Adenina, Citosina, Timina y Guanina.

ALGORITMO: Un conjunto preescrito de instrucciones o reglas bien definidas, ordenadas y finitas que permite realizar una actividad mediante pasos sucesivos que no generen dudas a quien deba realizar dicha actividad.

ALINEAMIENTO DE SECUENCIAS: Proceso en el cual se buscan las similitudes de dos secuencias de caracteres.

ALINEAMIENTO GLOBAL: Proceso por el cual se realiza un alineamiento de todos los caracteres de dos secuencias.

ALINEAMIENTO LOCAL: Proceso por el cual se busca un subalineamiento optimo entre dos secuencias.

AMINOÁCIDOS: Unidades básicas de las Proteínas, para el alineamiento solo se toman los 20 aminoácidos esenciales.

ARN: Ácido Ribonucleico, lo conforman los nucleótidos A, C, U, G correspondientes a Adenina, Citosina, Uracilo y Guanina.

BIOINFORMÁTICA: Usa de herramientas computacionales para ayudar a resolver problemas biológicos.

BIOLOGÍA COMPUTACIONAL: Es el uso de algoritmos y ordenadores para facilitar el entendimiento de problemas biológicos. La biología computacional abarca varios campos ya establecidos: química, bioquímica, matemáticas, ingeniería de sistemas, física, estadísticas, etc.

DOGMA CENTRAL: Es el proceso por el cual el ADN es transcrito a ARN y luego traducido a Proteínas.

GRANULO FINO: Si sus subtareas deben comunicase muchas veces por segundo.

GRANULO GRUESO: Si no se comunican muchas veces por segundo.

OPENMP: Es una interfaz de programación de aplicaciones (API) para la programación multiproceso de memoria compartida en múltiples plataformas.

PROGRAMACIÓN DINÁMICA: Es un método para reducir el tiempo de ejecución de un algoritmo mediante la utilización de subproblemas superpuestos y subestructuras óptimas.

PROGRAMACIÓN PARALELA: Es una forma de cómputo en la que muchas instrucciones se ejecutan simultáneamente, operando sobre el principio de que problemas grandes, a menudo se pueden dividir en unos más pequeños, que luego son resueltos simultáneamente (en paralelo).

PROTEÍNA: Conjunto de Aminoácidos conforman una proteína que tiene su estructura y funciones específicas en todos los seres vivos.

SIMD: Simple Instrucción y Múltiples Datos, refiere que una orden es aplicada a muchos datos en forma simultánea.

SPEEDUP: Incremento de la Aceleración medida con la que se ve el rendimiento de la programación paralela con la Secuencial.

TIEMPO DE EJECUCIÓN: Lapso en el cual el computador se demora en resolver un problema.

TPL: Librería de programación de Tareas. Task Parallel Library, difundido por Microsoft.

ANEXO 3: Código realizado para la implementación del algoritmo

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Threading;
using System.Threading.Tasks;
using System.IO;
using System.Net;
using System.Diagnostics;
namespace AlineamientoParalelo
{
    public partial class BloquesParalelo : Form
    {
        long tb,lb;
        string sc, sl;
        long lsc, lsl, npn;
        long[,] f;

        long Gap;
        Stopwatch tiempo;
        TimeSpan dt;
        public BloquesParalelo()
        {
            InitializeComponent();
        }
        private void btnParalelo_Click(object sender, EventArgs e)
        {
            tiempo = Stopwatch.StartNew();
            if (this.TxtSecA.Text.Length >= this.TxtSecB.Text.Length)
            {
                sl = this.TxtSecA.Text;
                sc = this.TxtSecB.Text;
                lsc = this.TxtSecB.Text.Length;
                lsl = this.TxtSecA.Text.Length;
            }
            else
            {
                sl = this.TxtSecB.Text;
                sc = this.TxtSecA.Text;
                lsc = this.TxtSecA.Text.Length;
                lsl = this.TxtSecB.Text.Length;
            }
            f = null;
            f = new long[lsc + 1, lsl + 1];
            matrizPDIAG();
            tiempo.Stop();
            dt = tiempo.Elapsed;
            this.lbltpar.Text = dt.TotalMilliseconds.ToString();
            this.lblTpar1.Text = dt.ToString();
            this.lbllongseca.Text = this.TxtSecA.Text.Length.ToString();
            this.lbllongsecb.Text = this.TxtSecB.Text.Length.ToString();
        }
        private void matrizPDIAG()
```

```
        {
            int tda, da, num, centi, j;
            long aux, iM;
            aux = lsl;
            num = 1;
            while (aux != 0) { aux = aux / 10; num = num * 10; }
            centi = 0;
            tda = 1;
            for (j = 0; j <= lsc; j++)
                f[j, 0] = j * (long)this.NumGap.Value;
            for (j = 0; j <= lsl; j++)
                f[0, j] = j * (long)this.NumGap.Value;
            iM = num + 1;
            for (da = 1; da <= lsc + lsl - 1; da++)
            {
                Parallel.For(0, tda, i =>
                {
                    almacenar(i, iM, num);
                });
                if (tda < lsc && centi == 0)
                {
                    tda = tda + 1;
                    iM = iM + 1;
                }
                else
                {
                    centi = 1;
                    if (da < lsl)
                    {
                        iM = iM + 1;
                    }
                    else
                    {
                        tda = tda - 1;
                        iM = iM + num;
                    }
                }
            }
            this.lblparalelo.Text = "El puntaje maximo es " + f[lsc,
lsl].ToString();
        }
        private static long max(long x, long y, long z)
        {
            long maxi;
            if (x > y)
            {
                if (x > z)
                    maxi = x;
                else maxi = z;
            }
            else
            {
                if (y > z)
                    maxi = y;
                else
                    maxi = z;
            }

            return maxi;
        }
```

```
        private long s(long x, long y)
        {
            char a, b;
            a = sc[(int)x - 1];
            b = sl[(int)y - 1];
            if (a == b)
                return (long)this.NumMatch.Value;
            else
                return (long)this.NumMiss.Value;
        }
        private void almacenar(int i, long iM, int num)
        {
            int X, Y;
            long aux;
            aux = iM + (num - 1) * (i);
            X = (int)(aux / num);
            Y = (int)(aux % num);

            f[X, Y] = max(f[X - 1, Y] + (long)this.NumGap.Value, f[X - 1, Y
- 1] + s(X, Y), f[X, Y - 1] + (long)this.NumGap.Value);
        }
        private void matrizpunt()
        {
            long i, j;
            Gap = (long)this.NumGap.Value;
            for (i = 0; i <= lsc; i++)
                f[i, 0] = i * Gap;
            for (i = 0; i <= lsl; i++)
                f[0, i] = i * Gap;
            for (i = 1; i <= lsc; i++)
                for (j = 1; j <= lsl; j++)
                    f[i, j] = max(f[i - 1, j] + Gap, f[i - 1, j - 1] + s(i,
j), f[i, j - 1] + Gap);
            this.lblresp.Text = "El puntaje maximo es " + f[lsc,
lsl].ToString();
        }
        private void btnLimpiar_Click(object sender, EventArgs e)
        {
            this.TxtSecA.Text = "";
            this.TxtSecB.Text = "";
        }
        private void btnEjecutar_Click(object sender, EventArgs e)
        {
            tiempo = Stopwatch.StartNew();
            matrizpunt();
            tiempo.Stop();
            dt = tiempo.Elapsed;
            this.lbltsec.Text = dt.TotalMilliseconds.ToString();
            this.lbltsec1.Text = dt.ToString();
            this.lbllongseca.Text = this.TxtSecA.Text.Length.ToString();
            this.lbllongsecb.Text = this.TxtSecB.Text.Length.ToString();
        }
        private void btnBloqParalelo_Click(object sender, EventArgs e)
        {
            tiempo = Stopwatch.StartNew();
            if (this.TxtSecA.Text.Length >= this.TxtSecB.Text.Length)
            {
                sl = this.TxtSecA.Text;
                sc = this.TxtSecB.Text;
                lsc = this.TxtSecB.Text.Length;
```

```
            lsl = this.TxtSecA.Text.Length;
        }
        else
        {
            sl = this.TxtSecB.Text;
            sc = this.TxtSecA.Text;
            lsc = this.TxtSecA.Text.Length;
            lsl = this.TxtSecB.Text.Length;
        }
        if (Environment.ProcessorCount > lsc)
        {
            f = null;
            f = new long[lsc + 1, lsl + 1];

            matrizPDIAG();
        }
        else
        {
            npn = Environment.ProcessorCount;
            tb = lsl / npn;
            if (lsl % npn == 0)
                lb = lsl;
            else
            {
                tb = tb + 1;
                lb = tb * npn;
            }
            int i;
            for (i = 1; i <= lb - lsl; i++)
                sl = sl + "X";
        f = null;
        f = new long[lsc + 1, lb + 1];
        MatrizBPDIAG();
        }
        tiempo.Stop();
        dt = tiempo.Elapsed;
        this.lbltbpar.Text = dt.TotalMilliseconds.ToString();
        this.lbltbpar1.Text = dt.ToString();
        this.lbllongseca.Text = this.TxtSecA.Text.Length.ToString();
        this.lbllongsecb.Text = this.TxtSecB.Text.Length.ToString();
    }
    private void MatrizBPDIAG()
    {
        int tda, da, num, centi;
        long aux, iM;
        aux = lb;
        num = 1;
        while (aux != 0) { aux = aux / 10; num = num * 10; }
        centi = 0;
        tda = 1;
        Parallel.For(0,lsc,j=>
            {
            f[j, 0] = j * (long)this.NumGap.Value;});
        Parallel.For(0, lb, j =>
        {
            f[0, j] = j * (long)this.NumGap.Value;
        });
        iM = num + 1;
        for (da = 1; da <= lsc + npn - 1; da++)
        {
```

```
                Parallel.For(0, tda, i =>
                {
                    almacenarB(i, iM, num);

                });
                if (tda < npn && centi == 0)
                {
                    tda = tda + 1;
                    iM = iM + tb;
                }
                else
                {
                    centi = 1;
                    if (da < lsc)
                    {
                        iM = iM + num;
                    }
                    else
                    {
                        tda = tda - 1;
                        iM = iM + num;
                    }
                }
            }
            this.lblBParalelo.Text = "El puntaje maximo es " + f[lsc,
lsl].ToString();
        }
        private void almacenarB(int i, long iM, int num)
        {
            int X, Y;
            long aux, val;
            aux = iM + (num - tb) * (i);
            for (val = aux; val <= aux + tb - 1; val++)
            {
                X = (int)(val / num);
                Y = (int)(val % num);

                f[X, Y] = max(f[X - 1, Y] + (long)this.NumGap.Value, f[X -
1, Y - 1] + s(X, Y), f[X, Y - 1] + (long)this.NumGap.Value);
            }
        }


    }
}
```

ANEXO 4: Paper presentado al V Congreso Internacional de Computación y Telecomunicaciones COMTEL 2013

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 difícil 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 algoritmica 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.

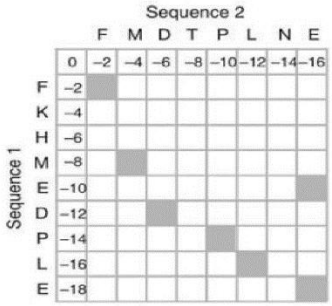

Figura 1. Inicialización con "gap".

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.

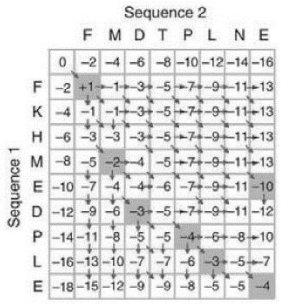

Figura 4. Llenado de la matriz de scores.

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].

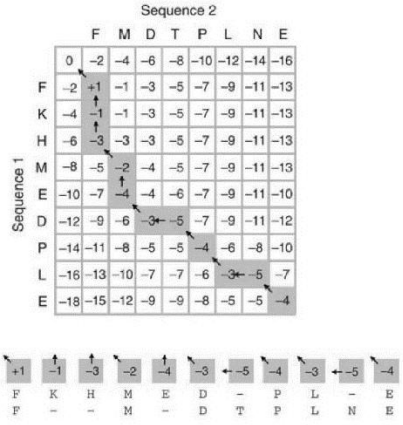

Figura 5. Traceback (recorrido hacia atrás).

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].

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

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.

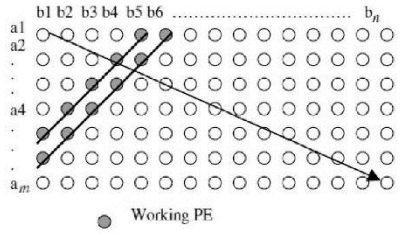

Figura 6. Barrido de antidiagonal en antidiagonal.

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.

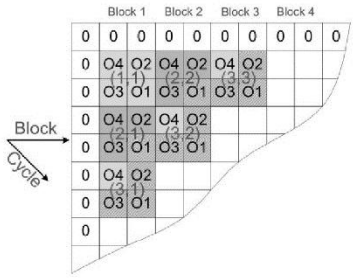

Figura 7. Llenado en Bloques.

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.

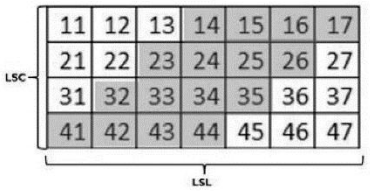

Figura 12. Matriz de scores con posiciones operadas en un instante de tiempo por 3 procesadores.

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*.

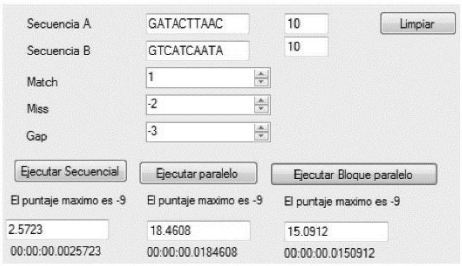

Figura 16. Interfaz de la aplicación de alineamiento de secuencias con los tiempos empleados.

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.

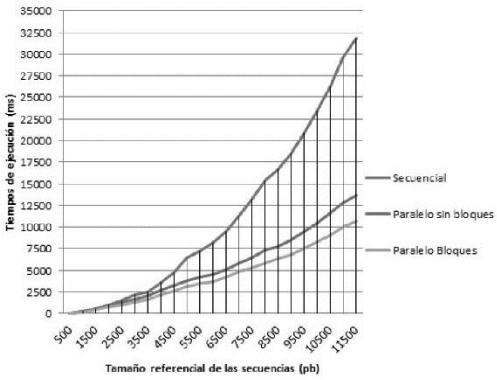

Figura 17. Tiempos de ejecución para el alineamiento de diferentes secuencias con los procesamientos Secuencial y Paralelo.

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.

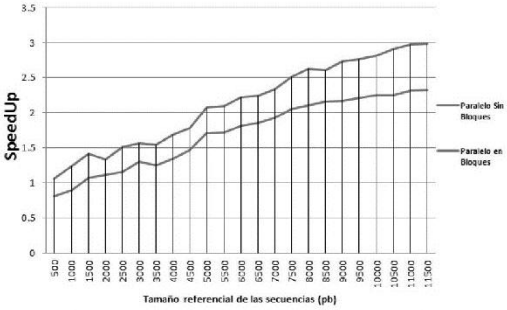

Figura 18. *SpeedUp* de la implementación paralela para los diferentes tamaños referenciales de secuencias.

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/.

ANEXO5: Paper presentado al INTERCON 2013.

Programación Paralela para el Alineamiento de Secuencias de ADN usando Task Parallel Library (TPL)

W. Callisaya, R. Callohuari, J. Jimenez

Abstract— **Sequence alignment is widely used in Bioinformatics for Genome Sequence difference identification. It is the main problem of computational biology. Any sequence of Deoxyribonucleic acid (DNA), Ribonucleic acid (RNA), and protein can be alignment by many algorithms called bioinformatics algorithms. Dynamic algorithms Smith-Waterman, Needleman- Wunsch assess efficiently the alignment of two sequences. These algorithms have been made for a single core computer and sequentially developed, but at the present time, it is necessary to use the Parallel Paradigm, due to the existence of more than one core computers. This paper presents a new implementation of the algorithm for sequence alignment using parallel programming, this requires a new reformulation of Dynamic programming algorithms. This implemented of the Parallel algorithm required a filling of the scores matrix from diagonal to diagonal; the filling of each diagonal was performed in parallel using all available processors. For that it was necessary to use Visual Studio 2010 C# and Framework 4.0 that offers us the Task Parallel Library – TPL. The implementation introduced in this paper made a comparison between the dynamic algorithms Needleman- Wunsch algorithm, Smith-Waterman and our algorithm to test the execution time. The results show that our Algorithm decreased the time of execution when it is compared with Needleman- Wunsch and Smith-Waterman algorithms.**

Keywords— **Bioinformatics, Sequence Alignment, Dynamic programming, Parallel programming, TPL (Task Parallel Library), Computational Biology.**

I. Introducción

En la bioinformática el principal problema de la Biología Computacional es el alineamiento de secuencias biomolecculares (ADN, ARN o secuencias de aminoácidos), ya que la similitud de 2 secuencias implica similitud funcional o estructural significativa.[1]

Actualmente el procedimiento para el análisis de una nueva secuencia de proteínas siempre comienza con una comparación de esta secuencia con las principales bases de datos. En realidad más secuencias han sido supuestamente caracterizadas por la búsqueda en bases de datos que por cualquier otra tecnología.[2]

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 [3], 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 todos los alineamientos existentes.

Existen 2 tipos de alineamientos: El Alineamiento Global y el Alincamiento Local, el primero busca encontrar el mejor alineamiento sobre toda la longitud de ambas secuencias y el segundo caso solo se interesa por la región local mejor alineada, esto se describe en la Fig. 1.

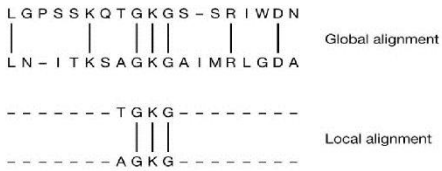

Figura1. Alineamiento Global y Alineamiento Local.

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 un computador paralelo y a su difícil implementación dado que se debia de preocuparse por la comunicación entre los procesadores, pero en la actualidad ya existen librerias que nos apoyan a desarrollar en Paralelo, tales como el OPENMP disponible para C++, pero Visual Studio con en 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 paper es un producto transdisciplinario desarrollado por informáticos y biólogos, presenta una propuesta para apoyar a la solución del alineamiento de secuencias usando la programación paralela usando TPL, enfocandose en el llenado de la matriz de scores descrito en la Sección II.

La organización del trabajo es la siguiente: en la Sección II se explica cómo es el procedimiento para realizar el alineamiento de secuencias con Programación Dinámica. La Sección III detalla como es el trabajo en Paralelo dando referencias a instrucciones con TPL. La Sección IV muestra trabajos relacionados que intentaron abordar el problema del alineamiento de secuencias con el Procesamiento Paralelo. La Sección V detalla la propuesta algorítmica para solucionar el problema del alineamiento de secuencias usando el Paradigma

W. Callisaya, Universidad Nacional Jorge Basadre Grohmann, Perú, Nosliwsys@gmail.com
R. Callohuari, Universidad Nacional Jorge Basadre Grohmann, Perú, Liabiol@yahoo.es
J. Jimenez, Universidad Nacional Jorge Basadre Grohmann, Perú, jjimenezc16@hotmail.com

Paralelo. La Sección VI muestra detalles de su implementación. La Sección VII muestra los 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 VIII.

II. ALINEAMIENTO DE SECUENCIAS CON PROGRAMACION DINAMICA

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 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 Fig.2, para el caso del Alineamiento Local tanto la primera fila y columna se llenan con 0.

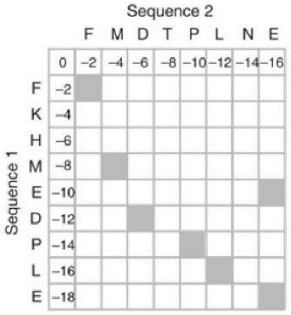

Figura 2. Inicialización con gap

Paso 2 llenado de la matriz de scores: Se procede a llenar todos los valores de la matriz según la función descrita en la Fig. 3, si es para un Alineamiento Global y con la función descrita en la Fig. 4 para un Alineamiento Local, el ejemplo de la Fig. 5 describe el proceso.

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

Figura 3. 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 4. Función del Alineamiento Local

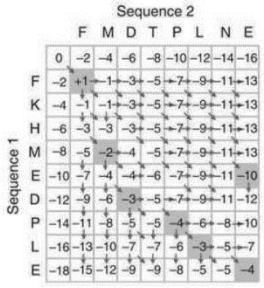

Figura 5. Llenado de la matriz de scores

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 del mejor alineamiento y se hace un recorrido hacia atrás para identificar el alineamiento como se describe en la Fig. 6, en el caso del Alineamiento Local inicia en el mayor valor de la matriz de scores hasta llegar a un valor 0.[4][5]

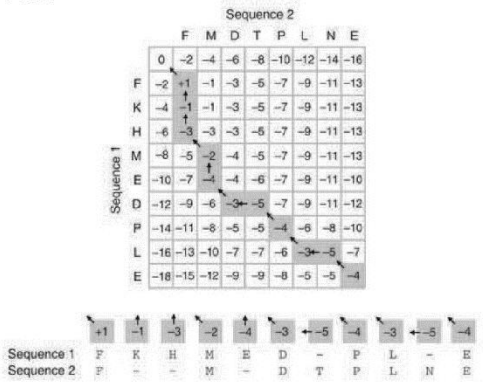

Figura 6. Traceback (recorrido hacia atrás)

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.[6]

III. PROGRAMACIÓN PARALELA

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 modelo de programación paralela usada es el Fork Join, por el cual un hilo es dividido en múltiples hilos y finalmente unidos en un único hilo, como se observa en la Fig. 7. [7]

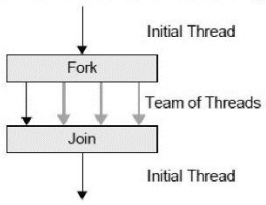

Figura 7. El Modelo Fork-Join

El for y foreach es la herramienta más usada en C# con ellas se pueden hacer bucles secuenciales, llamados así porque una iteración no se inicia hasta que la iteración anterior se ha completado. TPL 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. [8].

Si deseamos hacer 100 iteraciones la estructura del for secuencial sería:

```
for (int = 1; i<=100; i++)
{ ... instrucciones ... }
```

Pero en el caso de un for paralelo con TPL la estructura sería:

```
Parallel.For (1, 100+1, (int i) =>
{ .... instrucciones ...});
```

Existen más clases para el trabajo en paralelo como la clase Task, el presente trabajo usará la clase Parallel.for para su desarrollo.

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 por:

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

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. [9]

IV. Trabajos Relacionados

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 de la fase 3 del algoritmo de Alineamiento de Secuencias pero no deja de ser este una propuesta secuencial. [10]

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 en el 2006.[11] 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 diagonal en diagonal pudiendo, ser este trabajo distribuido en varios núcleos como se indica en la Fig. 8.

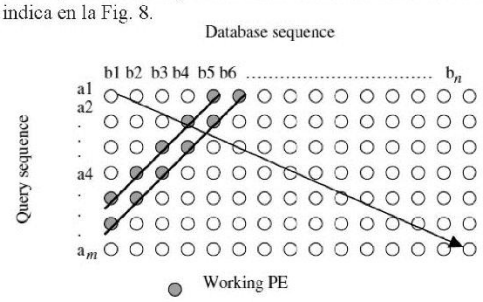

Figura 8. Barrido de diagonal en diagonal

Propuestas similares se observa en el 2010 para abordar el problema usando el procesamiento paralelo utilizando un procesador especializado, en el cual también refiere al llenado de diagonal en diagonal, y propone realizarlo por bloques en cada ciclo como se muestra en la Fig. 9. [12]

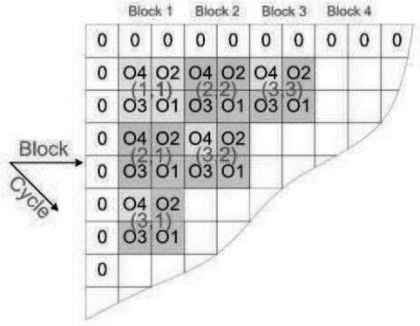

Figura9. Llenado en Bloques.

Propuesta similar se realiza esta vez usando la GPU y CUDA utilizando los multiprocesadores de la tarjeta de video.[13]

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 diagonal en diagonal para un proceso paralelo, la propuesta algorítmica se describe en la siguiente sección.

V. Diseño del algoritmo

Al existir una fuerte dependencia en el llenado de la matriz de scores, se tuvo que reformular el algoritmo para el llenado de diagonal en diagonal, observemos como es el comportamiento de una matriz no cuadrada. En la Fig. 10 se muestra una matriz de 5 filas y 7 columnas, con sus posiciones respectivas, podemos observar que si recorremos las diagonales 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 10. 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, 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.

A. Idea para la pararelización 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 Fig. 11.

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 11. Matriz con las posiciones i y j concatenadas

Al analizar ahora cada diagonal 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 numero 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 Fig. 12.

Considerando esta situación y futuras situaciones se evalúa la longitud de la cadena más larga, obteniendo el número de cifras de su longitud definimos un N que es igual a diez elevado a la potencia c, siendo c el número de cifras de su longitud, y de acuerdo a ello se multiplica N al valor de la posición de la fila i y posterior a ello se concatena con j, al realizar esto se puede observar que toda diagonal se comporta como una progresión aritmética de razón N-1, en la Fig. 13 se observa esta situación.

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

Bastará conocer el inicio de la diagonal para poder calcular cualquier valor de la diagonal con la ecuación:

$$NA=iM+(N-1)*(i-1) \quad (2)$$

Siendo:
NA=El valor de la celda actual
iM=El valor del inicio de la diagonal
$N=10^c$
i=Posición i-ésima de la diagonal.
c=Numero de cifras de la secuencia más larga.

La aplicación de (2) se puede observar en la Fig. 13, en este ejemplo podríamos asumir que se está usando 3 procesadores en un instante de tiempo se pueden llenar 3 valores de la matriz de scores, por ejemplo para el valor de NA=312, con N=100, los valores de su diagonal dependerán de i, entonces si asumimos un i=5.

NA=312+ (100-1)*(5-1)
NA=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 13. Matriz de scores con posiciones concatenadas desarrolladas en un instante de tiempo por 3 procesadores.

Para obtener el valor de la fila y la columna se dividirá el valor de NA 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.

B. Elaboración de los diagramas de Flujo

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 diagrama de flujo de la Fig. 14.

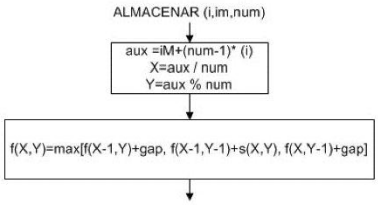

Figura 14. Diagrama de flujo de la función Almacenar.

En el diagrama de flujo se puede apreciar X y Y que son respectivamente las posiciones de la matriz recuperadas de aplicar la ecuación (2), f( ) 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.

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 diagrama de flujo de la Fig. 15.[14]

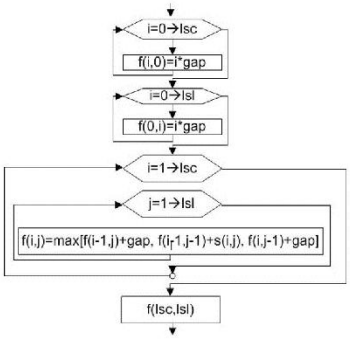

Figura 15. Diagrama de flujo del paso 2 para alineamiento de secuencias con programación Dinámica.

Este paso ha sido readaptado para que sea fácilmente paralelizable con la idea descrita en la Subsección A, para lo cual se elaboró el Diagrama de Flujo descrito en la Fig. 16.

En la Fig. 16 se observa un Pre procesamiento que es el desarrollo del paso 1 del Algoritmo de Alineamiento de Secuencias con Programación Dinámica, a su vez se obtiene el total de cifras de la secuencia más larga, a continuación se describe en detalle las variables usadas en la Fig. 16:

da = Representa a la diagonal Actual

lsc = Longitud de la secuencia más corta

lsl = Longitud de la secuencia más larga

tda = Tamaño de la diagonal actual

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

i = posición i-ésima de la diagonal.

iM = Valor inicial de la diagonal

num = El valor de 10 a la potencia del número de cifras de la longitud de la secuencia más larga

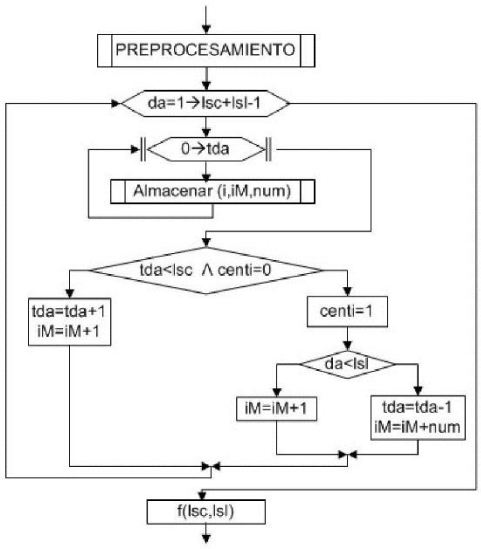

Figura 16. Diagrama de Flujo para el llenado de la Matriz de Scores usando varios procesadores.

VI. Implementación.

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. Se implementa también en C# el algoritmo de Programación Dinámica para alineamiento de secuencias para realizar la comparación de tiempos respectiva, en la Fig. 17 se observa la implementación.

```
Gap = (long)this.NumGap.Value;
for (i = 0; i <= lsc; i++)
 f[i, 0] = i * Gap;
for (i = 0; i <= lsl; i++)
 f[0, i] = i * Gap;
for (i = 1; i <= lsc; i++)
 for (j = 1; j <= lsl; j++)
  f[i, j] = max(f[i - 1, j] + Gap, f[i - 1, j - 1] + s(i, j), f[i, j - 1] + Gap);
this.lblresp.Text = "El puntaje maximo es " + f[lsc, lsl].ToString();
```

Figura 17. Implementación en C# del algoritmo de alineamiento de Secuencias con Programación Dinámica.

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. En la Fig. 18 se observa la interfaz de la aplicación final con los tiempos obtenidos usando StopWatch.

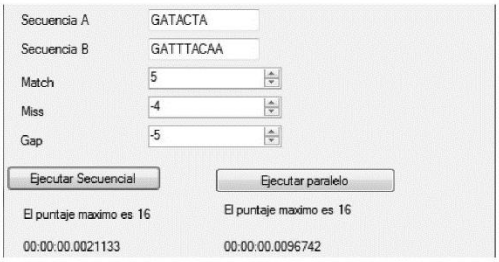

Figura 18. Interfaz de la aplicación de alineamiento de secuencias con los tiempos empleados.

VII. Experimentación y resultados.

En esta sección se hace un estudio comparativo de los algoritmos de programación dinámica clásico con el algoritmo Paralelo implementado.

En la Fig. 18 se observa que el algoritmo secuencial es más eficiente que el algoritmo desarrollado con programación paralela, pero estos resultados cambian a medida que el tamaño de las secuencias se incrementa. Si observamos en la Base de Datos del GenBank [15] podemos encontrar secuencias aproximadamente de 100 a 80 000 000 pares de bases (pb).

En el estudio del rendimiento del algoritmo se escogieron un total de 46 pares de secuencias obtenidas de la Base de Datos del GenBank, 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 el Paralelo se describen en la Tabla I.

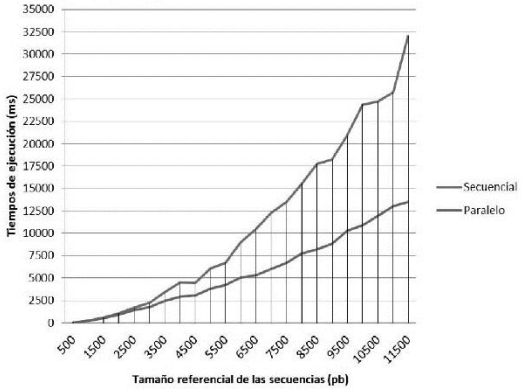

Figura 19. Tiempos de ejecución para el alineamiento de diferentes secuencias con los procesamientos Secuencial y Paralelo.

TABLA I
TIEMPOS DE EJECUCIÓN PARA LOS PROCESAMIENTOS SECUENCIAL Y PARALELO

Tamaño Referencial de las Secuencias (pb)	Tiempos de Ejecución (ms)		SpeedUp
	Secuencial	Paralela	
500	28.8855	60.4119	0.47814255
1000	220.5297	253.6239	0.86951466
1500	550.2301	518.5370	1.06112023
2000	1027.3235	937.7127	1.09556317
2500	1710.9242	1428.7854	1.19746758
3000	2218.3856	1747.0045	1.26982249
3500	3373.6354	2400.9828	1.40510603
4000	4467.6367	2925.8878	1.52693371
4500	4456.0655	3033.1840	1.46910491
5000	6047.5459	3819.2180	1.58345135
5500	6722.2968	4196.9825	1.6016976
6000	8960.7439	5031.8141	1.78081776
6500	10445.8243	5349.1646	1.95279545
7000	12248.9477	6040.1127	2.02793363
7500	13475.9287	6672.7782	2.01953793
8000	15517.5608	7756.9780	2.00046472
8500	17769.7332	8184.5972	2.17111884
9000	18228.2429	8827.8984	2.06484512
9500	20904.2593	10298.1009	2.02991401
10000	24367.5463	10913.9639	2.23269442
10500	24691.3196	11934.9467	2.06882529
11000	25722.6716	13031.3342	1.97390928
11500	32026.3403	13509.5016	2.37065299

Se puede apreciar que para secuencias cortas el procesamiento tiene un menor tiempo de ejecución pero a medida que va incrementándose el tamaño de las secuencias el procesamiento paralelo es mucho más eficiente esto se puede apreciar en la Fig. 19.

Se puede apreciar que se obtiene un incremento del tiempo de respuesta con el procesamiento paralelo hasta el doble que el procesamiento secuencial esto se aprecia en la Fig. 20.

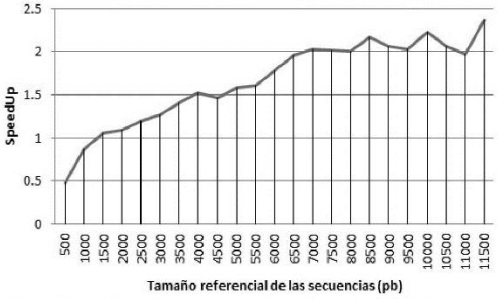

Figura 20. SpeedUp de la implementación paralela para los diferentes tamaños referenciales de secuencias.

VIII. Conclusiones.

En este paper hemos 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, teniendo como producto una herramienta de software que como se aprecia en la Fig. 19 logra 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.

Referencias

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

[2] D. Higgins and W. Taylor, *Bioinformatics: Sequence, Structure and Databanks*, vol. 2, no. 2. New York, United States, 2000.

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

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

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

[6] Http://www.ebi.ac.uk/Tools/psa/ .

[7] B. Chapman, G. Jost, and R. Van der Pas, *Using OpenMP: portable shared memory parallel programming*. London, England, 2008.

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

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

[10] 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.

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

[12] 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.

[13] 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.

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

[15] http://www.ncbi.nlm.nih.gov/genbank/.

Wilson Cesar Callisaya Choquecota Student of the last cycle of the career of Computer and Systems Engineering, National University Jorge Basadre Grohmann (UNJBG). He has collaborated in the Computational Biology course in UNBJG, between 2011 and 2012. He has been a speaker at CONSIC - UNA, in 2012, with the theme topic "Alignment of Sequences of DNA with Programming Dynamics". He has been a speaker at XIII CIIS - UNJBG, in 2012, with the theme topic "Heuristic Sequence Alignment - FASTA". His current interest is the use of parallel programming in the area of Computational Biology.

Rosalía Callohuari Quispe received the title of Biologist Microbiologist from UNJBG, in 2004, Master candidate in Bioquimica from UNMSM and graduate of the doctorate in Environmental Sciences from UNJBG, in 2012. She has been professor of Computational Biology at UNJBG between 2011 and 2012. Her current research interest includes thermopiles microorganisms.

Juan Ubaldo Jiménez Castilla received the Engineering degree in Systems Engineering from UPT, in 1996, Msc. Computer and Information Sciences from UNJBG, in 1998, Licensed Education Electronic from UJCM, in 2005, Commercial Engineer from UJCM, in 2006, Dr. Science Education from USP and Doctor candidate in Systems Engineering from UNFV. Currently, he is a Professor of Paralell Algoritms and Paralell Programming. His current research interest includes Systems and Rhizomes.