UNIVERSIDAD DE SANTIAGO DE CHILE

FACULTAD DE CIENCIA

Departamento de Matemática y Ciencia de la Computación

ALGORITMO PARALELO APLICADO A LA CARACTERIZACIÓN FÍSICA DE ARN NO CODIFICANTE EN GENOMAS BACTERIANOS

WLADIMIR YALL ALBORNOZ LEYTON

ANDRÉS ALBERTO BARRERA SOZA

Profesor Guía: Oscar Francisco Rojas Díaz

TRABAJO DE TITULACIÓN PRESENTADO
EN CONFORMIDAD A LOS REQUISITOS
PARA OBTENER EL TÍTULO DE
ANALISTA EN COMPUTACIÓN CIENTÍFICA

Santiago, Chile

2017

RESUMEN

En esta memoria se presenta un algoritmo de cómputo paralelo para la caracterización de genes de ARN no codificante en genomas bacterianos. Para ello, se proponen dos algoritmos paralelos para procesar la secuencia de nucleotidos que describe a un genoma. El primer algoritmo consiste en dividir la secuencia del genoma según el número de hilos o hebras (desde ahora llamados threads) disponibles en la arquitectura de procesador, teniendo todos los threads una similar carga de trabajo. El segundo algoritmo, es una propuesta que utiliza un gestor que se encarga de asignar de forma dinámica el número de threads según el largo de sub-secuencias (segmentos del genoma a ser analizadas), lo cual permite utilizar los recursos de procesador de forma eficiente.

Ambas soluciones calculan la energía mínima de plegamiento de cada segmento para posteriormente calcular la variable estadística Z-score de cada segmento, siendo el Z-score una variable significativa en el proceso de detección de genes de ARN no codificante. Para ambas soluciones se utiliza la librería pthread de Posix para el cómputo paralelo. Además, se presenta un análisis de resultados de eficacia y eficiencia de los algoritmos propuestos.

ABSTRACT

A parallel algorithm for the characterization of non-coding RNA in bacterial genomes is presented in this report. To do this, we propose two parallel algorithms to process the nucleotide sequence that describes a genome. The first algorithm consists in dividing the sequence of the genome according to the number of threads available in the processor architecture, all threads having a similar workload. The second algorithm is a manager that dynamically allocates the number of threads according to the length of subsequences (segments of the genome to be analyzed), which allows to use the resources of the processor efficiently.

Both solutions calculate the minimum energy of folding of each segment, to later calculate the statistical variable Z-score of each segment, being the Z-score a significant variable in the process of detection of non-coding RNA genes. For both solutions the pthread of Posix library is used for the parallel work. In addition, an analysis of the results of efficiency and efficiency of the proposed algorithms is presented.

AGRADECIMIENTOS

Gracias a

Wladimir Yall Albornoz Leyton

AGRADECIMIENTOS

Gracias a

Andrés Alberto Barrera Soza

TABLA DE CONTENIDOS

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

CAPÍTULO 1. INTRODUCCIÓN

El ácido desoxirribonucleico (ADN), es una molécula que se encuentra en los cromosomas del núcleo de las células y contiene la información genética de un ser vivo y de algunos virus. Los segmentos de ADN que constituyen la unidad fundamental, física y funcional de la herencia en los seres vivos se denominan genes, los cuales en su conjunto forman el material genético que posee un organismo, más conocido como genoma. La molécula de ADN almacena a largo plazo información con la cual construye otros componentes de las células, como lo son las proteínas y las moléculas de ARN (ácido ribonucleico). El ARN trabaja de intermediario en la información genética y de catalizador en la síntesis de proteínas (Clayton & Dennis, 2003). En este trabajo se focaliza la atención en las moléculas de ARN (llamados genes de ARN), específicamente en moléculas de ARN no codificante[1] que son genes que están ocultos en el ARN y por tanto difíciles de detectar en los genomas. Además, las diversas funciones que desempeña el ARN no codificante en el proceso de regulación de la expresión génica[2] lo convierte en un objetivo importante de estudio (Kaikkonen et al., 2011).

El trabajo consiste en desarrollar algoritmos paralelos que permitan procesar de manera eficiente y eficaz las secuencias de ARN que describen a dichas moléculas, donde en específico se requiere calcular el valor de la medida Z-score de manera veloz en sub-secuencias de ARN extraídas de genomas bacterianos. La medida Z-score es un descriptor relevante que permite detectar la presencia de genes pequeños de ARN no codificante (ncRNAs por sus siglas en inglés) en secuencias genómicas (Rogers, 2012), donde al no ser codificantes, estos son difíciles de detectar en los genomas y por lo tanto se requieren programas especializados de reconocimiento de patrones que detecten su ubicación dentro de un genoma en estudio.

1.1. ANTECEDENTES Y MOTIVACIÓN

La existencia de los genes de ARN no codificantes es conocida desde el año 1981, sin embargo no habían recibido mayor atención hasta que estudios recientes han descubierto su rol en la regulación de una diversa gama de procesos celulares (Argaman et al., 2001; Eddy, 1999; Vogel et al., 2003), motivo por el cual la detección computacional de genes se centra en la identificación de genes con expresión proteica (Waters & Storz, 2009).

[1] ARN no codificante: Molécula de ARN funcional que no se traduce en una proteína.

[2] Expresión génica: Proceso por medio del cual se transforma la información codificada por los ácidos nucleicos en proteínas.

La característica de los genes de ARN no codificantes es que no se traducen en proteínas sino que se transcriben en moléculas de ARN con actividad regulatoria (Vogel & Sharma, 2005). El descubrimiento de este transcriptoma oculto[3] añade un nivel de mayor complejidad a la discriminación entre ARN codificante y no codificante, por lo que se considera un desafío el estudio de estos genes para el entendimiento de la expresión y regulación de la información genética (Dinger et al., 2008).

Para avanzar en el conocimiento biológico, se requieren métodos computacionales que puedan detectar con rapidez pequeños genes ARN no codificante en secuencias genómicas (Uzilov et al., 2006), permitiendo así la identificación a gran escala de genes nóveles y facilitando la exploración de los mecanismos de regulación génica (Xu et al., 2009).

1.2. DESCRIPCIÓN DEL PROBLEMA

Con el reciente crecimiento en la disponibilidad de material genómico secuenciado, existe la necesidad de mejorar la capacidad de registro de genes y elementos estructurales de una manera rápida, eficiente y precisa (Chen et al., 2002; Tran, 2009). Actualmente, los algoritmos que realizan procesamiento de cadenas genómicas (Swenson et al., 2012; Zuker, 2003) y que calculan medidas de energía requeridas para la obtención de Z-score, se enfocan principalmente al cálculo secuencial y paralelo multithread de energías mínimas de plegamiento[4] (emp) de secuencias de genes y no al procesamiento completo de una secuencia que describe a un genoma.

A su vez, para describir a un genoma a través de los valores de Z-score, es necesario dividir el genoma en estudio, en múltiples segmentos (sub-cadenas) con diferentes tamaños (ej. largo 50, 100, 150, 200, 250 y 300), para luego a cada segmento calcular su Z-score. El cálculo de Z-score es lento, ya que requiere que se calcule 1000 veces la emp de secuencias generadas de forma aleatoria, donde si las secuencias obtenidas no cumplen ciertas propiedades de equilibrio químico-físicas (equilibrio energético de la molécula en estudio), éstas no son consideradas en la muestra de 1000 repeticiones.

En este contexto, los algoritmos y programas disponibles (Swenson et al., 2012; Zu-

[3] Transcriptoma oculto: Conjunto de moléculas de ARN obtenidas a partir de la transcripción de informaciones genéticas de la molécula de ADN. El término transcriptoma oculto se refiere a los genes de ARN no codificante, de los cuales no se tiene toda la información respecto a sus funciones.

[4] Energía mínima de plegamiento: energía mínima libre de la estructura secundaria de ARN producida al aparearse las bases (Clote et al., 2005)

ker, 2003) no ofrecen herramientas que permitan procesar un genoma completo de forma rápida y menos aun gestionar los recursos de procesador de forma eficiente. Esto se debe a que los procesamientos completos de los genomas en los cuales se requiera buscar ARN no codificante utilizando la medida de Z-score como descriptor, necesariamente requieren ser implementados sobre arquitecturas paralelas y distribuidas de gran capacidad de cómputo, y la disponibilidad actual tal como se comentó, solo se remite al cálculo de emp de forma paralela multi-thread y no al cálculo de Z-score ya sea de forma paralela como distribuida.

En este sentido, es necesaria la utilización de algoritmos paralelos eficientes, que permitan un escalamiento para un procesamiento distribuido, para así procesar secuencias de genomas en tiempos mínimos y a su vez usando los recursos disponibles de forma eficiente.

1.3. SOLUCIÓN PROPUESTA

Esta memoria se centra en el desarrollo de dos soluciones de cómputo paralelo multi-tread. La primera solución realiza la división de la secuencia completa de un genoma en segmentos según número de threads asignados, y cada thread procesa de forma independiente cada segmento del genoma. La segunda solución asigna dinámicamente el número de threads de acuerdo con una estimación a posteriori del tiempo de procesamiento de sub-cadenas de menor tamaño extraídas de la secuencia del genoma, lo cual es obtenido en un período de calibración del algoritmo. Luego, se determina de forma automática cuántos threads deben ser asignados para procesar cada sub-cadena (de diferentes tamaños) para así obtener una tasa de procesamiento mayor o igual a lo que un experto humano podría configurar en base a observaciones y análisis de rendimiento.

La segunda solución propuesta es muy simple, pero eficaz y consiste esencialmente en el uso de un algoritmo paralelo que se adapta a la arquitectura multiprocesador usando para el cómputo todos los recursos de CPU, procurando la no existencia de threads sin trabajo. La propuesta consta de lo siguente: a) capturar tiempos de procesamiento paralelos usando diferentes números de threads y diferentes tamaños de secciones del genoma (secuencias de largo 50, 100, 150, 200, 250 y 300), b) obteniendo un tiempo promedio para cada configuración (tiempo de uso de threads según el largo de las sub-cadenas) y c) usando los tiempos promedios menores para cada tamaño, se asigna el número de thread a utilizar que asegure un mínimo tiempo de procesar cada sub-secuencia. De esta manera, utilizando todos los threads disponibles en la arquitectura,

se distribuye el cálculo, minimizando el tiempo total de procesamiento del genoma.

1.4. OBJETIVO GENERAL

El objetivo general de este trabajo es el diseño e implementación de un algoritmo paralelo adaptativo que obtenga el valor de Z-score en secuencias genómicas bacterianas de forma eficiente y eficaz.

1.4.1. Objetivos Específicos

1. Estudiar técnicas de cálculo de Z-score.
2. Analizar y diseñar de estrategias de cómputo paralelo multi-thread para el cálculo de Z-score.
3. Diseñar e implementar un gestor paralelo adaptativo para el procesamiento eficiente de secuencias genomicas.
4. Concluir sobre el trabajo realizado y su proyección.

1.5. ALCANCES Y LIMITACIONES

1.5.1. Alcances

El principal alcance es que el algoritmo puede escalar a soluciones distribuidas, donde el paralelismo local se realiza en un nodo con soporte multi-thread de un cluster de computadores. Mientras que la solución distribuida tiene como objetivo disminuir el tiempo global de procesar un genoma completo, al distribuir sub-secciones del genoma en todo el cluster del sistema distribuido.

1.5.2. Limitaciones

La primera limitación es que los experimentos se realizan sobre un solo genoma, específicamente el genoma bacteriano de la Echerichia Coli, cepa K-12 MG-1655. La segunda limitación, es que los experimentos se realizaron en un solo nodo con soporte para 32 threads de un cluster de computadores del centro de investigación CITIAPS[5] de la Universidad de Santiago de Chile.

1.6. CONTRIBUCIÓN DEL TRABAJO

La contribución principal de este trabajo, es el diseño e implementación de una herramienta que sea parte de un framework que detecte patrones en genomas, específicamente enfocado en la rápida localización de genes de ARN no codificante. Esta herramienta servirá de apoyo para la investigación del área de Bioinformática. Específicamente, esta será utilizada por el Centro de Investigación de Biotecnología y Bioingeniería (CeBiB)[6] para el análisis y búsqueda de patrones de ARN no codificante en genomas, donde parte de este trabajo fue presentado en (Rojas et al., 2017).

1.7. ORGANIZACIÓN DEL DOCUMENTO

El resto del documento esta organizado de la siguiente manera. El Capítulo 2 presenta el marco teórico relacionado con este trabajo de memoria. El Capítulo 3 presenta la propuesta de algoritmo paralelo adaptativo para el cálculo de Z-score. Además, en el mismo Capitulo 3 se presenta y detalla el proceso de desarrollo de software y su implementación. El Capítulo 4 presenta el análisis de resultados del trabajo. Finalmente, el Capítulo 5 presenta las conclusiones y trabajos futuros derivados de esta memoria.

[5]CITIAPS: Centro de Innovación en Tecnologías de la Información para Aplicaciones Sociales.

[6]CeBiB: Centro de investigación en bioninformática, bioingeniería y biotecnología compuesto por varias universidades chilenas (CeBiB, 2017).

CAPÍTULO 2. MARCO TEÓRICO

A continuación se presenta un resumen de los contenidos teóricos requeridos para el desarrollo de este trabajo, donde gran parte del contenido del marco teórico del área de biología molecular corresponde a extractos de una tesis de magister disponible en (Rogers, 2012). El Capítulo esta organizado de la siguiente manera.

La Sección 2.1 presenta el dogma central de la biología molecular. La Sección 2.2 presenta los genes de ARN no codificante. La Sección 2.3 presenta la predicción de ARN no codificante. La Sección 2.4 presenta el uso de la computación paralela en el procesamiento de secuencias de ARN. Finalmente, la Sección 2.5 presenta el genoma en estudio y el cálculo de Z-score.

2.1. DOGMA CENTRAL DE LA BIOLOGÍA MOLECULAR

El dogma central de la biología moderna es un concepto que ilustra los mecanismos de transmisión y expresión de la herencia genética tras el descubrimiento de la codificación de ésta en la doble hélice[1] del ADN (Clayton & Dennis, 2003). Esto propone que exista una unidireccionalidad en la expresión de la información contenida en los genes de una célula.

La información genética usada en el desarrollo y funcionamiento de todos los organismos vivos, de las bacterias y de algunos virus, y que a su vez es responsable de la transmisión hereditaria está representada por cuatro bases nucleotídicas: Adenina (A), Citosina (C), Guanina (G) y Tiamina (T). Estas bases, representadas por una doble cadena, generan una larga secuencia de nucleótidos, compuestos que forman el ácido desoxirribonucleico, también conocido como ADN de cada organismo (Clayton & Dennis, 2003).

Los procesos básicos de flujo de información consisten en la copia de moléculas de ADN llamada replicación, el traspaso de la información desde los genes a moléculas de ARN llamado transcripción y el traspaso de la información contenida en las moléculas de ARN mensajero a proteínas llamado traducción (Rogers, 2012), lo cual se ilustra en la Figura 2.1.

Sin embargo, se han descubierto una serie de excepciones al dogma, como el paso de ARN a ADN mediante la enzima transcriptasa inversa, la replicación del ARN y el paso directo de ADN a proteínas. Estos casos se consideran aislados ya que ocurren sólo en condiciones particulares, ya sea en virus o en laboratorio (Rogers, 2012).

[1] Doble Hélice: Nombre que se asocia con la estructura molecular del ADN.

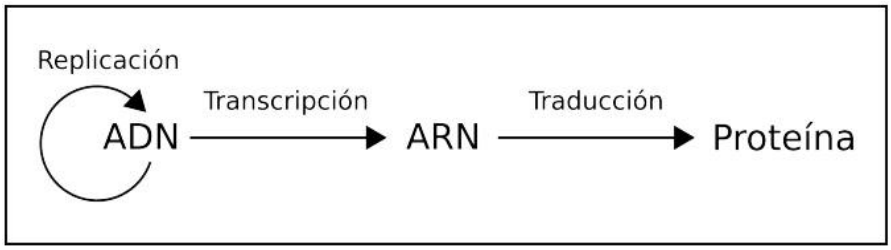


Figura 2.1: Dogma central biología molecular.
Fuente: Modificada de (Rogers, 2012), 2017.

Si bien el dogma central continúa siendo válido en cuanto a la descripción del proceso general de formación de proteínas desde el material genético, también se ha notado que existe una mayor cantidad de información dentro del genoma que la contenida en los genes que codifican proteínas; esta información es la que controla el momento adecuado y la rapidez en el proceso de síntesis de proteínas (Perkins et al., 2005; Rogers, 2012).

En organismos complejos se observa que sólo una minoría de los transcritos genéticos codifica proteínas (2-3 %), el resto había sido llamado ADN basura y se explicaba su existencia como vestigios de genes codificantes perdidos durante la evolución (Perkins et al., 2005).

Sin embargo, estudios recientes demuestran que los transcritos no codificantes juegan un papel crítico en la regulación de la expresión génica a través de mecanismos de control que alteran la estabilidad del ARN mensajero, el inicio de la transcripción de genes o la traducción de proteínas (Perkins et al., 2005).

2.2. GENES DE ARN NO CODIFICANTE

El ácido ribonucleico, también conocido como ARN, es una macromolécula esencial para todas las formas de vida conocidas. La información que contiene el ARN también está representada por las bases nucleotídicas del ADN aunque, a diferencia de estas bases, la Tiamina (T) es reemplazada por el Uracilo (U). La secuencia que se genera en el ARN es lineal y monocateriana (de una sola cadena).

Los genes de ARN no codificante son un tipo de secuencias genómicas, las cuales transcriben moléculas que funcionan directamente como ARN, en vez de su traducción en proteínas (Eddy, 2002), cumpliendo importantes funciones estructurales, catalíticas y regulatorias

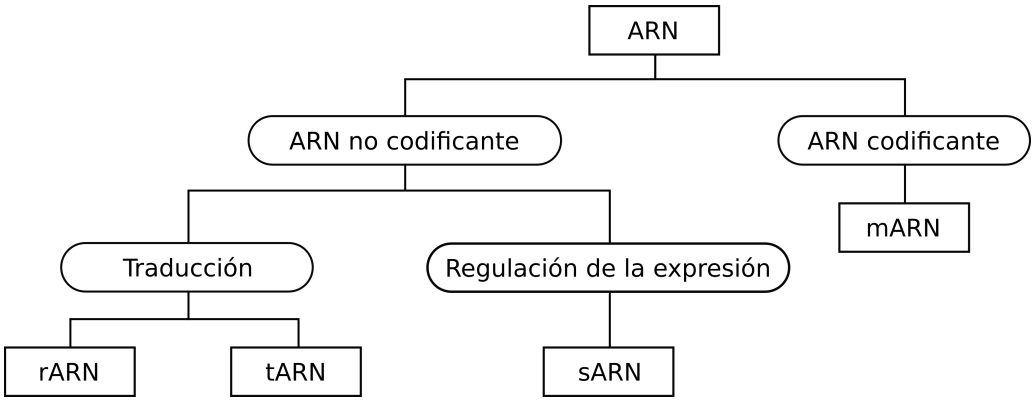


Figura 2.2: Clasificación del ARN en genomas bacterianos.
Fuente: Modificada de (Rogers, 2012), 2017.

dentro de la célula (Chen et al., 2002; Mimouni et al., 2009; Rivas & Eddy, 2001).

Ejemplos clásicos son los genes de ARN ribosomal (rARN) y ARN de transferencia (tARN), los cuales están presentes en todo tipo de vida celular (Nawrocki & Eddy, 2013). Sin embargo, se conoce la existencia de otros tipos de genes de ARN no codificante con roles estructurales y regulatorios cuyo número e importancia se creía marginal hasta hace poco tiempo (Eddy, 2002).

Estudios recientes (Argaman et al., 2001; Eddy, 1999; Vogel et al., 2003) han encontrado que esta nueva clase de genes de ARN no codificante son abundantes en todos los dominios de la vida, indicando que tienen una antigua historia evolutiva (Gottesman, 2005).

En los genomas bacterianos, los genes que codifican moléculas de ARN con actividades regulatorias son conocidos como genes pequeños de ARN no codificante, comúnmente llamados *ncRNAs* en la literatura por su nombre en inglés *small non-coding RNA*, ya que son secuencias cortas con un largo que varía entre 50 y 300 bases nucleotídicas (Vogel & Sharma, 2005).

De ahora en adelante, los genes pequeños de ARN no codificante serán mencionados como genes ncRNAs, apuntando específicamente a los genes pequeños de ARN no codificante de origen bacteriano. En la Figura 2.2 se muestra un esquema simplificado de las clases de ARN en genomas bacterianos.

2.2.1. Importancia de los Genes ncRNAs

En los años 2001 y 2002, cuatro grupos (Argaman et al., 2001; Chen et al., 2002; Rivas & Eddy, 2001; Wassarman et al., 2001) publicaron la identificación de un gran número de genes ncRNAs encontrados mediante un análisis computacional de conservación de secuencias en regiones intergénicas del genoma de la bacteria Escherichia Coli (Livny & Waldor, 2007).

Desde entonces, la apreciación de la importancia de los genes ncRNAs ha crecido enormemente debido a que múltiples estudios genómicos han mostrado su diversidad y amplia distribución en una gran variedad de organismos (Gottesman, 2005; Hershberg et al., 2003; Matera et al., 2007; Vogel & Sharma, 2005; Wassarman et al., 2001).

Actualmente, el análisis de ncRNAs está emergiendo como el factor clave en la regulación celular y ocupando un área central de investigación en biología molecular (Rogers, 2012), ya que toma roles activos en las múltiples capas de regulación de la expresión génica, desde transcripción, maduración de ARN, modificación de ARN y regulación de la traducción (Backofen et al., 2007; Bompfünewerer et al., 2005).

Es importante mencionar que muchas de las clases de ncRNAs fueron descubiertas recientemente o sus números incrementados en análisis recientes, lo que apunta a la relativa dificultad de descubrir las secuencias de ncRNAs comparado al descubrimiento de genes codificantes de proteínas. Esto abre la opción de que existan muchas más clases de genes ncRNAs todavía por descubrir (Eddy, 2002; Machado-Lima et al., 2008; Nawrocki & Eddy, 2013).

Descubrir genes ncRNAs mediante el análisis computacional de secuencias resulta un problema de gran interés (Backofen et al., 2007; Machado-Lima et al., 2008; Pichon & Felden, 2008), siendo la identificación computacional de genes ncRNAs uno de los problemas más desafiantes y relevantes en la actualidad (Machado-Lima et al., 2008; Tran, 2009).

2.2.2. Función Principal de los ncRNAs

Se pensaba que la expresión génica ocurría principalmente mediante el control de la transcripción mediante proteínas represoras o activadoras. Sin embargo, estudios recientes revelan que existen otros mecanismos de regulación basados en moléculas de ARN (Johansson & Cossart, 2003).

La función principal de los genes ncRNAs es la de regular la expresión génica en

distintos niveles, ya sea controlando el inicio de la transcripción, modificando la estabilidad del mARN, modulando la actividad de proteínas y regulando la traducción (Gottesman, 2005; Mimouni et al., 2009; Repoila & Darfeuille, 2009; Wassarman et al., 2001).

Los genes ncRNAs regulan una gran cantidad de procesos biológicos (Livny & Waldor, 2007) permitiendo la adaptación celular en respuesta a cambios en el medio ambiente (Brouns et al., 2008; Pichon & Felden, 2008; Repoila & Darfeuille, 2009).

Algunas de las funciones celulares específicas que se sabe son reguladas por genes ncRNAs son: a) respuesta a cambios de nutrientes en el medio, b) respuesta a estrés celular, c) motilidad celular, d) homeostasis de la membrana celular, e) producción de toxinas y antitoxinas, f) coordinación de la virulencia, g) control de replicación plasmidial y viral (Bejerano-Sagie & Xavier, 2007; Brouns et al., 2008; Johansson & Cossart, 2003; Perez et al., 2009; Pichon & Felden, 2008; Szymanski et al., 2007; Toledo-Arana et al., 2007; Voß et al., 2009).

2.2.3. La Estructura Secundaria

El ARN se pliega como resultado de la presencia de regiones cortas con apareamiento intramolecular de bases, es decir, pares de bases formados por secuencias complementarias más o menos distantes dentro de la misma hebra. La estructura secundaria se refiere, entonces, a las relaciones de apareamiento de bases. El término estructura secundaria denota cualquier patrón plano de contactos por apareamiento de bases (Fontana, 2002) .

Las moléculas de ARN transcritas por los genes ncRNAs tienen una fuerte tendencia a adoptar una conformación plegada mediante el apareamiento de bases complementarias en la misma hebra, resultando en una estructura secundaria relacionada directamente con su función (Lee & Kim, 2008).

Una molécula de ARN puede adoptar múltiples estructuras distintas al haber diversas formas posibles de apareamiento entre sus bases. Al conjunto de todas estas estructuras posibles se le llama ensamble de estructuras secundarias de una secuencia.

Sin embargo, se acepta que la estructura que cuenta con mayor probabilidad de ser adoptada es aquella que posee la energía mínima de plegamiento (emp). Esta estructura puede ser calculada mediante el programa RNAfold[2] (Hofacker, 2009).

Las estructuras secundarias pueden ser representadas de diversas formas, siendo

[2]RNAfold: Programa que predice las estructucturas secundarias de secuencias de ARN o ADN disponible en http://rna.tbi.univie.ac.at/

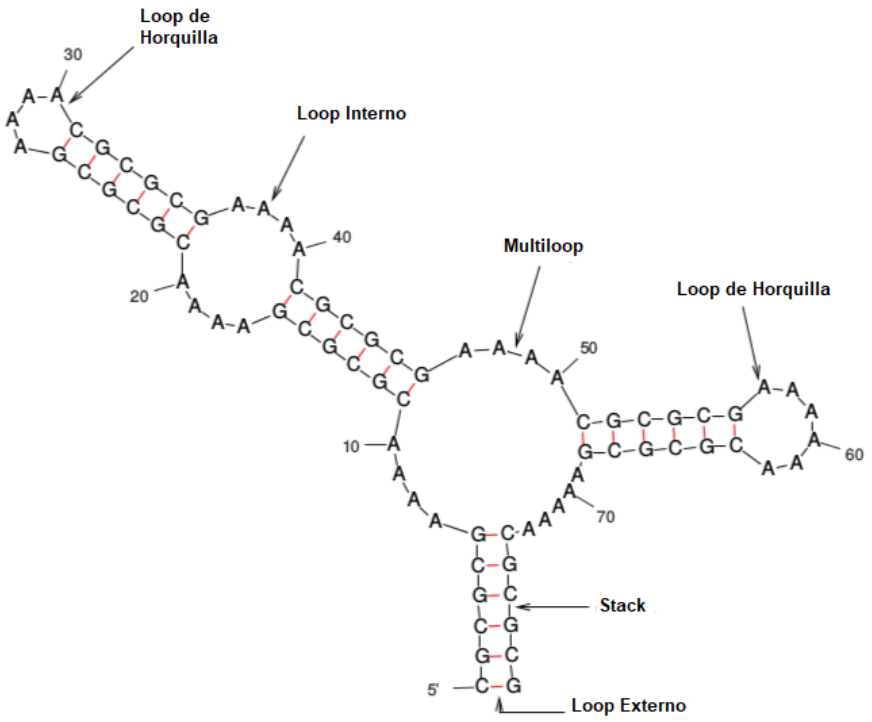


Figura 2.3: Ejemplo de estructura secundaria.
Fuente: Modificada de (Swenson et al., 2012), 2017.

una de las más importantes la representación bidimensional. Para ejemplificar se toma la siguiente secuencia de ARN.

UACCGGUCUGAGCUGUGAUGACAUCGUGCAC

Se puede observar en la secuencia, la existencia de varios pares G-U, los que se forman además de los pares típicos como C-G y-U. El par G-U tiene una gran importancia en la formación de la estructura secundaria en moléculas de ARN y en su interacción con otras moléculas (Varani & McClain, 2000). En la Figura 2.3 se puede ver la representación de una estructura secundaria de ARN:

2.2.4. Propiedades de ncRNAs

La mayoría de los genes ncRNAs encontrados se localizan en regiones intergénicas del genoma; es decir, regiones del genoma flanqueadas por genes que codifican proteínas (Altuvia, 2007; Vogel & Sharma, 2005). En general, los genes ncRNAs tienen un mayor contenido de bases C y G que una región promedio, y el largo del transcrito varía típicamente de 50 a 300 bases nucleotídicas (Altuvia, 2007; Vogel & Sharma, 2005).

Los genes ncRNAs muestran un proceso evolutivo sin igual, ya que son indiferentes a substituciones de bases mientras se conserve la estructura secundaria de la molécula de ARN, por lo cual no existe una fuerte conservación de la secuencia primaria salvo en organismos cercanos filogenéticamente [3] (Tabei & Asai, 2009; Vogel & Sharma, 2005).

También se ha encontrado que los transcritos de genes ncRNAs adoptan estructuras secundarias con energías mínimas de plegamiento (emp) menores que las de secuencias del mismo largo generadas aleatoriamente, siempre y cuando éstas conserven las mismas frecuencias dinucleotídicas (Clote et al., 2005).

2.2.5. Dificultades en Identificación de Genes de ncARNs

El analizar computacionalmente distintos tipos de secuencias genómicas, ha sido eficaz para la predicción de genes que codifican proteínas. Sin embargo, los genes ncRNAs presentan un nuevo conjunto de desafíos para la genómica computacional (Eddy, 2002; Livny & Waldor, 2007).

Los genes ncRNAs muestran un proceso evolutivo único, en el cual las sustituciones de bases distantes se correlacionan con el fin de conservar la estructura secundaria de la molécula de ARN. Por ende, se debe tomar en cuenta no sólo la conservación de la secuencia primaria sino también la estructura secundaria (Eddy, 2002; Tabei & Asai, 2009). Por este motivo, los métodos computacionales desarrollados para identificar genes que codifican proteínas fallan en identificar a los genes ncRNAs, lo que hace que el descubrimiento de estos genes requiera de un método diferente al de los genes que codifican proteínas, ya que a diferencia de estos, los genes ncRNAs no cuentan con atributos genómicos aparentes (Kavanaugh & Dietrich, 2009; Machado-Lima et al., 2008; Rivas & Eddy, 2001).

[3] Filogenética: Estudio de las relaciones evolutivas de distintos organismos a partir de la distribución de los caracteres primitivos y derivados en cada taxón, utilizando matrices de información de moléculas de ADN.

Una de las características que los hacen especialmente difíciles de identificar es su pequeño tamaño, el cual varía entre 50 a 300 pares de bases, a diferencia de genes de proteínas que pueden medir varios miles de pares de bases (Argaman et al., 2001; Machado-Lima et al., 2008; Tran, 2009; Vogel & Papenfort, 2006).

Otra característica es la ausencia de un marco de lectura abierto como el de los genes que codifican proteínas, por lo que no existen señales estadísticas fuertes en la composición de sus secuencias (Nawrocki & Eddy, 2013).

Por último, aunque algunos genes ncRNAs tienen promotores[4] y terminadores[5] conocidos, la identificación de estas señales regulatorias resulta actualmente un desafío bastante difícil (Nawrocki & Eddy, 2013; Tran, 2009; Vogel & Papenfort, 2006).

2.3. PREDICCIÓN DE ARN NO CODIFICANTE

Los genes ncRNAs han sido en su mayoría descubiertos con una pre-selección de candidatos mediante un método computacional predictivo (Rogers, 2012), seguido de la validación experimental de estos candidatos (Altuvia, 2007; Kulkarni & Kulkarni, 2007; Livny & Waldor, 2007).

Existe una gran cantidad de métodos predictivos enfocados a genes ncRNAs. Sin embargo, no existe un método que tenga una gran eficiencia y especificidad, lo cual hace difícil la comprobación experimental de las secuencias debido a la gran cantidad de falsos positivos en las predicciones, por lo que se considera éste un problema abierto en bioinformática (Machado-Lima et al., 2008; Rogers, 2012).

Para la validación experimental, existen variadas técnicas donde se aplica biotecnología y análisis molecular de ARN, pero para ello se requiere acceso a material biológico y laboratorios especializados. Desafortunadamente, la validación experimental de los transcritos es muy costosa, por lo cual resulta importante filtrar computacionalmente los candidatos para reducir los transcritos a validar (Kin et al., 2007). Resulta entonces necesaria la generación de métodos computacionales que disminuyan la cantidad de experimentos de laboratorio (Rogers, 2012) e integren la creciente cantidad de información genómica disponible (Kin et al., 2007).

[4]Promotor: Región de ADN que controla la iniciación de la transcripción de una determinada porción del ADN a ARN.
[5]Terminador: Región de ADN que controla la terminación de la transcripción.

2.3.1. Métodos Computacionales para la Predicción de Genes ncRNAs

Se han propuesto diferentes métodos en la literatura publica. Estos se basan en las características comunes de los ncRNAs encontrados a la fecha y generalmente se separan en dos clases: a) los métodos basados en *homología de la secuencia* y b) métodos *de novo*. Sin embargo, todos ellos generan una gran cantidad de falsos positivos, lo que dificulta el análisis a gran escala de un genoma completo (Tran, 2009).

2.3.1.1. Métodos basados en homología

Los métodos basados en homología buscan secuencias nucleotídicas altamente conservadas entre regiones intergénicas de genomas bacterianos cercanos filogenéticamente (Rogers, 2012), como por ejemplo entre Escherichia Coli y enterobacterias cercanas como Salmonella Typhimurium y Yersinia Pestis. Además, toman en cuenta la conservación de la estructura secundaria y la existencia de un promotor y señal de término a una distancia de 50-400 nucleótidos (Vogel y Sharma, 2005).

Este tipo de métodos ha sido ampliamente utilizado para la predicción de genes ncRNAs con una secuencia homóloga conocida. Mientras resulta bastante eficiente para identificar genes sARN en diversos genomas, no sirven para identificar genes sARN nóveles (aun no descubiertos) (Tran, 2009). Algunos de estos métodos son: QRNA (Rivas & Eddy, 2001), RNAz (Washietl et al., 2005) y Dynalign (Uzilov et al., 2006).

2.3.1.2. Métodos de novo

Los métodos *de novo* utilizan algoritmos sofisticados de aprendizaje automático y no dependen de conocimientos previos sobre el microorganismo a estudiar (Vogel & Sharma, 2005). Este tipo de método consiste en el entrenamiento de un algoritmo clasificador que recibe las variables de un grupo de datos clasificados como casos positivos y negativos, para luego realizar predicciones sobre nuevos grupos de datos sin clasificar (Rogers, 2012). La efectividad del método depende de la capacidad de identificar las características comunes de los genes ncRNAs para ser medidas como variables, que los distingan de otros tipos de secuencias como genes codificantes y secuencias al azar. Algunos de estos métodos son: CONC (Liu, 2007) y

RSSVM (Xu et al., 2009) donde ambos usan maquinas de vector de soporte (SVM por sus siglas en Inglés).

Además, existen métodos que usan minería de datos (Tran, 2009) y/o técnicas de inteligencia artificial para predecir genes de ncRNAs, donde el uso de redes neuronales artificiales ha permitido realizar predicciones con alta eficacia (Rogers, 2012). Estas técnicas para identificar ncRNAs usan como patrones de entrada características físicas, químicas y estructurales de las secuencias de ARN, donde las más significativas son el emp_I[6], *valor de partición*[7], el *porcentaje de bultos*[8] y Z-score[9]. Otra investigación destacable analiza la estructura secundaria del ARN y para ello utiliza dos componentes, una medida para la conservación de la estructura secundaria del ARN basada en el cálculo de una estructura secundaria de consenso y una medida de estabilidad termodinámica, donde el valor de Z-score se normaliza con respecto a la longitud de secuencia y su composición base, pero se puede calcular sin muestreo de secuencias mezcladas (Washietl et al., 2005).

Además de estos métodos predictivos, se han realizado análisis sobre las energías de plegamiento de las secuencias de ARN, en estos se concluye que los ncRNAs tienen energías de plegamiento menores que secuencias aleatorias con la misma composición de bases (Freyhult et al., 2005). Gracias a estas investigaciones se han identificado una serie de variables que se utilizan para analizar las emp, que con sus propiedades discriminantes resultan muy útiles al momento de buscar genes de ncRNAs, donde una de las variables más descriptivas es el Z-score que usa para su cálculo valores estimados de energía mínima de plegamiento (Rogers, 2012).

2.3.2. Análisis de Energía Mínima de Plegamiento

La mayoría de las investigaciones de bioinformática, que están enfocadas en la predicción de genes, toman como base el cálculo de energía mínima de plegamiento (emp de ahora en adelante) tanto de secuencias completas, como de ciertos segmentos de ellas.

[6] emp_I: Es la energía mínima de plegamiento (emp) dividida por el número de nucleótidos G y C.

[7] Valor de partición: Número de secuencias aleatorias que poseen una emp menor que la secuencia original normalizado por el número total de secuencias aleatorias.

[8] Porcentaje de bultos: Porcentaje de bultos presentes en la estructura secundaria que posee la emp de una secuencia de ARN.

[9] Z-score: Corresponde a una medida de la distancia en desviaciones estándar entre la emp de la secuencia original y el promedio de las emp de secuencias aleatorias.

Se han realizado experimentos que han determinado que para genes de ARN en los que la emp de la estructura secundaria es funcionalmente importante, se tiene un menor emp que otros genes del mismo largo y la misma frecuencia dinucleotida, lo que explica la importancia del análisis de energía mínima. Gracias a estos experimentos se empiezan a realizar análisis de Z-score para detectar genes potencialmente funcionales dentro de una secuencia (Clote et al., 2005) y se ha determinado la importancia de la minimización de energía libre en la predicción de la estructura secundaria de secuencias de ARN. Gracias a esto se han desarrollado métodos que permiten mejorar la precisión en la predicción de genes de ncRNAs (Mathews et al., 1999). Otra investigación ha determinado la relación entre la emp y el largo de la secuencia, que se traduce en otra variable a considerar para la predicción de genes ncRNAs (Trotta, 2014).

2.3.2.1. Predicción de estructura secundaria y cálculo de emp

Tomando como base el *algoritmo de Zuker*[10] (Zuker & Stiegler, 1981) se han desarrollado aplicaciones, como por ejemplo el paquete de software de la Universidad de Viena (Hofacker, 2009), el cual se ha convertido en una herramienta indispensable para investigaciones en el campo de la bioinformática. Esta herramienta tiene aplicaciones para el cálculo de emp y aplicaciones que dibujan la estructura secundaria de una secuencia.

Además de aplicaciones destinadas a equipos de escritorio, portátiles o clusters de computadores, el algoritmo de Zuker y un conjunto completo de herramientas fueron adaptados para su uso como un servidor Web, el cual es un paquete llamado Mfold[11] (Zuker, 2003). La mayoría de las herramientas que ofrece Mfold están destinadas a la predicción de la estructura secundaria y estimación de emp y gracias a sus intuitivas interfaces de usuario es muy sencillo de utilizar. Su limitante es que esta diseñado para procesar secuencias cortas y no secuencias de genomas, lo cual es lógico debido a que no se calculan emp sobre genomas, sino sobre subsecuencias de un largo mucho menor (ej: 100, 1000, 2000, etc) que el total de la secuencia (ej: aproximadamente 4.6 millones de nucleótidos en el genoma de la Echerichia Coli).

Otra aplicación es GTfold[12] (Swenson et al., 2012), la cual realiza predicciones de la estructura secundaria de genes de ARN implementando el algoritmo de Zuker y además inte-

[10] El algoritmo de Zuker es descrito en detalle en el Anexo A.1.3

[11] Mfold: La descripción de este paquete de software, manual de instalación, uso y descarga esta disponible en el URL http://unafold.rna.albany.edu/?q=mfold.

[12] GTfold: La descripción de este software, manual de instalación, uso y descarga esta disponible en el URL http://gtfold.sourceforge.net/

gra directivas de OpenMP para realizar los cálculos de emp de manera paralela, lo cual permite procesar sub-secuencias de mayor tamaño en comparación con Mfold.

2.3.3. Computación Paralela para la Predicción de ncRNAs en Genomas

En varias investigaciones que analizan genes de ARN se han implementado paradigmas de *computación paralela*[13]. Estas investigaciones están netamente enfocadas en la predicción de estructuras secundarias como es el caso de GTfold y no a la búsqueda y/o clasificación de genes ncRNAs. Los paradigmas de computación paralela se han implementado utilizando diferentes conceptos de bioinformática para lograr predicciones más rápidas de la estructura secundaria.

En (Nakaya et al., 1995) se implementó el criterio de energía mínima libre de manera que se enumeran todas las regiones de una secuencia de ARN y combina aquellas que pueden coexistir para producir múltiples estructuras secundarias usando computo paralelo. En (Taufer et al., 2008) se segmenta la cadena de ARN de manera sistemática y la reconstruye combinando las estructuras de los segmentos, donde este proceso se realiza de manera paralela. Los algoritmos de predicción paralelos descritos en (Nakaya et al., 1995) y (Taufer et al., 2008) muestran mejoras en tiempo de ejecución, pero el detalle de sus implementaciones no es de acceso publico y por lo tanto se desconoce su real alcance en lo que se refiere a rendimiento computacional en la actualidad.

En general, las propuestas disponibles en la literatura publica solo aplican paralelismo para la estimación de emp, la cual es una variable requerida para el calculo de Z-score, en que Z-score es la variable descriptiva más significativa para el reconocimiento de ncRNAs cuando se realiza predicción de ncRNAs usando métodos *de novo*. Además, tal como se comentó en la Sección 1.2, el cálculo es Z-score tiene un alto costo computacional, debido al cálculo de muchas repeticiones de emp.

En este contexto, no existen aplicaciones o algoritmos paralelos para el cálculo de Z-score, donde el tiempo de predictores que usan redes neuronales es mínimo con tiempos menores al milisegundo (Rojas et al., 2016) (ver Tabla 2.1) en comparación con lo consumido por el cálculo de Z-score al procesar sub-secuencias donde el tiempo es del orden de segundos a minutos (ver Tabla 2.2). Por tanto al procesar un genoma completo compuesto por miles de secuencias puede tomar desde horas hasta varios días de tiempo de cómputo. Entonces, para

[13]Computación paralela: Como complemento a esta Sección en el Anexo B se presenta un resumen de conceptos básicos de computación paralela.

Tabla 2.1: Tiempos promedio de ejecución en nanosegundos al procesar predicciones con una red neuronal básica que recibe como entrada un vector de seis descriptores.

Red Neuronal				
Capa oculta	5 neuronas	10 neuronas	25 neuronas	50 neuronas
Tiempo	6844 ns	9150 ns	14108 ns	23397 ns

Fuente: Modificada de (Rojas et al., 2016), 2017.

Tabla 2.2: Tiempos promedio de ejecución en segundos al procesar secuencias de diferente largo para el cálculo de Z-score (obtenidos en la fase de análisis de esta memoria).

Tiempo (segundos) & largo de secuencias				
Largo	50	100	200	300
Tiempo	1.9 s	32.2 s	125.3 s	492.9 s

Fuente: Elaboración propia, 2017.

acelerar la predicción de genes ncRNAs de un genoma se debe focalizar la atención en optimizar el pre-procesamiento del genoma, la extracción de descriptores como lo es Z-score y no dedicar esfuerzos de optimización paralela al modelo de regresión usado en la predicción. Además, es importante mencionar que el Z-score tiende a ser significativamente menor en secuencias que pueden contener ncRNAs, un indicio que aumenta significativamente la posible existencia de ncRNAs en la secuencia de ARN estudiada (Freyhult et al., 2005), por este motivo y los mencionados en secciones anteriores, es que este trabajo de memoria se enfoca al cálculo paralelo de esta característica (Z-score) en sub-secuencias de una secuencia genómica.

2.4. GENOMA DE ESTUDIO Y CÁLCULO DE Z-SCORE

Al realizar experimentos sobre una secuencia genómica, es importante determinar con qué secuencia se trabajará y de que organismo se obtendrá, ya que la cantidad de información disponible dependerá del conocimiento que se tenga tanto del organismo como de la secuencia en estudio (investigaciones en las cuales ha sido utilizada) (Blattner et al., 1997; Rogers, 2012).

Para esta memoria se utilizó el genoma de la bacteria Escherichia Coli, cepa K-12

MG1655. La cepa de laboratorio K12 de la Escherichia Coli ha sido intensamente utilizada como un organismo modelo en estudios de evolución experimental (Papadopoulos et al., 1999). Debido a lo cual existe una gran variedad de información disponible sobre este organismo. Además, se acota el organismo de estudio convenientemente ya que la extensión de las secuencias genómicas bacterianas es menor a secuencias de otros organismos (Blattner et al., 1997).

2.4.1. Genoma de Escherichia Coli

La obtención de secuencias genómicas es a través del sitio Web proporcionado por el NCBI[14] (Sayers, 2009), donde se accede a la base de datos GenBank[15] (Benson et al., 2011).

El genoma a procesar será el de la bacteria Escherichia Coli, la cual es uno de los organismos más conocidos del mundo, ya que se ha utilizado como modelo para estudiar todo tipo de aspectos de genética y fisiología. Su genoma entero se conoce desde hace algunos años (Blattner et al., 1997) y su biología general esta bien estudiada. También ha sido utilizada como organismo modelo en estudios de evolución experimental y genética de poblaciones para entender la acción de las diferentes fuerzas evolutivas a corto y largo plazo (Papadopoulos et al., 1999).

2.4.2. Cálculo de Z-score

El valor Z o Z-score, corresponde al número de desviaciones estándar por las cuales la energía mínima de plegamiento de la secuencia original se desvía de la energía mínima de plegamiento de un promedio de secuencias aleatorias (Freyhult et al., 2005; Washietl et al., 2005). En la Ecuación (2.1), se muestra la forma de calcular el valor Z, siendo emp la energía mínima de plegamiento de la secuencia original, mientras que μ y σ representan respectivamente el promedio y la desviación estándar del conjunto de emp de 1000 secuencias aleatorias.

$$Z\text{-score} = \frac{emp - \mu}{\sigma} \tag{2.1}$$

[14]NCBI: El Centro Nacional para la Información Biotecnológica o National Center for Biotechnology Information es parte de la Biblioteca Nacional de Medicina de Estados Unidos, una rama de los Institutos Nacionales de Salud, disponible en el enlace https://www.ncbi.nlm.nih.gov/

[15]GenBank: Base de datos de secuencias genéticas NIH, una colección anotada de todas las secuencias de ADN disponibles públicamente. Es parte de la Colaboración internacional en bases de datos de secuencias de nucleótidos, disponible en el enlace https://www.ncbi.nlm.nih.gov/genbank/

La emp se calcula teniendo en cuenta los valores de energía mínimos obtenidos por pares de bases complementarias disminuido por la energía de apilamiento de pares de bases sucesivas o incrementado por la energía desestabilizadora asociado con bases no complementarias (Zuker & Stiegler, 1981). El cálculo de la emp se puede obtener con el algoritmo de Zuker, descrito en detalle en el Anexo A.1.

CAPÍTULO 3. PROPUESTA

En este Capítulo se presentan los modelos y planteamientos de las soluciones propuestas. El Capítulo esta organizado de la siguiente manera.

La Sección 3.1 presenta el análisis del problema. La Sección 3.2 presenta los Requerimientos de la solución. La Sección 3.3 presenta las herramientas de desarrollo. Finalmente, la Sección 3.4 presenta las soluciones propuestas de este trabajo de memoria.

3.1. ANÁLISIS DEL PROBLEMA

Como se comentó en la Sección 1.2 el cálculo de Z-score tiene un alto costo temporal, debido a que se enfoca también en el cálculo de muchas otras características de la secuencia en estudio. Para realizar el cálculo de Z-score se debe obtener la energía mínima de plegamiento (emp) de la secuencia original (Rogers, 2012). La obtención de esta energía se realiza mediante el análisis de la estructura secundaria, formada por combinaciones de las bases propias del ARN.

El Z-score es una variable de tipo estadístico que utiliza tanto la energía mínima de plegamiento, como la desviación estándar y la media de cada secuencia existente en una muestra. Esta muestra, en su totalidad es construida a partir de la generación de N secuencias aleatorias, de misma composición. Debido a que la cantidad de secuencias generadas ayudan a minimizar el error obtenido y que éstas deben tener una composición idéntica a la secuencia, el tiempo de procesar las secuencias de ARN y obtener las características que requiere el cálculo de Z-score es alto en términos de cómputo.

3.1.1. Algoritmo Paralelo por Dominio

De manera intuitiva la primera alternativa de algoritmo paralelo es dividir la secuencia del genoma completo según número de threads y de esta manera distribuir los cálculos de Z-score, donde cada thread obtendrá los 1000 cálculos requeridos para cada valor de Z-score utilizando la Ecuación (2.1). Esta estrategia es simple de programar, vergonzosamente paralela (distribución del *dominio del problema* y *escalamiento lineal*) y no presenta mayor innovación. Lo cual se debe a que la velocidad de cómputo total dependerá siempre de los recursos de procesador y threads disponibles en la arquitectura de CPU. Por lo tanto es necesario estudiar y proponer otro tipo de estrategia que permita procesar de manera mucho más eficiente una se-

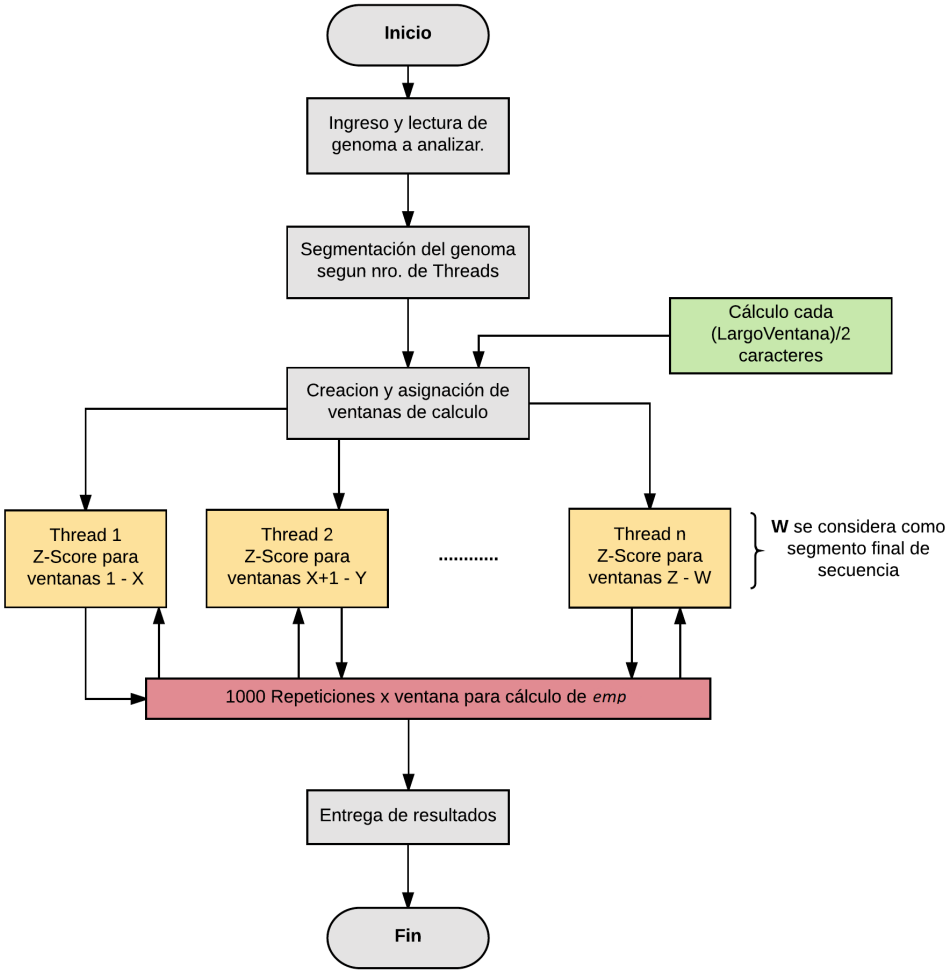


Figura 3.1: Diagrama de flujo de datos del algoritmo de dominio paralelo.

Fuente: Elaboración propia, 2017.

cuencia genómica.

A modo explicativo en la Figura 3.1 se presenta un diagrama del algoritmo que divide el problema por dominio y en el Algoritmo 1 se describe en pseudo-código su funcionamiento.

El Algoritmo 1 trabaja de la siguiente manera:

Paso 1: se ingresa el genoma para su análisis.

Paso 2: se segmenta el genoma según el número de threads disponibles para el

Algoritmo 1: Algoritmo paralelo por dominio.

Paso 1: Ingresa el genoma para su análisis.

Paso 2: Segmentación del genoma dependiendo del número de threads.

Paso 3: Partición de ventanas de cálculo.

Paso 4: Iniciar threads y asignación de segmentos correspondientes a cada uno.

Paso 5: Cómputo de Z-score.

Paso 6: Entrega de resultados y termino del algoritmo.

cálculo. Debido a que es una paralelización por dominio, la secuencia a analizar se divide en la misma cantidad de Threads que se dispongan para el cálculo, por lo que cada thread tiene un segmento del mismo largo, siendo el ultimo Thread quien puede llegar a tener un segmento que varíe de tamaño en relación a los demás.

Paso 3: se particionan las ventanas de cálculo. Dado que se analiza solo un largo de segmento, la dimensión de las ventanas será la misma que el largo de segmento ingresado.

Paso 4: se inician los threads y se asigna el segmento correspondiente a cada uno. Estos tendrán 1000 iteraciones para calcular las emp y el correspondiente Z-score por ventana. Por ejemplo, si la secuencia fuese de largo 1000 y se tuviese 4 threads disponibles, el primer thread tendría como inicio la posición 1 y como fin la posición 250. El segundo thread tendría como inicio la posición 251 y como fin la posición 500, y así sucesivamente. Cada thread tendrá asignado una secuencia del genoma completo y realizará los cálculos correspondientes.

Paso 5: se realizan los cómputos de Z-score correspondientes.

Paso 6: se almacena los resultados en un archivo y finaliza el algoritmo.

Debido a la poca innovación del Algoritmo 1, es necesario profundizar en el diseño de un nuevo algoritmo paralelo y cómo poder optimizar su funcionamiento. Además, tal como se comentó en la Sección 2.3.3 y como se muestra en la Tabla 2.2 los tiempos de procesar diferentes tamaños de sub-secuencias son diferentes, lo cual es evidencia que la asignación de un solo thread para secuencias grandes produciría cuellos de botella, debido a la lentitud de estos procesamientos. Por lo tanto se debe analizar una estrategia que disminuya los tiempos de computo para sub-secuencias que tienen mayor costo computacional, donde en especifico se debe disminuir el tiempo de procesamiento de Z-score.

En este contexto, al analizar el problema se concluye que la estrategia a proponer debe tener los siguientes requerimientos:

- Pre-procesamiento del genoma de ADN/ARN: Se debe realizar una transformación de la secuencia de ADN a una secuencia de ARN.
- Obtención de sub-secuencias: Se realiza un particionamiento de la secuencia de origen, en grupos de sub-secuencias de diferentes tamaños.
- Cálculo de emp mediante threads: Se distribuye el cómputo de los cálculos de emp en varios threads que procesarán la misma sub-secuencia del genoma en estudio.
- Cálculo de Z-score: Un thread debe calcular el valor final de Z-score, donde los demás thread colaboran en el cálculo, aportando los valores de emp que obtuvo cada uno.
- Se debe incorporar un gestor de uso de threads que según costo computacional asignar un número de thread suficiente para minimizar el tiempo de computo de Z-score.
- El gestor no debe influir en el costo total de procesamiento de la secuencia completa del genoma.

Es importante destacar que el Algoritmo 1 divide el dominio de la secuencia en los threads, pero si uno termina antes que otro de trabajar, este se libera, no utilizando al máximo los recursos disponibles. En cambio, dado los requerimientos de la propuesta de solución, específicamente en el momento en que los threads terminan su funcionamiento y entregan sus resultados, estos no se liberan, lo que les permite pasar a otras tareas y no quedar en un estado de espera hasta que todos los demás threads finalicen, utilizando todos los recursos de CPU disponibles. En la Figura 3.2, se expresan gráficamente de manera general las dos soluciones y cómo se aplican en ellas el paradigma de computación paralela. La Figura 3.2a explica el funcionamiento de la solución paralela por dominio del genoma, en cambio la Figura 3.2b explica el funcionamiento de la solución paralela que asigna de forma dinámica los trabajos y que no deja threads sin trabajo, es decir sin perdida de recursos de CPU en el procesamiento de la secuencia genómica.

3.1.2. Paradigma de Solución

En este trabajo se utiliza Pthread para la implementación de los algoritmos paralelos. Una de las principales ventajas de usar esta librería es la protección que se puede realizar para evitar las múltiples escrituras sobre una estructura de datos en memoria. Esto se logra utilizando la sentencia *pthread_barrier*, la que impide que más de un thread alcance a realizar funciones que

Algoritmo 2: Algoritmo paralelo adaptativo con gestor de trabajo.

Paso 1: Ingresa el genoma para su análisis.

Paso 2: Cálculo del número de repeticiones por threads.

Paso 3: Lectura y segmentación del genoma en los distintos tamaños (50, 100, 150, 200, 250, 300).

Paso 4: Determinación de la configuración óptima en el periodo de calibración.

Paso 5: Cómputo de Z-score por segmentos, según gestor de trabajo.

Paso 6: Asignación de trabajo a threads libres, según gestor de trabajo

Paso 7: Entrega de resultados y termino del algoritmo.

estén demarcadas por un *pthread_barrier*. Gracias a esto todos los threads pueden realizar sus respectivos procesamientos y en el momento en que todos terminen, solo un thread realiza los cálculos utilizando los valores obtenidos en todos los threads. Mayor detalle sobre el lenguaje de programación y librerías de desarrollo utilizadas en esta memoria esta disponible para su lectura en el Anexo C.

A continuación se explica la propuesta de algoritmo paralelo, que incorpora los requerimientos descritos en la Sección 3.1.1

3.2. SOLUCIÓN PROPUESTA

En la Figura 3.3 se presenta el diagrama de flujo de datos de la propuesta, con sus entradas, ciclos, condiciones y salidas. Como también, la información detallada del funcionamiento. El algoritmo que incorpora el gestor descrito en la Figura 3.3 se presenta en pseudo-codigo en el Algoritmo 2.

El funcionamiento del Algoritmo 2 es el siguiente:

Paso 1: se ingresa el genoma para su análisis. Esta solución permite el análisis de hasta 6 largos de segmentos (50,100,150,200,250 y 300).

Paso 2: se calcula el número de repeticiones por thread. Teniendo en cuenta que para los cálculos que se buscan se debe iterar 1000 veces, estas se dividirán según el número de threads asignados para un largo especifico. Por ejemplo, si para el análisis de un segmento de largo x se asignan 4 threads, cada uno realizaría los cálculos sobre el mismo segmento, pero iteraría solo 250 veces. Luego se obtiene el promedio de todos los threads que trabajan sobre el

mismo segmento, obteniendo el resultado en un tiempo mucho más acotado.

Paso 3: se lee el genoma a analizar y se segmenta en los distintos tamaños (50, 100, 150, 200, 250 y 300), esto será guardado en memoria. De esta manera se accede a estos cuando sean necesario y es más fácil trabajar con posiciones. Así, se reduce la carga de trabajo por thread, ya que solo deben acceder a los valores almacenados en memoria y no segmentar la secuencia para un tamaño especifico cada vez sea necesario.

Paso 4: Se determina la configuración óptima mediante un periodo de calibración, lo que determina el número de threads por segmento, según la estimación a posteriori del tiempo de procesamiento de subcadenas de menor tamaño.

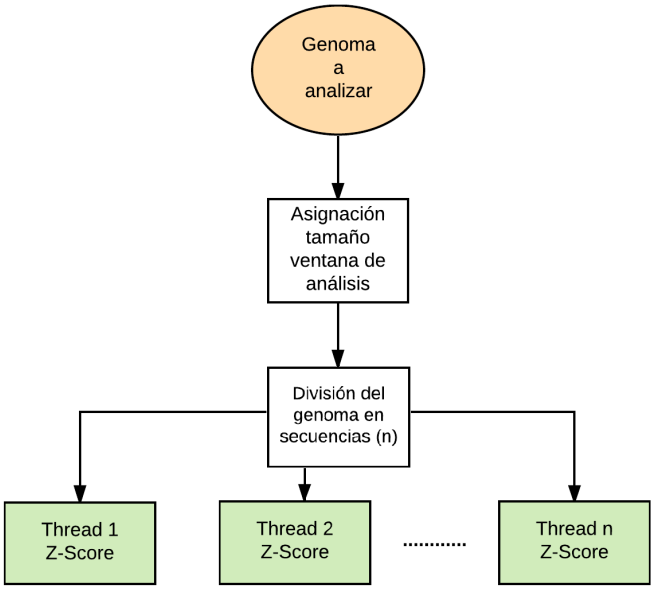
Paso 5: A medida que trabajan todos los threads se calculan por segmento el correspondiente Z-score. Para esto se aplica un bloqueo a los threads que estén trabajando sobre este segmento, y así solo uno recolecta los valores y calcula el Z-score general para el segmento completo. Al calcular, se cierra el bloqueo y los threads pasan al siguiente segmento. Esta coordinación de tareas es realizada por el gestor.

Paso 6: se asignan trabajos a los threads que hayan terminado sus correspondientes cómputos, y así colaborar con otros cómputos que aún no han finalizado. Función realizada por el gestor de trabajo.

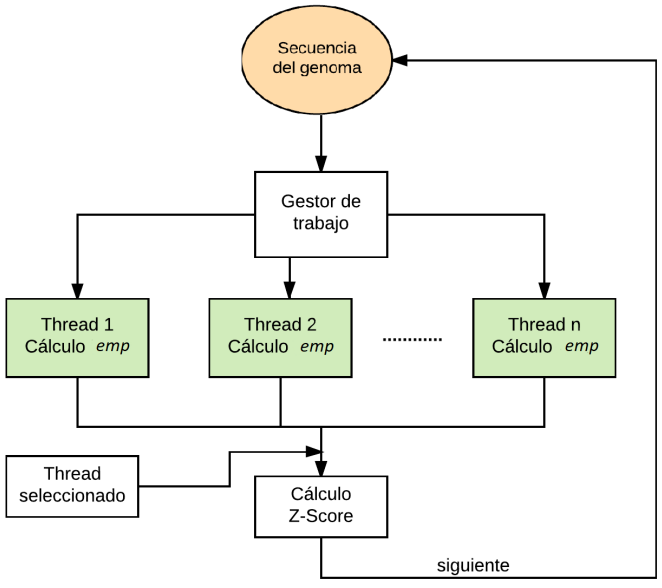
Paso 7: se entregan los resultados y finaliza el algoritmo.

La principal innovación del algoritmo propuesto es el gestor de trabajo que contiene, ya que éste será el encargado de distribuir las distintas cargas de trabajo, entre lo que gestionará está largo de segmento, el número de threads por largo de segmento, el número de repeticiones que le corresponde a cada thread, por nombrar las más relevantes.

A continuación en el Capítulo 4 se presenta la evaluación de la ejecución del algoritmo propuesto y se mide la eficacia del cálculo de Z-score.



(a)



(b)

Figura 3.2: Funcionamiento general de algoritmo paralelo por (a) dominio y (b) adaptativo.

Fuente: Elaboración propia, 2017.

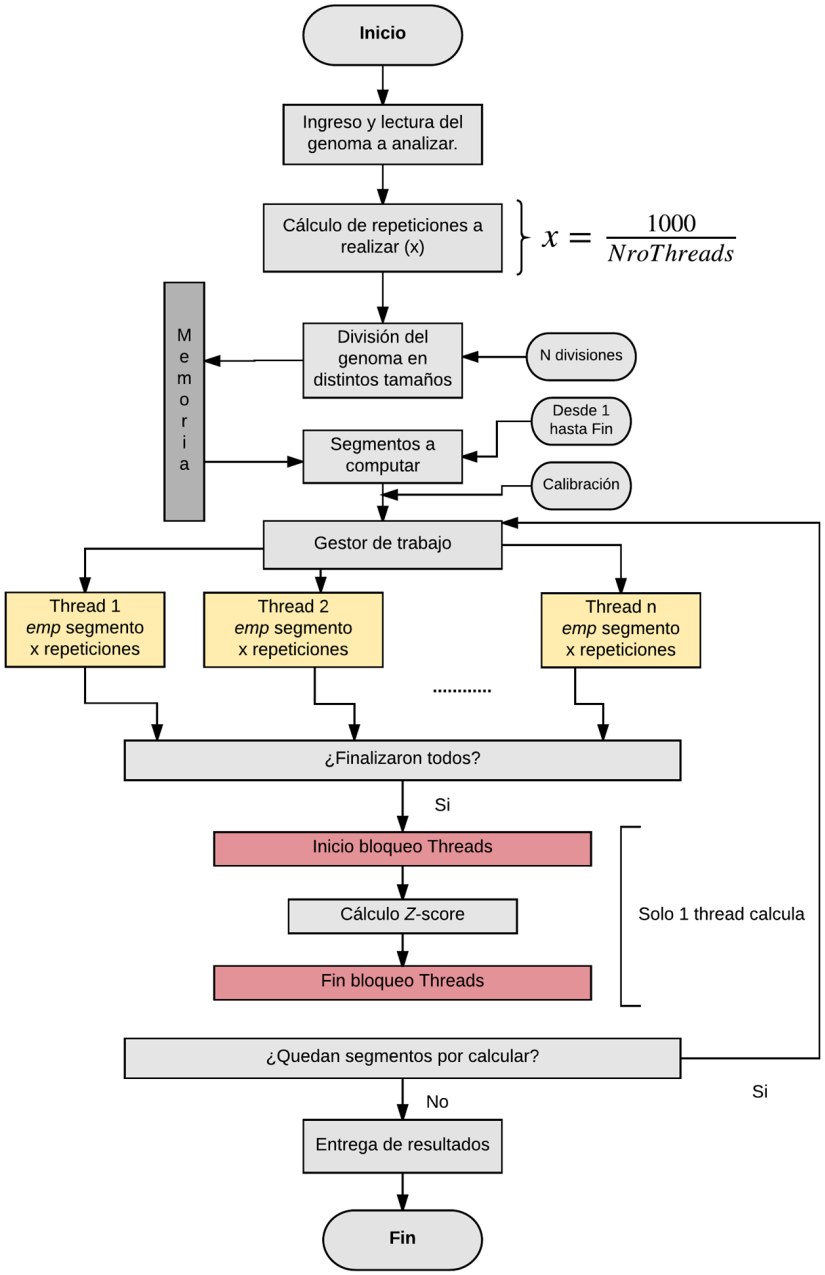


Figura 3.3: Diagrama de flujo de datos del algoritmo paralelo adaptativo con aisgnación dinámica de threads.

Fuente: Elaboración propia, 2017.

CAPÍTULO 4. ANÁLISIS DE RESULTADOS

En este Capitulo se presenta los resultados de eficiencia y eficacia de las propuestas de solución para el cálculo de Z-score con cómputo paralelo. El Capítulo esta organizado de la siguiente manera. La Sección 4.1 presenta la configuración de experimentos. La Sección 4.2 presenta la evaluación de eficiencia. La Sección 4.3 presenta la evaluación de eficacia. Finalmente, en la Sección 4.4 se presenta las conclusiones del análisis de resultados.

4.1. CONFIGURACIÓN DE EXPERIMENTOS

A continuación se describe el hardware utilizado y se describen los experimentos realizados para la medición de eficiencia de los algoritmos y la eficacia del cálculo de Z-score.

4.1.1. Hardware

El hardware de pruebas corresponde a un nodo de un cluster con capacidad de 32 threads con procesadores Xeon de 2.4 Ghz y con 128 Gbytes de memoria. Sistema operativo GNU Linux Ubuntu 14.04 64 Bits.

4.1.2. Medición de Eficiencia

Para medir la eficiencia se realizan pruebas comparativas que miden el costo temporal de los algoritmos. Para ello se realizan pruebas donde se evalúa el tiempo de ejecución de las versiones secuenciales y paralelas. Además, se mide la aceleración de los algoritmos paralelos (Speed-Up en inglés) y se realiza un análisis del uso del gestor dinámico aplicado al procesamiento paralelo adaptativo propuesto en esta memoria, utilizando técnicas de simulación de cómputo paralelo. Para la evaluación de los algoritmos paralelos se utilizan 1, 2, 4, 8 ,16 y 32 threads de la arquitectura descrita en la Sección 4.1.1.

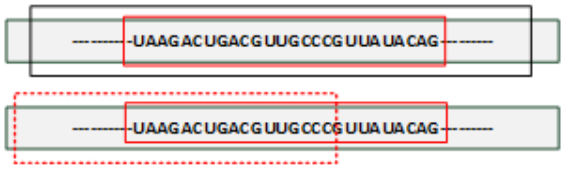


Figura 4.1: Desplazamiento en la secuencia del genoma.
Fuente: Elaboración propia, 2017.

4.1.3. Medición de Eficacia

Para medir la eficacia del cálculo de Z-score, se comparan los resultados obtenidos por las técnicas paralelas con lo reportado por la API Bioperl[1], que es un herramienta de bioinformática muy usada por la comunidad científica del área de bioinformática. Esta herramienta se basa en módulos de Perl[2]. Dichos módulos permiten procesar datos de secuencias, alineamientos de ellas, extraer características, localizaciones génicas, por nombrar algunas de sus características.

En especifico la API Bioperl se utiliza para: a) la obtención de la cadena de ARN a partir de una secuencia de ADN, b) realizar los cómputos de los valores de emp y c) calcular el valor de Z-score. Luego, los resultados de Z-score obtenidos con la API de Bioperl son comparados y evaluados con lo reportado por la propuesta de algoritmo paralelo adaptativo.

4.1.4. Preparación de Datos

Para las pruebas se usa la secuencia de ADN del genoma de la Echerichia Coli K-12 MG1655 (E. Coli de ahora en adelante), la cual se obtiene desde la base de datos GenBank descrita en la Sección 2.4.1. Luego, se procesa la secuencia de ADN de la E. Coli (4641652 nucleótidos) realizando la transformación de ADN a ARN usando la API Bioperl.

Posteriormente se utiliza la *hebra positiva*[3] de ARN y se realiza un muestreo de sub-

[1]Bioperl: Proyecto activo de código abierto para Bioinformática y genómica. Financiado por Open Bioinformatics Foundation, disponible en enlace http://bioperl.org

[2]Perl: Lenguaje de programación caracterizado por su destreza en el procesado de texto y sus ínfimas limitaciones comparado con otros lenguajes de script.

[3]Hebra positiva: La molécula de ADN esta formada de dos cadenas o hebras, una positiva y una negativa, las cuales tienen sentidos opuestos de orientación y como las cadenas de ARN están acopladas en cada hebra, se produce una secuenciación que se identifica con la misma orientación de la hebra de ADN que se esta procesando.

Tabla 4.1: Número de secuencias generadas según el largo de cada sub-secuencia de nucleótidos de la E. Coli.

Nucleótidos	**Secuencias**	**Porcentaje**
50	186501	41 %
100	90976	20 %
150	63683	14 %
200	45488	10 %
250	36390	8 %
300	31842	7 %

Fuente: Elaboración propia, 2017.

secuencias de largos 50, 100, 150, 200, 250 y 300 nucleótidos, realizando un desplazamiento en el genoma proporcional a la mitad de cada largo (25, 50, 75, 100, 125 y 150 respectivamente). Este desplazamiento se ilustra en la Figura 4.1.

Como resultado del muestreo se obtienen un total de 454880 sub-secuencias genomicas. Lo cual da como resultado un total de 186501 secuencias de largo 50, 90976 secuencias de largo 100, 63683 secuencias de largo 150, 45488 secuencias de largo 200, 36390 secuencias de largo 250 y 31842 secuencias de largo 300. Esta distribución y el porcentaje relativo a las 454880 sub-secuencias se muestra en la Tabla 4.1.

Para los experimentos que miden la eficiencia de los algoritmos se utilizo solo el 1 % de las sub-secuencias, lo que es equivalente al 1 % de la secuencia completa de la E. Coli. Estas mediciones corresponden a los experimentos de tiempo real de ejecución, los cuales permiten proyectar a través de técnicas de simulación una estimación global del tiempo de procesamiento del genoma completo de la E. Coli.

Se realiza de esta manera, debido a que los recursos disponibles para los experimentos (hardware descrito en la Sección 4.1.1) no pueden ser utilizados de forma exclusiva por periodos extensos (varios días de cómputo) ya que son requeridos para diversas investigaciones del área de bioinformática y computación paralela de grupo de investigación Citiaps de la Universidad de Santiago. Es por ello que la evaluación del procesamiento completo de la secuencia del genoma de la E. Coli, se realiza a través de la simulación, donde dichos análisis y simulaciones fueron realizadas usando el software matemático Matlab.

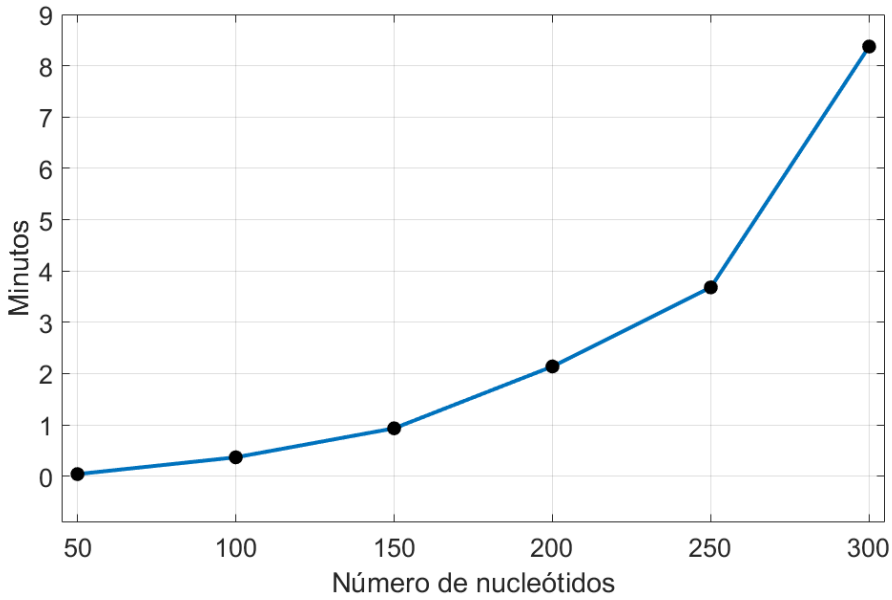


Figura 4.2: Tiempos promedio de algoritmo secuencial para el cálculo de Z-score.
Fuente: Elaboración propia, 2017.

4.2. EVALUACIÓN DE EFICIENCIA

A continuación, se presenta el análisis de eficiencia de los algoritmos: a) algoritmo secuencial, b) algoritmo paralelo y c) algoritmo paralelo adaptativo.

4.2.1. Eficiencia Algoritmo Secuencial

En la Figura 4.2 se muestra el tiempo promedio del cómputo secuencial de Z-score al procesar las secuencias de largo 50, 100, 150, 200, 250 y 300 nucleótidos. Como se puede apreciar en la Figura 4.2, el algoritmo secuencial tiene una directa proporcionalidad entre el largo de la secuencia (número de nucleótidos) y el tiempo de cómputo usado para el cálculo de Z-score. La gráfica de la Figura 4.2 proporciona una idea del gran costo de cómputo requerido al procesar sub-secuencia para obtener Z-score, donde por ejemplo al procesar la secuencia completa de la E.Coli usando sub-secuencias de 300 nucleótidos (31842 sub-secuencias), el algoritmo secuencial demoraría un tiempo no menor a 176 días de cómputo ($31842 \times 8min/60min/24hrs$).

Por lo anterior, es que el uso de computación paralela es requerida para resolver

estos problemas que usan gran tiempo cómputo, donde además se debe agregar que se deben procesar las secuencias a diferentes tamaños y es evidente que un procesamiento secuencial requeriría muchos más días (mayor a 176 días). A su vez la implementación paralela que se utilice, debe usar los recursos de procesador de forma eficiente, principalmente para no usar un número mayor de recursos de CPU, donde un mayor uso de procesadores no asegura en la realidad un menor tiempo de cómputo al procesar la secuencia completa de un genoma, sobre todo si los tiempos no se reducen cercanos a la razón de $T_1(P)/N$, donde T_1 es el tiempo reportado al usar un solo procesador y/o thread asignado a un problema P y N el número de procesadores y/o threads asignados para resolver el problema P.

A continuación se presenta el análisis de eficiencia del algoritmo paralelo propuesto en este trabajo y cómo se puede optimizar su uso, utilizando un gestor de asignación de trabajo de threads, donde la combinación del algoritmo paralelo con el gestor, es en definitiva el algoritmo paralelo adaptativo propuesto en esta memoria.

4.2.2. Eficiencia Algoritmo Paralelo

En la Figura 4.3 se muestra los tiempos promedio (en minutos) de procesamiento según número de threads asignados para cada largo de secuencias (número de nucleótidos), donde se puede apreciar que para secuencias de menores número de nucleótidos la ganancia de usar un número mayor de threads no es alta (ej. 50, 100 y 150), no así cuando aumenta el número de nucleótidos donde la disminución del tiempo de procesamiento a medida que se utiliza un número mayor de threads es evidente (ej: 200, 250 y 300). Lo cual se confirma al analizar los Speed-Up del algoritmo, donde para secuencias de menor tamaño el Speed-Up es mínimo e incluso decrece cuando se usan 32 threads. Esto último es prueba de que para secuencias de rápido procesamiento secuencial el uso de un mayor número de threads no tiene sentido.

A pesar de la baja aceleración para secuencias de menor tamaño, el algoritmo paralelo propuesto logra en la mayoría de los casos de estudio un Speed-Up con pendiente positiva y a medida que aumenta el costo de cómputo de procesar secuencias de mayor tamaño, la propuesta logra disminuir el tiempo de procesamiento cercana a un Speed-Up lineal, como es el caso del procesamiento de 300 nucleótidos donde el valor de Speed-Up es cercano a 25, en que el caso lineal seria de 32 ($T_1/32$).

Es importante destacar que las mediciones de Speed-Up son realizadas con una implementación síncrona del algoritmo paralelo, es decir no se procesa una sub-secuencia hasta

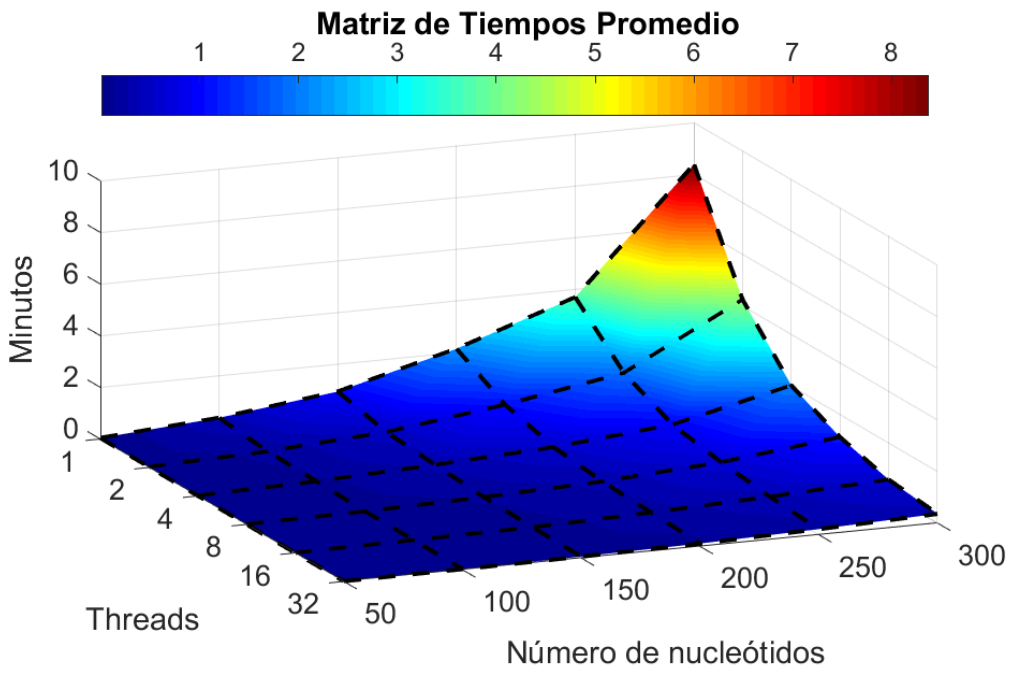


Figura 4.3: Matriz de tiempos promedios de cómputo.

Fuente: Elaboración propia, 2017.

que termina la actual y por tanto los tiempos de procesamiento medidos no se ven afectados por el trabajo que realizan otros threads. Se realiza de esta manera con el objetivo de medir de forma precisa los tiempos de procesamiento a medida que se procesa el mismo conjunto de subsecuencias a un número mayor de threads. Además, medido de esta manera se minimiza el efecto de la arquitectura interna del procesador, como lo son las velocidades del procesador, tamaños de las memorias cache y sus velocidades de transferencia, el tipo de arquitectura multi-core (ej: cuantos threads por core).

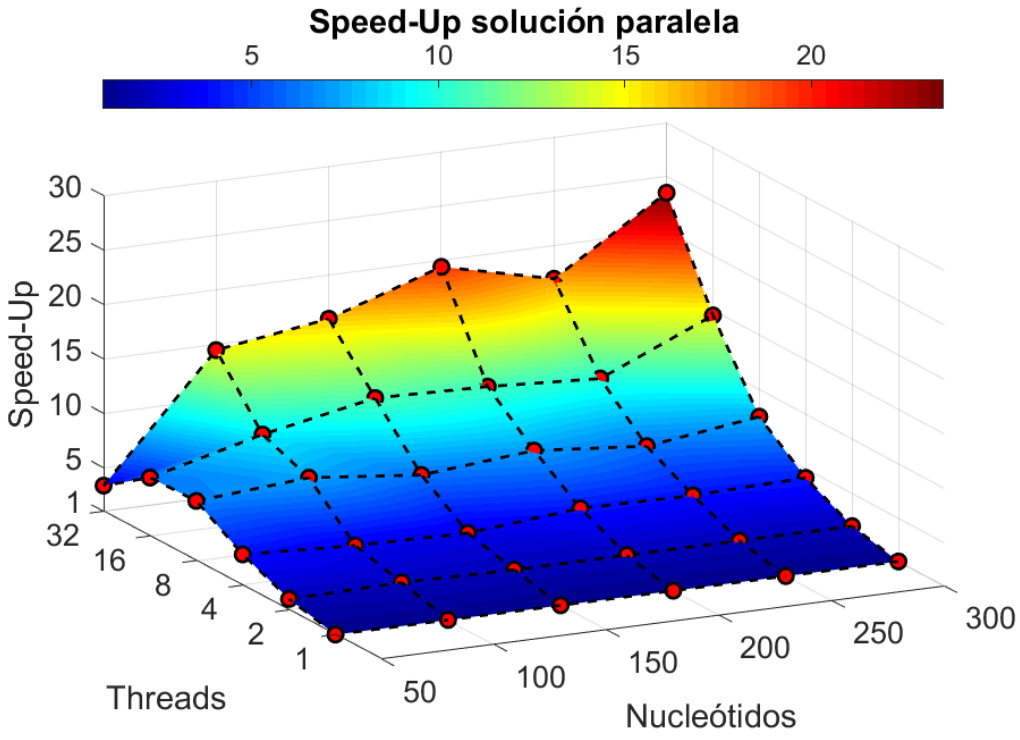


Figura 4.4: Speed-Up de algoritmo paralelo según número de nucleótidos.

Fuente: Elaboración propia, 2017.

En este contexto, es relevante la medición de tiempos de forma síncrona al momento de comparar resultados con otros experimentos sobre diferentes arquitecturas de procesador, ya que evidentemente una arquitectura más moderna obtendría mejores resultados, pero como el objetivo no es evaluar la arquitectura de CPU, se evaluó de forma empírica experimental los tiempos de procesamiento y la aceleración del algoritmo paralelo con el menor ruido que pudiese afectar a la medición de tiempos.

4.2.3. Eficiencia Algoritmo Paralelo Adaptativo

En la ejecución del algoritmo paralelo adaptativo se utilizan los tiempos que se presentan en la Figura 4.3. Se utilizan estos tiempos para determinar cuál es la configuración optima de asignación de threads para el procesamiento del genoma de la E. Coli usando el número de secuencias presentado en la Tabla 4.1. En este caso se realiza una simulación asíncrona que proyecta el cómputo de procesar el número total de sub-secuencias de distinto largo. Este proceso

se presenta en la Figura 4.5, donde se realizan 32 procesos P de 1 thread (T), 16 procesos que usan 2 thread y así sucesivamente hasta usar los 32 thread para un solo proceso P. Luego, al estimar en el periodo de calibración los tiempos de cómputo globales se determina la configuración que asegura un mínimo tiempo de procesamiento para procesar una secuencia genómica.

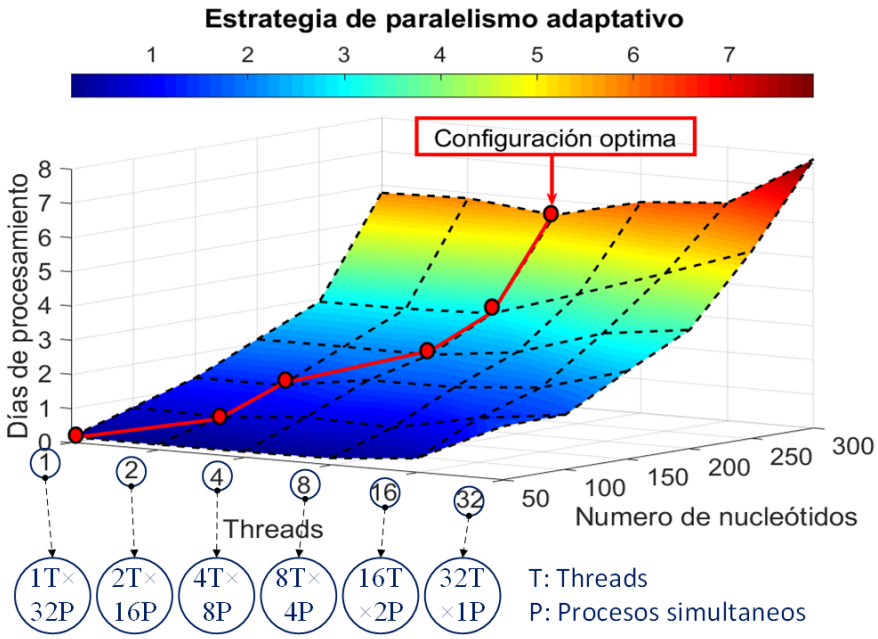


Figura 4.5: Estrategia de paralelismo adaptativo.

Fuente: Elaboración propia, 2017.

Para ello se utilizan los mínimos tiempos reportados en la fase de calibración, lo cual se presenta en la Figura 4.5 con una linea demarcada con círculos, que se puede interpretar como el cauce de un río, donde el flujo de agua que presenta mayor velocidad o bien la unión de puntos donde se produce una mayor pendiente. La Figura 4.5 muestra que para procesar secuencias de largo 50 se debe usar un thread, para largos 100 y 150 usar dos threads, para largos 200, 250 y 300 usar cuatro threads, lo cual confirma nuevamente que no es necesario utilizar todos los recursos de procesador para procesar una sub-secuencia, sino que el uso de procesadores y/o threads se puede adaptar de forma dinámica según el tamaño del problema, que en este caso depende del tamaño de las secuencias a procesar.

El resultado global de aplicar la técnica adaptativa, es que el genoma de la E. Coli puede ser procesada en aproximadamente 13 días de procesamiento usando los 32 threads de la

Tabla 4.2: Número de secuencias procesadas por día (tasa) según estrategia y uso de threads.

Estrategia	Estática						Adaptativa
Thread	1	2	4	8	16	32	Dinámico
Tasa	35025.6	34969.7	33986.8	30274.2	25647.4	20137.2	**35824.3**

Fuente: Elaboración propia, 2017.

arquitectura (con asignación dinámica) y asegurando un mínimo tiempo de cómputo si se compara con el algoritmo paralelo estático implementado de forma asíncrona, donde este ultimo asignaría un número fijo de threads al procesar toda la secuencia completa de la E. Coli. Estos resultados se presentan en la Tabla 4.2, donde la estrategia adaptativa que asigna dinámicamente el número de threads para computar la secuencia de la E. Coli obtiene una mayor tasa de procesamiento por día, si se compara con la asignación estática de uso de threads.

4.3. EVALUACIÓN DE EFICACIA

A continuación, se presenta las mediciones de eficacia obtenido con el algoritmo paralelo propuesto en esta memoria y se comparan los resultados entregados por las herramientas Bioperl y RNAfold. Los resultados se expresan gráficamente de la siguiente manera, en Figura 4.6 se presentan los resultados Z-score obtenidos en base a una secuencia aleatoria del genoma E. Coli. Por último, en Figura la 4.7 se presenta los resultados Z-score obtenidos en base a un gen ncRNAs, extraído desde base de datos Rfam[4].

La Figura 4.6 permite observar los valores Z-score obtenidos para secuencias de largo 50, 100, 150, 200, 250 y 300. Cada una de estas secuencias fueron computadas con el algoritmo paralelo adaptativo desarrollado en esta memoria, con la herramienta RNAFold y con la API Bioperl. Por ejemplo, para una secuencia de 50 nucleótidos de largo, el resultado entregado por el algoritmo adaptativo alcanza un error relativo[5] del 24 % , considerando las otras herramientas como valores exactos. Siendo este error el mayor observado en la gráfica de experimentos.

La Figura 4.7 presenta los resultados obtenidos de un gen de ncRNAs. Como se

[4]Rfam: Colección de familias de ARN, cada una representada por alineaciones de secuencias múltiples, consenso estructuras secundarias y modelos de covarianza

[5]Error relativo: Es el cociente de la división entre el error absoluto y el valor exacto.

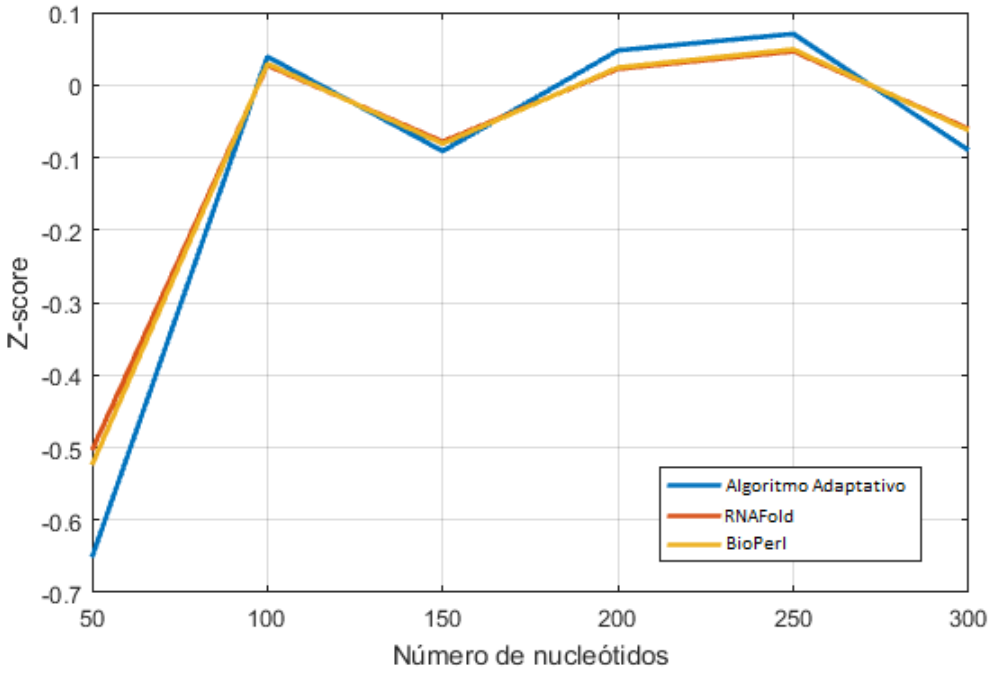


Figura 4.6: Z-score obtenidos según herramientas utilizadas, sobre secuencia aleatoria.
Fuente: Elaboración propia, 2017.

puede apreciar, los resultados que entrega el algoritmo adaptativo aumentan el error a medida que los números de nucleótidos de la secuencia aumentan, a pesar de esto, los resultados del algoritmo adaptativo alcanzan un error relativo del 22 % aproximadamente, al computar una gen ncRNAs de 300 nucleótidos.

4.4. CONCLUSIONES DE EXPERIMENTOS

En los experimento presentados en este Capítulo, se han mostrado los resultados obtenidos de eficiencia y eficacia.

Respecto a la eficiencia, se realizaron análisis comparativos considerando el algoritmo secuencial y un algoritmo adaptativo paralelo síncrono. Se evidencio la directa proporcionalidad que existe entre el tiempo de cómputo usado para el cálculo de Z-score y el número de nucleótidos de la secuencia computada. En esta ocasión, se pudo observar que proyectando el tiempo de cómputo para el genoma completo, considerando secuencias de 300 nucleótidos de

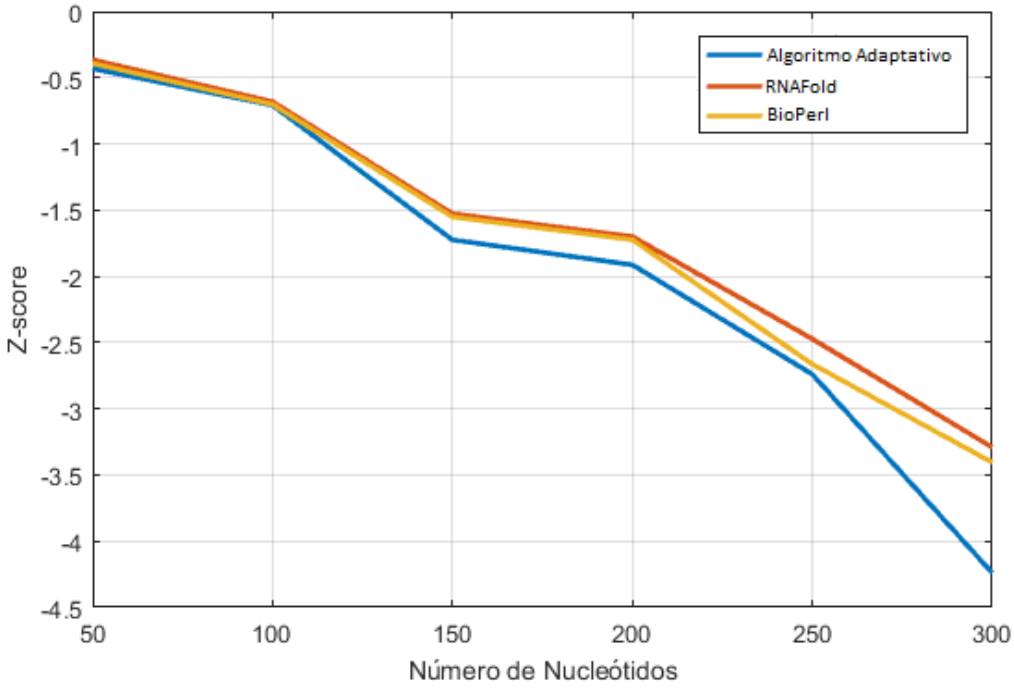


Figura 4.7: Z-score obtenidos según herramientas utilizadas, sobre gen de ncRNAs.
Fuente: Elaboración propia, 2017.

largo, éste alcanzaría los 176 días de cómputo aproximadamente (con los secuencias de largo 300). Estos resultados demuestran la necesidad de implementar técnicas de computación paralela para reducir los tiempos de cómputo total. Donde la propuesta usando 32 threads puede procesar el genoma completo de la E. Coli en 13 días (procesando todas las secuencias de largo 50, 100, 150, 200, 250 y 300).

La primera solución paralela trabaja de manera síncrona, donde a pesar de disminuir el tiempo de cómputo, el hecho de ser una solución síncrona limita la disminución del tiempo que demora computar el genoma completo. Es por ello que en un ambiente real, las soluciones paralelas deben trabajan de forma asíncrona. En este contexto las pruebas sobre el modelo de simulación asíncrono muestran que la versión paralela asíncrona que usa el gestor dinámico de uso de threads es el más eficiente, permitiendo además una configuración óptima de cargas de trabajo.

Por último, considerando los resultados de ambas proyecciones de tiempo de cómputo, se puede apreciar que la solución que utiliza un gestor de trabajo logra disminuir considera-

blemente el tiempo de cómputo total de procesar el genoma E. Coli en comparación a la versión paralela que no utiliza este gestor.

Respecto a la eficacia, los resultados obtenidos permiten demostrar que la precisión de RNAFold y de Bioperl es similar. En cambio, los resultados que entrega el algoritmo paralelo adaptativo varían poco en comparación a estas herramientas. A pesar de esta variación, el resultado del algoritmo adaptativo mantiene la tónica con respecto a los resultados de las otras herramientas. En ningún caso observado la diferencia es demasiado considerable, sobre todo considerando una medida de error relativo.

El error relativo nos permite obtener una medida descriptiva de la variación de los resultados entre las soluciones expresadas en los experimentos de eficacia. Específicamente, la Figura 4.6 permite observar que el cómputo de secuencias de 50 nucleótidos es en donde mayor diferencia existe, esto considerando los resultados del algoritmo adaptativo y las herramientas RNAfold y Bioperl. Para este análisis, se obtiene un error relativo aproximado del 24 %, lo que expresa la poca variación entre los resultados.

Por último, el error relativo obtenido al momento de computar un gen de ncRNAs es de aproximadamente un 22 %, lo que similar al apartado anterior, demuestra que el algoritmo adaptativo es una herramienta que entrega resultados similares a los que entregan las herramientas RNAfold y Bioperl. Esto se puede observar de manera gráfica en Figura 4.7.

Finalmente, se puede observar que, a pesar de que los experimentos se desarrollaron tanto sobre una secuencia aleatoria que no representaba un gen de ncRNAs, como también para un secuencia que si representaba un gen de ncRNAs, los resultados de errores relativos conservaron una media. Esto, permite concluir que a pesar de las características que presente la secuencia computada, el resultado del algoritmo adaptativo entrega una similitud considerable al compararlo con los resultados de las herramientas RNAfold y Bioperl.

CAPÍTULO 5. CONCLUSIONES

En este trabajo de memoria se diseñó e implementó un algoritmo de cómputo paralelo para la obtención de características físicas (emp y Z-score) en secuencias genómicas bacterianas útil para la detección de ARN no codificante.

El algoritmo adaptativo propuesto usa como base un algoritmo paralelo multithread que combinado con un gestor de asignación dinámica de recursos de procesador (threads), permite distribuir de manera equitativa la carga de trabajo a cada uno de los threads evitando que existan threads sin trabajo, lo cual permite usar los recursos de procesador de forma eficiente.

Como aporte de investigación de pregrado, esta solución forma parte del trabajo (Rojas et al., 2017), el cual fue presentado en el VI Workshop del Centro de Biotecnología y Bioingeniería CeBiB (CeBiB, 2017), en Junio de 2017, la cual será utilizada para tareas de investigación del área de bioninformática de la Universidad de Santiago de Chile. Específicamente, se utilizará para extraer características físicas (emp y Z-score) requeridas en la búsqueda de patrones de genes de ARN no codificantes con técnicas de inteligencia artificial.

Finalmente, un tema relevante que se deja como trabajo futuro es escalar el algoritmo paralelo adaptativo a una solución distribuida, donde el objetivo sea además de procesar de manera veloz una secuencia genómica, el estudio de técnicas de optimización dinámica para un cluster de computadores, considerando en el modelo, costos de comunicación y sincronización, donde el objetivo sea la minimización de los tiempos de cómputo y a su vez minimizar el uso de energía del cluster de computadores.

Bibliografía

Altuvia, S. (2007). Identification of bacterial small non-coding rnas: experimental approaches. *Current opinion in microbiology*, *10*(3), 257–261.

Argaman, L., Hershberg, R., Vogel, J., Bejerano, G., Wagner, E. G. H., Margalit, H., & Altuvia, S. (2001). Novel small rna-encoding genes in the intergenic regions of escherichia coli. *Current Biology*, *11*(12), 941–950.

Asanovic, K., Bodik, R., Catanzaro, B. C., Gebis, J. J., Husbands, P., Keutzer, K., Patterson, D. A., Plishker, W. L., Shalf, J., Williams, S. W., et al. (2006). The landscape of parallel computing research: A view from berkeley. Tech. rep., Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley.

Backofen, R., Bernhart, S. H., Flamm, C., Fried, C., Fritzsch, G., Hackermüller, J., Hertel, J., Hofacker, I. L., Missal, K., Mosig, A., et al. (2007). Rnas everywhere: genome-wide annotation of structured rnas. *Journal of Experimental Zoology Part B: Molecular and Developmental Evolution*, *308*(1), 1–25.

Barney, B. (2009a). Introduction to parallel computing. `https://computing.llnl.gov/tutorials/parallel_comp/`. Revisado el 24/06/2016.

Barney, B. (2009b). Openmp tutorial. `https://computing.llnl.gov/tutorials/openmp/`. Revisado el 24/06/2016.

Barney, B. (2010). Posix threads programming. `https://computing.llnl.gov/tutorials/pthreads/`. Revisado el 24/06/2016.

Bejerano-Sagie, M., & Xavier, K. B. (2007). The role of small rnas in quorum sensing. *Current opinion in microbiology*, *10*(2), 189–198.

Benson, I., D. A.and Karsch-Mizrachi, Lipman, D. J., Ostell, J., & Sayers, E. W. (2011). Genbank. *Nucleic acids research*, *41*(1), 1–4.

Blattner, F. R., Plunkett, G., Bloch, C. A., Perna, N. T., Burland, V., Riley, M., Collado-Vides, J., Glasner, J. D., Rode, C. K., Mayhew, G. F., et al. (1997). The complete genome sequence of escherichia coli k-12. *science*, *277*(5331), 1453–1462.

Bompfünewerer, A. F., Flamm, C., Fried, C., Fritzsch, G., Hofacker, I. L., Lehmann, J., Missal, K., Mosig, A., Müller, B., Prohaska, S. J., et al. (2005). Evolutionary patterns of non-coding rnas. *Theory in Biosciences*, *123*(4), 301–369.

Brouns, S. J., Jore, M. M., Lundgren, M., Westra, E. R., Slijkhuis, R. J., Snijders, A. P., Dickman, M. J., Makarova, K. S., Koonin, E. V., & Van Der Oost, J. (2008). Small crispr rnas guide antiviral defense in prokaryotes. *Science*, *321*(5891), 960–964.

CeBiB (2017). Centro de investigación de biotecnologia y bioingenieria. `http://www.cebib.cl`.

Chen, S., Lesnik, E. A., Hall, T. A., Sampath, R., Griffey, R. H., Ecker, D. J., & Blyn, L. B. (2002). A bioinformatics based approach to discover small rna genes in the escherichia coli genome. *Biosystems*, *65*(2), 157–177.

Clayton, J., & Dennis, C. (2003). Years of dna. *Nature/Palgrave Macmillan, London, England*, (pp. 12–35).

Clote, P., Ferré, F., Kranakis, E., & Krizanc, D. (2005). Structural rna has lower folding energy than random rna of the same dinucleotide frequency. *Rna*, *11*(5), 578–591.

Cook, S. (2012). *CUDA programming: a developer's guide to parallel computing with GPUs*. Newnes.

Dinger, M. E., Pang, K. C., Mercer, T. R., & Mattick, J. S. (2008). Differentiating protein-coding and noncoding rna: challenges and ambiguities. *PLoS Comput Biol*, *4*(11), e1000176.

Eddy, S. R. (1999). Noncoding rna genes. *Current opinion in genetics & development*, *9*(6), 695–699.

Eddy, S. R. (2002). Computational genomics of noncoding rna genes. *Cell*, *109*(2), 137–140.

Eddy, S. R. (2004). How do rna folding algorithms work? *Nature biotechnology*, *22*(11), 1457–1458.

Elliott, D., & Ladomery, M. (2017). *Molecular biology of RNA*. Oxford University Press.

Fontana, W. (2002). Modelling ?evo-devo?with rna. *BioEssays*, *24*(12), 1164–1177.

Freyhult, E., Gardner, P. P., & Moulton, V. (2005). A comparison of rna folding measures. *BMC bioinformatics*, *6*(1), 241.

Golub, G. H., & Ortega, J. M. (2014). *Scientific computing: an introduction with parallel computing*. Elsevier.

Gottesman, S. (2005). Micros for microbes: non-coding regulatory rnas in bacteria. *TRENDS in Genetics*, *21*(7), 399–404.

Hershberg, R., Altuvia, S., & Margalit, H. (2003). A survey of small rna-encoding genes in escherichia coli. *Nucleic acids research*, *31*(7), 1813–1820.

Hofacker, I. L. (2009). Rna secondary structure analysis using the vienna rna package. *Current protocols in bioinformatics*, (pp. 12–2).

Johansson, J., & Cossart, P. (2003). Rna-mediated control of virulence gene expression in bacterial pathogens. *Trends in microbiology*, *11*(6), 280–285.

Kaikkonen, M. U., Lam, M. T., & Glass, C. K. (2011). Non-coding rnas as regulators of gene expression and epigenetics. *Cardiovascular research*, *90*(3), 430–440.

Kavanaugh, L. A., & Dietrich, F. S. (2009). Non-coding rna prediction and verification in saccharomyces cerevisiae. *PLoS Genet*, *5*(1), e1000321.

Kin, Y. K., T., Terai, G., & Okida, Y. Y.-O. Y. â. A. K., H. (2007). frnadb: a platform for mining/annotating functional rna candidates from non-coding rna sequences. *Nucleic Acids Research*, *35*(1), 145–148.

Kulkarni, R., & Kulkarni, P. R. (2007). Computational approaches for the discovery of bacterial small rnas. *Nucleic Acids Research*, *43*(2), 131–9.

Lee, M. T., & Kim, J. (2008). Self containment, a property of modular rna structures, distinguishes micrornas. *PLoS Comput Biol*, *4*(8), e1000150.

Liu, G. J.-. R. B., J. (2007). Distinguishing protein-coding from non-coding rnas through support vector machines. *PLoS Genetics*, *2*(4), 29.

Livny, J., & Waldor, M. K. (2007). Identification of small rnas in diverse bacterial species. *Current opinion in microbiology*, *10*(2), 96–101.

Machado-Lima, A., Del Portillo, H. A., & Durham, A. M. (2008). Computational methods in noncoding rna research. *Journal of mathematical biology*, *56*(1-2), 15–49.

Matera, A. G., Terns, R. M., & Terns, M. P. (2007). Non-coding rnas: lessons from the small nuclear and small nucleolar rnas. *Nature reviews Molecular cell biology*, *8*(3), 209–220.

Mathews, D. H., Sabina, J., Zuker, M., & Turner, D. H. (1999). Expanded sequence dependence of thermodynamic parameters improves prediction of rna secondary structure. *Journal of molecular biology*, *288*(5), 911–940.

McCool, M. D., Robison, A. D., & Reinders, J. (2012). *Structured parallel programming: patterns for efficient computation*. Elsevier.

Mimouni, N. K., Lyngsø, R. B., Griffiths-Jones, S., & Hein, J. (2009). An analysis of structural influences on selection in rna genes. *Molecular biology and evolution*, *26*(1), 209–216.

Nakaya, A., Yamamoto, K., & Yonezawa, A. (1995). Rna secondary structure prediction using highly parallel computers. *Computer applications in the biosciences: CABIOS*, *11*(6), 685–692.

Nawrocki, E. P., & Eddy, S. R. (2013). Computational identification of functional rna homologs in metagenomic data. *RNA biology*, *10*(7), 1170–1179.

Nussinov, R., & Jacobson, A. B. (1980). Fast algorithm for predicting the secondary structure of single-stranded rna. *Proceedings of the National Academy of Sciences*, *77*(11), 6309–6313.

Owens, J., & Luebke, D. (2009). Intro to parallel programming. an open, online course from udacity. `http://www.nvidia.com/object/cuda_home_new.html`. Revisado el 24/06/2016.

Papadopoulos, D., Schneider, D., Meier-Eiss, J., Arber, W., Lenski, R. E., & Blot, M. (1999). Genomic evolution during a 10,000-generation experiment with bacteria. *Proceedings of the National Academy of Sciences*, *96*(7), 3807–3812.

Perez, N., Treviño, J., Liu, Z., Ho, S. C. M., Babitzke, P., & Sumby, P. (2009). A genome-wide analysis of small regulatory rnas in the human pathogen group a streptococcus. *PloS one*, *4*(11), e7668.

Perkins, D., Jeffries, C., & Sullivan, P. (2005). Expanding the central dogma: the regulatory role of nonprotein coding genes and implications for the genetic liability to schizophrenia. *Molecular psychiatry*, *10*(1), 69–78.

Pichon, C., & Felden, B. (2008). Small rna gene identification and mrna target predictions in bacteria. *Bioinformatics*, *24*(24), 2807–2813.

Repoila, F., & Darfeuille, F. (2009). Small regulatory non-coding rnas in bacteria: physiology and mechanistic aspects. *Biology of the Cell*, *101*(2), 117–131.

Rivas, E., & Eddy, S. R. (2001). Noncoding rna gene detection using comparative sequence analysis. *BMC bioinformatics*, *2*(1), 1.

Rogers, A. E. (2012). Predicción computacional de genes de arn no codificante pequeño en genomas bacterianos. (pp. 1–42).

Rojas, Ó., Albornoz, W., & Marín, M. (2017). Adaptive parallelism for the processing of bacterial genomes in the prediction of ncrnas. VI workshop CeBiB Bioinformatics Metabolic Engineering and Genomics in Biotechnological Applications, Santiago, Chile.

Rojas, O., Gil-Costa, V., & Marin, M. (2016). Running time prediction for web search queries. *Parallel Processing and Applied Mathematics: 11th International Conference,PPAM 2015, Krakow, Poland, September 6-9, 2015.*, (pp. 210–220).

Sayers, E. W. y. B. (2009). Database resources of the national center for biotechnology information. *Nucleic acids research*, *9*(1), 5–15.

Shah, H. (2012). Algorithms for predicting secondary structures of human viruses. (pp. 3–13).

Swenson, M. S., Anderson, J., Ash, A., Gaurav, P., Sükösd, Z., Bader, D. A., Harvey, S. C., & Heitsch, C. E. (2012). Gtfold: enabling parallel rna secondary structure prediction on multi-core desktops. *BMC research notes*, *5*(1), 341.

Szymanski, M., Erdmann, V. A., & Barciszewski, J. (2007). Noncoding rnas database (ncrnadb). *Nucleic acids research*, *35*(suppl 1), D162–D164.

Tabei, Y., & Asai, K. (2009). A local multiple alignment method for detection of non-coding rna sequences. *Bioinformatics*, *25*(12), 1498–1505.

Taufer, M., Solorio, T., Licon, A., Mireles, D., & Leung, M.-Y. (2008). On the effectiveness of rebuilding rna secondary structures from sequence chunks. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, (pp. 1–8). IEEE.

Toledo-Arana, A., Repoila, F., & Cossart, P. (2007). Small noncoding rnas controlling pathogenesis. *Current opinion in microbiology*, *10*(2), 182–188.

Tran, T. T. T. (2009). Genomic data mining for the computational prediction of small non-coding rna genes. (pp. 1–5).

Trotta, E. (2014). On the normalization of the minimum free energy of rnas by sequence length. *PloS one*, *9*(11), e113380.

Uzilov, A. V., Keegan, J. M., & Mathews, D. H. (2006). Detection of non-coding rnas on the basis of predicted secondary structure formation free energy change. *BMC bioinformatics*, *7*(1), 173.

Varani, G., & McClain, W. H. (2000). The g· u wobble base pair. *EMBO reports*, *1*(1), 18–23.

Vogel, J., Bartels, V., Tang, T. H., Churakov, G., Slagter-Jäger, J. G., Hüttenhofer, A., & Wagner, E. G. H. (2003). Rnomics in escherichia coli detects new srna species and indicates parallel transcriptional output in bacteria. *Nucleic acids research*, *31*(22), 6435–6443.

Vogel, J., & Papenfort, K. (2006). Small non-coding rnas and the bacterial outer membrane. *Current opinion in microbiology*, *9*(6), 605–611.

Vogel, J., & Sharma, C. M. (2005). How to find small non-coding rnas in bacteria. *Biological chemistry*, *386*(12), 1219–1238.

Voß, B., Georg, J., Schön, V., Ude, S., & Hess, W. R. (2009). Biocomputational prediction of non-coding rnas in model cyanobacteria. *BMC genomics*, *10*(1), 1.

Washietl, S., Hofacker, I. L., & Stadler, P. F. (2005). Fast and reliable prediction of noncoding rnas. *Proceedings of the National Academy of Sciences of the United States of America*, *102*(7), 2454–2459.

Wassarman, K. M., Repoila, F., Rosenow, C., Storz, G., & Gottesman, S. (2001). Identification of novel small rnas using comparative genomics and microarrays. *Genes & development*, *15*(13), 1637–1651.

Waters, L. S., & Storz, G. (2009). Regulatory rnas in bacteria. *Cell*, *136*(4), 615–628.

Xu, X., Ji, Y., & Stormo, G. D. (2009). Discovering cis-regulatory rnas in shewanella genomes by support vector machines. *PLoS computational biology*, *5*(4), e1000338.

Zuker, M. (2003). Mfold web server for nucleic acid folding and hybridization prediction. *Nucleic acids research*, *31*(13), 3406–3415.

Zuker, M., & Stiegler, P. (1981). Optimal computer folding of large rna sequences using thermodynamics and auxiliary information. *Nucleic acids research*, *9*(1), 133–148.

Zuker, M., et al. (1989). On finding all suboptimal foldings of an rna molecule. *Science*, *244*(4900), 48–52.

ANEXO A. CÁLCULO DE EMP

A.1. MÉTODOS DE CÁLCULO DE LAS EMP

La estructura tridimensional de una molécula de ARN está determinada por la información contenida dentro de la secuencia de bases nucleotídicas. Analizar la estructura secundaria de la secuencia de bases nucleotídicas puede proporcionar una visión del primer borrador de la molécula (Zuker & Stiegler, 1981).

El análisis de la estructura secundaria trae consigo valores representativos que ayudan a la obtención de las emp y Z-score, lo cual se especifica en la Sección 2.4.2. Estos valores representativos, son obtenidos mediante distintos tipos de algoritmos, los cuales tienen como principal función la predicción de la estructura secundaria del ARN (Eddy, 2004). En investigaciones relacionadas, los que frecuentemente se utilizan son el algorito de Nussinov, el algoritmo de Minimización sencilla y el algoritmo de Zuker (Zuker & Stiegler, 1981; Zuker et al., 1989).

A.1.1. Algoritmo de Nussinov

Este es un algoritmo de programación dinámica que calcula el número máximo de pares de bases en una molécula de ARN doblada. El algoritmo de Nussinov es simple y actúa como una base para todos los otros algoritmos avanzados de predicción de ARN. El algoritmo de Nussinov está diseñado para evaluar la contribución de pares de bases individuales a la estructura secundaria de una cadena polinucleotídica (Nussinov & Jacobson, 1980). Considere una secuencia de n nucleótidos $S_1 \dots S_n$. Para identificar la estructura con el número máximo de pares de bases, el sistema de puntuación agrega uno por par de bases a la puntuación, añade cero para cualquier otra cosa (Eddy, 2004). La puntuación óptima $S_{(i,j)}$, de una subsecuencia del ARN de la posición i a la posición j, puede definirse recursivamente en términos de puntuaciones óptimas de subsecuencias menores (Shah, 2012). Sólo hay cuatro formas posibles de construir una estructura de pares de bases anidados, esto se puede apreciar en la Figura A.1 (Eddy, 2004).

1. i y j pares de bases, añadidos a una estructura para $i+1, j-1$.
2. i está desemparejado, añadido a una estructura para $i+1, j$.
3. j está desemparejado, añadido a una estructura para $i, j-1$.

4. iyj par de bases pero no entre sí; la estructura para $i..j$ agrega sub-estructuras para dos sub-secuencias, $i..k$ y $k+1..j$ (una bifurcación).

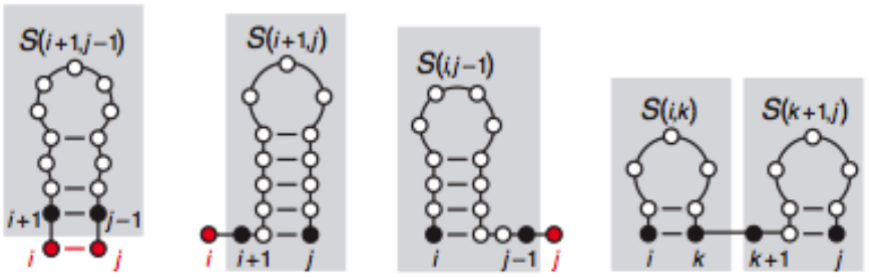


Figura A.1: Cuatro posibilidades de la definición recursiva.
Fuente: Modificada de (Eddy, 2004), 2017.

El algoritmo utiliza la siguiente recursión matemática:

$$S(i,j) = max \begin{cases} S(i+1,j-1)+1, \\ S(i+1,j), \\ S(i,j-1), \\ max_{i<k<j}[S(i,k)+S(k+1,j)] \end{cases} \tag{A.1}$$

Almacenar la matriz $S(i,j)$ requiere memoria en el orden de N^2 donde N es el número de nucleótidos en la secuencia a plegar. Sin embargo, el bucle más interno al tener que encontrar los puntos de bifurcación potencial más óptimos hace que la complejidad temporal pase a ser de orden N^3 (Eddy, 2004). La maximización de pares de bases es un esquema de puntuación inferior para la predicción de estructuras secundarias de ARN. El uso de la energía global mínima de una estructura es un mejor sistema de puntuación y el algoritmo de Zuker (Sección A.1.3) utiliza un enfoque de programación dinámica que incorpora las energías de las subestructuras (Zuker & Stiegler, 1981).

A.1.2. Algoritmo de Minimización sencilla de energía

Este algoritmo busca obtener mejores predicciones minimizando la siguiente función de energía para una secuencia de ARN x y el conjunto P de pares de bases:

$$E(x, P) = \sum_{(i,j)\in P} e(x_i, x_j), \tag{A.2}$$

donde $e(x_i, x_j)$ es la cantidad de emp asociada al par de bases (x_i, x_j). Los valores razonables para e a 37°C son -3, -2 y -1 kcal/mol para los pares de bases C - G, A - U y G - U, respectivamente (Elliott & Ladomery, 2017). Usando lo descrito anteriormente, se generaliza el algoritmo de Nussinov considerando la energía libre de un par de bases, debido a que se utiliza la función $e(x_i, x_j)$ en lugar de la función $\delta(i, j)$ simple. Dado que la energía libre de un par de bases es negativa, se busca las estructuras con energía mínima global. Así, se genera la siguiente formula recursiva:

$$E(i,j) = min \begin{cases} E(i+1, j), \\ E(i, j-1), \\ E(i+1, j-1) + e(x_i, x_j), \\ min_{i<k<j}[E(i,k) + E(k+1, j)] \end{cases} \tag{A.3}$$

Desafortunadamente, este enfoque no produce buenas predicciones de estructura y por lo tanto un cálculo exacto, esto debido a que no tiene en cuenta que las pilas helicoidales de pares de bases tienen un efecto estabilizador, mientras que los bucles tienen un efecto desestabilizador sobre la estructura. Debido a esto, se requiere un enfoque más sofisticado para el cálculo de emp. (Elliott & Ladomery, 2017)

A.1.3. Algoritmo de Zuker

El algoritmo de Zuker utiliza un enfoque de programación dinámica que se basa en los mismos cuatro pasos de reducción que el algoritmo de Nussinov. Una diferencia importante es que el algoritmo de Zuker se centra en bucles en lugar de pares de bases. Esto proporciona una mejor adaptación a los datos observados experimentalmente. (Zuker & Stiegler, 1981; Zuker et al., 1989).

Dada una secuencia de nucleótidos S de largo N. Los nucleótidos de una molécula de ARN son referidos como vértices en la representación gráfica. Los N-1 arcos del semicírculo entre las bases se llaman bordes exteriores. Los bordes exteriores representan los enlaces fosfodiéster entre nucleótidos consecutivos. Los acordes en el semicírculo representan el emparejamiento de la base entre dos nucleótidos. Estos acordes se llaman bordes interiores. Los bordes y los vértices se combinan para formar la gráfica de una secuencia de ARN. No se permite que los acordes se corten o se toquen entre sí, ya que descarta todas las subestructuras anidadas. La energía libre de la estructura está asociada con las regiones entre enlaces. Para la representación gráfica, la energía depende de las caras existentes. (Zuker & Stiegler, 1981)

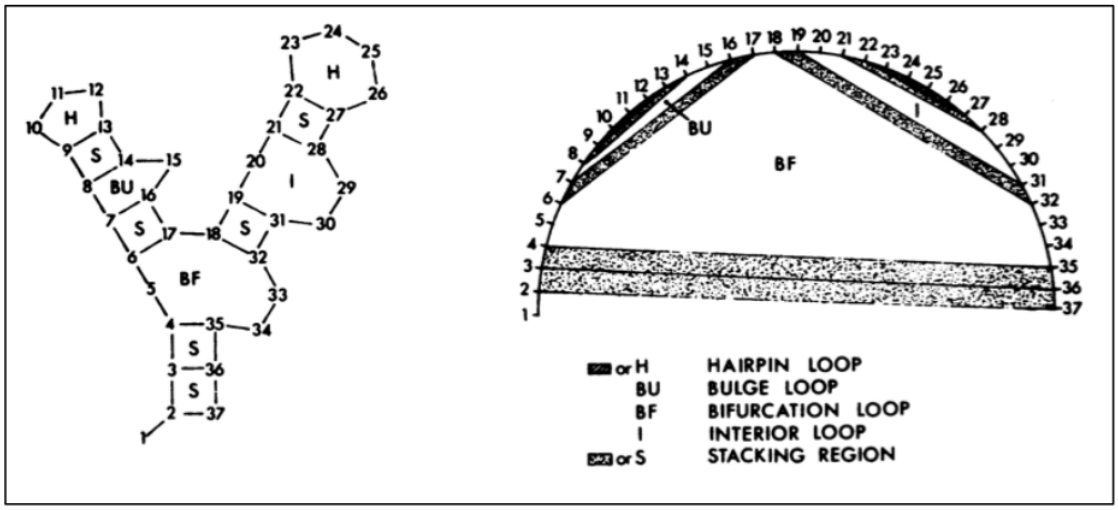


Figura A.2: Representación de una estructura secundaria de ARN.
Fuente: Modificada de (Zuker & Stiegler, 1981), 2017.

Existen 5 tipos de bucles diferentes:

1. Loop horquilla: Una cara con un solo borde interior.
   Los tres tipos siguientes tienen dos bordes interiores.
2. Región de apilado: Los bordes interiores están separados por bordes exteriores únicos en cada lado.
3. Loop Bulto: Los bordes interiores están separados por un solo borde exterior en un lado, pero por más de un borde exterior en el otro lado.
4. Loop Interior: Cara con exactamente dos bordes interiores sin restricciones en los bordes exteriores de los lados.
5. Loop de bifurcación: Cara con tres o más bordes interiores.

La energía de una estructura es la suma de las energías de sus caras. El algoritmo de Zuker encuentra una estructura secundaria que tiene la energía mínima usando principios de programación dinámica. (Zuker & Stiegler, 1981; Zuker et al., 1989)

Sea $x = (x_1, x_2, ..., x_n)$ una cadena sobre el alfabeto $\sum = A, G, C, U$. Para $i < j$, sean:

- $W_{(i,j)}$: Denota la energía mínima de plegamiento de todos los plegamientos no vacíos de la subsecuencia $x_i, ..., x_j$.
- $V_{(i,j)}$: Denota la energía mínima de plegamiento de todos los plegamientos no vacíos de la subsecuencia $x_i, ..., x_j$, que contiene el par de bases (i, j).

El siguiente hecho es crucial:

$$W_{(i,j)} \leq V_{(i,j)} \text{ para todo } i, j.$$

Las dos matrices V y W se inicializan de la siguiente manera:

$$W_{(i,j)} = V_{(i,j)} = \infty \text{ para todo } i, j, \text{ con } j - 4 < i < j.$$

Con esto se cumple el hecho de que dos bases pareadas estén al menos 3 posiciones alejadas unas de otras.

A.1.3.1. Energías dependientes por Bucle

Las diferentes funciones de energía para los diferentes tipos de bucles son:

- Sea $eh(i,j)$ la energía de la horquilla cerrada por el par de bases (i,j),
- Sea $es(i,j)$ la energía del par apilado (i,j) y $(i+1,j-1)$,
- Sea $ebi(i,j,i',j')$ la energía del bulto o lazo interior cerrada por (i,j), con (i',j') accesible desde (i,j) y
- a denota un término de energía constante asociado con un multi-lazo.

Valores de energía libre pronosticados (kcal/mol a 37°C) para el apilamiento de pares de bases:

Tabla A.1: Valores pronosticados para pares de bases (Zuker & Stiegler, 1981; Zuker et al., 1989).

	A/U	**C/G**	**G/C**	**U/A**	**G/U**	**U/G**
A/U	-0.9	-1.8	-2.3	-1.1	-1.1	-0.8
C/G	-1.7	-2.9	-3.4	-2.3	-2.1	-1.4
G/C	-2.1	-2.0	-2.9	-1.8	-1.9	-1.2
U/A	-0.9	-1.7	-2.1	-0.9	-1.0	-0.5
G/U	-0.5	-1.2	-1.4	-0.8	-0.4	-0.2
U/G	-1.0	-1.9	-2.1	-1.1	-1.5	-0.4

Valores predictivos de energía libre (kcal/mol a 37°C) para las características de las estructuras secundarias de ARN predicho, por tamaño de bucle:

Tabla A.2: Valores pronosticados de emp por tamaño de Loop (Zuker & Stiegler, 1981; Zuker et al., 1989).

Size	**Internal Loop**	**Bulge**	**Hairpin**
1	.	3.9	.
2	4.1	3.1	.
3	5.1	3.5	4.1
4	4.9	4.2	4.9
5	5.3	4.8	4.4
10	6.3	5.5	5.3
15	6.7	6.0	5.8
20	7.0	6.3	6.1
25	7.2	6.5	6.3
30	7.4	6.7	6.5

A.1.3.2. La recursión principal

La recursión principal completa, para todo i, j con $1 \leq i < j \leq L$:

$$
\begin{aligned}
W(i,j) &= min \begin{cases} W(i+1,j), \\ W(i,j-1), \\ V(i,j), \\ min_{i<k<j}[W(i,k) + W(k+1,j)] \end{cases} \\
V(i,j) &= min \begin{cases} eh(i,j), \\ es(i,j) + V(i+1,j-1) \\ VBI(i,j), \\ VM(i,j), \end{cases} \\
VBI(i,j) &= min_{i<i'<j'<j} \Big\{ \ ebi(i,j,j',i') + V(i',j'), \text{ y } i' - i < +j - j' > 2 \\
VM(i,j) &= min_{i<k<j-1} \Big\{ \ W(i+1,k) + W(k+1,j-1) + a.
\end{aligned}
\tag{A.4}
$$

A continuación se expone cada uno de los cuatro casos en detalle. El primer caso considera las cuatro posibilidades en las cuales (a) i está desemparejado, (b) j está desemparejado, (c) i y j están emparejados entre sí y (d) i y j están posiblemente apareados, pero no entre sí. En el caso (c) se hace referencia a la matriz auxiliar V.

$$
W(i,j) = min \begin{cases} W(i+1,j) & \text{(a)} \\ W(i,j-1) & \text{(b)} \\ V(i,j) & \text{(c)} \\ min_{i<k<j}[W(i,k) + W(k+1,j)] & \text{(d)} \end{cases}
\tag{A.5}
$$

El segundo caso considera las diferentes situaciones que surgen cuando las bases i y j están emparejadas, cerrando (a) un bucle de horquilla, (b) un par apilado, (c) un bulto o bucle interior o (d) un bucle múltiple. Los dos últimos casos son más complicados y se obtienen de las Ecuaciones (A.7) y (A.8).

$$V(i,j) = min \begin{cases} eh(i,j) & \text{(a)} \\ es(i,j) + V(i+1,j-1) & \text{(b)} \\ VBI(i,j) & \text{(c)} \\ VM(i,j) & \text{(d)} \end{cases} \tag{A.6}$$

El tercer caso toma en cuenta todas las formas posibles de definir un bulto o bucle interior que involucra un par de bases (i', j') y está cerrado por (i, j). En cada situación, tenemos una contribución de la protuberancia o bucle interior y una contribución de la estructura que está en el lado opuesto de (i', j').

$$VBI(i,j) = min_{i<i'<j'<j} \Big\{ \ ebi(i,j,j',i') + V(i',j'), \text{ y } i'-i<+j-j'>2 \tag{A.7}$$

El cuarto caso considera las diferentes maneras de obtener un multi-loop de dos estructuras más pequeñas y añade una contribución constante de a para cerrar el bucle.

$$VM(i,j) = min_{i<k<j-1} \Big\{ \ W(i+1,k) + W(k+1,j-1) + a. \tag{A.8}$$

ANEXO B. COMPUTACIÓN PARALELA

La computación paralela es un tipo de procesamiento de la información que permite la ejecución de varios procesos concurrentemente, operando sobre el principio de que problemas grandes a menudo se pueden dividir en unos más pequeños, que luego son resueltos simultáneamente (en paralelo). Hay varias formas diferentes de computación paralela: paralelismo a nivel de bit, paralelismo a nivel de instrucción, paralelismo de datos y paralelismo de tareas. El paralelismo se ha empleado durante muchos años, sobre todo en la computación de altas prestaciones, pero el interés en ella ha decrecido últimamente debido a las limitaciones físicas que impiden el aumento de la frecuencia (Golub & Ortega, 2014).

Los programas paralelos son más difíciles de escribir que los secuenciales, esto debido a que la concurrencia introduce nuevos tipos de errores de software. La comunicación y sincronización entre diferentes sub-tareas son algunos de los mayores obstáculos para obtener un buen rendimiento del programa paralelo.

Los enfoques paralelos resultan una necesidad dada la constante afirmación de que la velocidad máxima en procesamiento secuencial se alcanzará prontamente, excepto que aparezcan avances en otras áreas que definan nuevos mecanismos de procesamiento, los cuales se obtienen de nuevos descubrimientos en áreas tales como la computación cuántica (Golub & Ortega, 2014).

Es razonable pensar que la naturaleza humana vislumbre nuevas aplicaciones que excedan las capacidades de los sistemas computacionales. Por ejemplo, las actuales aplicaciones de realidad virtual requieren considerables velocidades de cálculo para obtener resultados interesantes en tiempo real. Esto refleja la necesidad permanente de realizar estudios sobre nuevos esquemas de optimización en el ámbito computacional (Golub & Ortega, 2014).

La computación en paralelo, por el contrario, utiliza simultáneamente múltiples elementos de procesamiento para resolver un problema. Esto se logra mediante la división del problema en partes independientes de modo que cada elemento de procesamiento pueda ejecutar su parte del algoritmo de manera simultánea con los otros. Los elementos de procesamiento son diversos e incluyen recursos tales como una computadora con múltiples procesadores, varios ordenadores en red, hardware especializado, o cualquier combinación de los anteriores (Golub & Ortega, 2014).

B.1. MÉTODOS ALGORÍTMICOS

Mientras que las computadoras paralelas se hacen más grandes y más rápidas, se hace factible resolver problemas que antes tomaban mucho tiempo de ejecución. La computación paralela se utiliza en una amplia gama de campos, desde la bioinformática (plegamiento de proteínas y análisis de secuencia) hasta la economía (matemática financiera). También se debe considerar que existen problemas encontrados comúnmente en las aplicaciones de computación en paralelo, tales como la álgebra lineal dispersa, la lógica combinacional y el recorrido de grafos, para los cuales aún es necesaria más investigación (Asanovic et al., 2006).

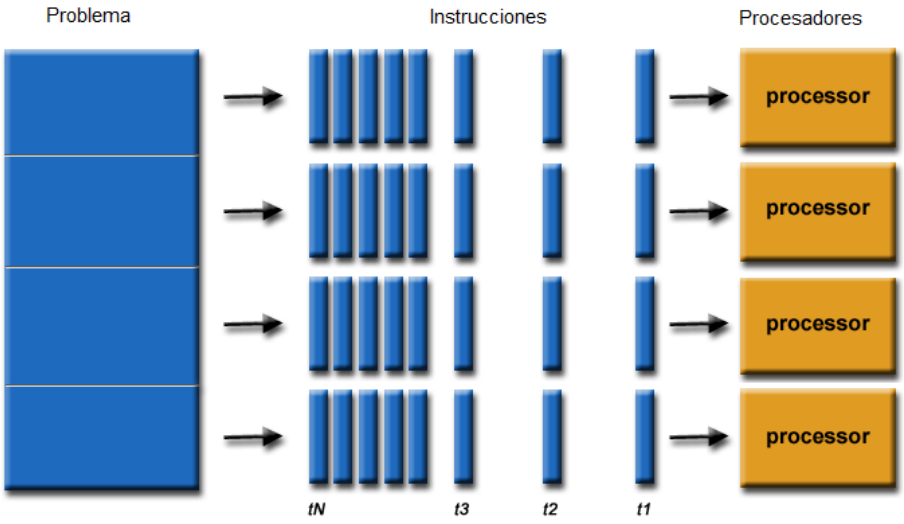


Figura B.1: Ejemplo gráfico del análisis paralelo de una instrucción.
Fuente: Modificada de (Barney, 2009a), 2017.

En la actualidad, la cantidad de métodos para desarrollar aplicaciones de computación paralela es muy extensa, por lo que se hace importante describir algunos de ellos, ésto permite generar una base de información para identificar cuál método sería el más adecuado para enfrentar la problemática. Entre los más importantes se encuentran:

B.1.0.3. Pthread

Los POSIX threads, comúnmente llamados Pthreads, son un modelo de ejecución en paralelo que existe independientemente del lenguaje escogido. Permite a un programa controlar múltiples flujos de trabajo diferentes en el tiempo. Cada flujo de trabajo es llamado thread, y para crear y controlar esos flujos se utiliza la libreria de POSIX threads (Barney, 2010).

Las aplicaciones en hilos ofrecen posibles mejoras de rendimiento y ventajas prácticas sobre las aplicaciones sin ellos en varias otras maneras, por ejemplo, el trabajo de la CPU con la superposición de entrada/salida de datos, esto quiere decir que un programa puede tener secciones donde se está realizando una operación larga de entrada/salida, mientras que un hilo está esperando una llamada al sistema de entrada/salida para completar, el trabajo intenso de la CPU puede ser realizado por otros hilos, lo que implica la división del trabajo (Barney, 2010).

A continuación, se presenta una gráfica representativa del funcionamiento de la librería.

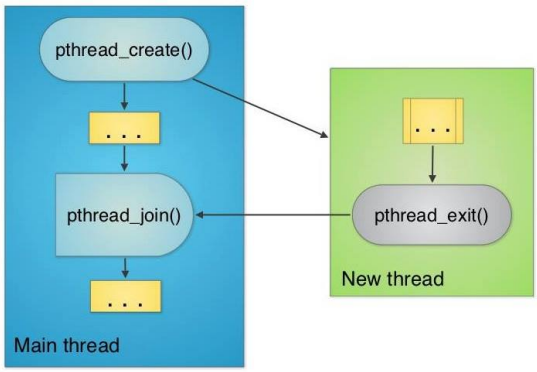


Figura B.2: Esquema general del modelo Pthread.
Fuente: Modificada de (Barney, 2010), 2017.

B.1.0.4. CUDA

Es una plataforma de computación paralela y modelo de programación inventado por Nvidia. Permite incrementar exponencialmente la velocidad de cómputo utilizando el poder de las unidades de procesamiento gráfico (GPU).

Con CUDA se pueden enviar códigos en C, C++ y Fortran directamente a la GPU sin necesidad de utilizar lenguaje ensamblador. Al utilizar lenguajes de alto nivel junto con CUDA, las aplicaciones ejecutan su parte secuencial en la CPU (la cual está optimizada para el trabajo en un thread), mientras el procesamiento paralelo es acelerado mediante la GPU (Owens & Luebke, 2009).

CUDA es ideal para trabajar con problemas de computación paralela que son muy complejos donde se requiere una mínima comunicación en la ejecución. CUDA es divide los problemas en grillas de bloques, donde cada uno de estos bloques contiene múltiples threads, cada uno de estos bloques se ejecutan por secciones en el tiempo (Cook, 2012). A continuación, en la Figura B.3 se muestra gráficamente el funcionamiento de CUDA.

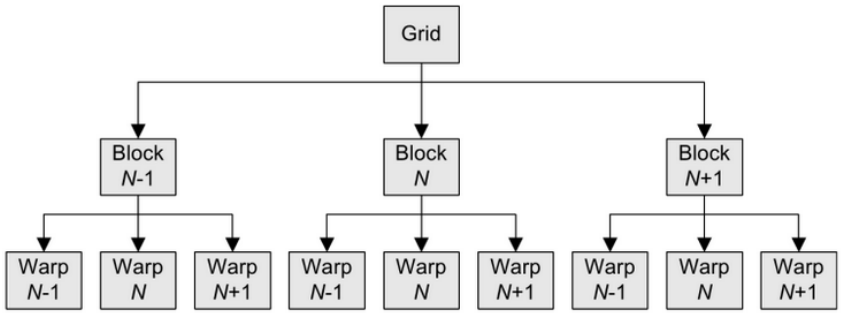


Figura B.3: Ejemplo de bloques de CUDA.
Fuente: Modificada de (Cook, 2012), 2017.

B.1.0.5. Modelo Fork-Join

En la computación paralela, el modelo fork-join es una manera de crear y ejecutar programas paralelos, de manera que las ramas de ejecución del programa son paralelas, pero en ciertos puntos designados el programa estas ramas se unen y reanudan la ejecución secuencial (McCool et al., 2012).

Existe una serie de APIs y librerías que implementan el modelo Fork-join, una de las más conocidas es OpenMP, esta API que provee un modelo portable y escalable para los desarrolladores de aplicaciones de memoria compartida paralela. Esta API soporta C/C++ y Fortran en una gran cantidad de arquitecturas. Cabe destacar que OpenMP es una abreviación de Open

Multi Processing (Barney, 2009b).

A continuación, en la Figura B.4 se muestra un esquema de un proceso en el modelo fork-join.

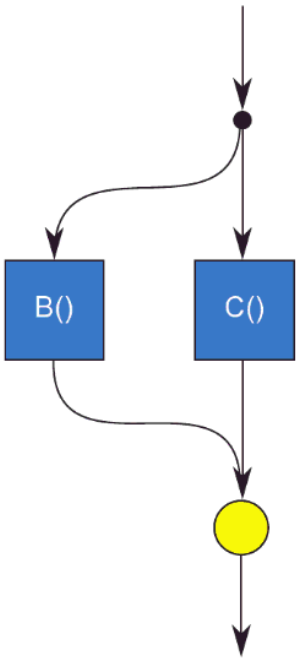


Figura B.4: Ejemplo del modelo fork-join.
Fuente: Modificada de (McCool et al., 2012), 2017.

ANEXO C. Herramientas de desarrollo

C.1. LENGUAJE DE PROGRAMACIÓN Y LIBRERÍAS DE DESARROLLO

En este Anexo se presenta las el lenguaje de programación y librerías utilizadas para el desarrollo del software.

- **Lenguajes:**
  - **C:** Desarrollo de algoritmo de cálculo de emp.
  - **C++:** Desarrollo de algoritmo paralelo de solución.
- **Librerías:**
  - **errno.h** (Definición de macros que presentan un informe de error a través de códigos de error.)
  - **iostream** (Operaciones de entrada/salida.)
  - **iomanip** (Manipulación para formateo de salidas, enteros y punto flotante.)
  - **vector** (Manipulación de arreglos unidimensionales.)
  - **math.h** (Operaciones Matemáticas básicas.)
  - **sys/time.h** (Formateo para cálculo del tiempo.)
  - **unistd.h** (Provee acceso a la API del sistema operativo POSIX.)
  - **string.h** (Define un tipo de variable, una macro y variadas funciones para el manejo de arreglos de caracteres.)
  - **stdlib.h** (Prototipos de funciones de C para gestión de memoria dinámica, control de procesos y otras. (ejemplo calloc, malloc, free).)
  - **stdio.h** (Definiciones de las macros, constantes y declaraciones de funciones para hacer operaciones, estándar, de entrada y salida, así como la definición de tipos necesarias para dichas operaciones.)
  - **pthread.h** (Definiciones de la librería POSIX threads, con ella se pueden definir y trabajar con threads.)
  - **string.h** (Funciones para trabajar con strings y utilizar buffers para el acceso a cadenas de caracteres.)

La librería que resalta en importancia y en la cual se apoya esta memoria es la librería **pthread.h**. El conjunto de herramientas disponibles en **pthread.h** permiten desarrollar funciones que computen Z-score de forma paralela. El trabajo que realizarán estas funciones se presenta en la Sección 3.1.2.