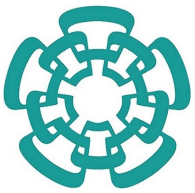

Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Unidad Zacatenco

Departamento de Computación

Aceleración de un Algoritmo Evolutivo en GPUs para la Inferencia de Modelos de Redes Reguladoras de Genes.

Tesis que presenta

Luis Enrique Ramírez Chávez

para obtener el Grado de

Maestro en Ciencias

en Computación

Directores de la Tesis

Dr. Carlos Artemio Coello Coello

Dr. Eduardo Arturo Rodríguez Tello

México, D.F. **Febrero 2012**

Aceleración de un Algoritmo Evolutivo en GPUs para la Inferencia de Modelos de Redes Reguladoras de Genes.

Luis Enrique Ramirez Chavez

2 de febrero de 2012

Abstract

Bioinformatics studies the application of computer science techniques to problems arising in biological sciences through the use of algorithms, databases, concepts and several other information technologies. Within computer science, we have evolutionary computation, which uses metaheuristics based on the principle of the survival of the fittest from Charles Darwin's evolutionary theory in order to solve optimization and classification problems.

This thesis deals with the solution of a problem from bioinformatics known as the inference of models of gene regulatory networks, using evolutionary computation and parallel techniques. The evolutionary algorithm adopted for this work is differential evolution. Such an algorithm was chosen due to its well-known efficiency (as reported in the specialized literature) when solving optimization problems in which all the decision variables are real numbers.

Gene regulatory networks are sets of genes that interact with each other, affecting the expression levels of the genes involved in the network. Inferring the behavior of gene regulatory networks is a computationally expensive process. Thus, the use of high performance computing is an attractive choice within this domain. Recently, and thanks to the technological advances in the development of graphical processing

units (GPUs), they have been adopted as an economical and very efficient alternative for high performance computing.

In this thesis, we present what seems to be the first GPU-based implementation (using CUDA) of differential evolution for solving the problem of inferring models of gene regulatory networks. Our implementation adopts systems of differential equations (called S systems) for modelling the dynamism of the expression levels that occur in the regulatory networks of our interest.

The results presented in this thesis show that the use of parallel computing through the use of GPUs produces an important reduction in the computational time required to solve this expensive optimization problem. The results shown here indicate reductions of up to 140 times in the computational time required for one execution of our optimization algorith, with respect to its sequential version.

Although the networks solved in this thesis are of modest size, the possible extrapolation of our algorithm to larger instances could bring important benefits, mainly because of the many applications that gene regulatory networks have in medicine, environmental engineering, biological design and the pharmaceutical industry.

Resumen

La bioinformática estudia la aplicación de las ciencias computacionales a problemas existentes en las ciencias biológicas mediante el uso de algoritmos, bases de datos, conceptos y diversas tecnologías de la información [1]. Dentro de las ciencias computacionales, encontramos el área de la computación evolutiva, la cual usa metaheurísticas basadas en el principio de la supervivencia del más apto de la teoría de Charles Darwin, para resolver problemas de optimización y clasificación. Esta tesis aborda la solución de un problema de bioinformática conocido como la inferencia de modelos de redes reguladoras de genes, usando técnicas de computación evolutiva y paralelismo. El algoritmo evolutivo adoptado para este trabajo es la evolución diferencial. Dicho algoritmo se eligió debido a su probada eficiencia (reportada en la literatura especializada) para resolver problemas de optimización en los que todas las variables de decisión son números reales.

Las redes reguladoras de genes son conjuntos de genes que interactúan entre sí, afectando los niveles de expresión de los genes involucrados en la red. Inferir el comportamiento de las redes reguladoras de genes es un proceso computacionalmente costoso. Por ello, el uso de cómputo de alto rendimiento resulta una alternativa atractiva en este dominio. Recientemente, y gracias al desarrollo tecnológico de las unidades de

procesamiento gráfico (GPUs, por sus siglas en inglés), éstas han sido utilizadas como una alternativa económica y muy eficiente para realizar cómputo de alto rendimiento.

En esta tesis presento la primera implementación de evolución diferencial basada en GPUs (utilizando CUDA) para resolver el problema de la inferencia de modelos de redes reguladoras de genes. En nuestra implementación, se adoptan sistemas de ecuaciones diferenciales ordinarias (llamados sistemas S) para modelar el dinamismo de los niveles de expresión que ocurren en las redes reguladoras de genes de nuestro interés.

Los resultados presentados en esta tesis demuestran que el uso de cómputo paralelo mediante GPUs produce una importante reducción del tiempo de cómputo necesario para resolver este costoso problema de optimización. Los resultados que aquí se presentan, indican reducciones de hasta 140 veces en el tiempo de cómputo requerido para una ejecución del algoritmo propuesto, con respecto a su versión secuencial.

Aunque las redes resueltas en esta tesis tienen tamaños modestos, la posible extrapolación de nuestro algoritmo a instancias más grandes puede pueden traer beneficios importantes, debido a las diversas aplicaciones que tienen las redes reguladoras de genes en medicina, ingeniería ambiental, diseño biológico y la industria farmacéutica.

Agradecimientos

Agradezco a mi papá y a mi mamá por el infinito cariño que tienen hacia a mi, ellos siempre serán mi máximo orgullo y mi principal fortaleza.

A mis hermanos Juan Ramón y Lupita a quienes quiero mucho y que siempre me han apoyado.

Mis más profundo agradecimiento y admiración al Dr. Carlos Artemio Coello Coello por haberme aceptado como su tesista, su gran paciencia, sus enseñanzas, observaciones y consejos que hicieron posible esta tesis, además de ser una inspiración.

Mi especial reconocimiento y admiración al Dr. Eduardo Arturo Rodríguez Tello por ser un ejemplo a seguir y agradecerle el haberme aceptado como su tesista, y, que a pesar de la distancia, estuvo al pendiente de mi trabajo y siempre me brindo su apoyo y consejos que hicieron posible el desarrollo de esta tesis.

A mis revisores de tesis el Dr. Gregorio Toscano Pulido y al Dr. Luis Gerardo de la Fraga por que gracias a la dedicación y tiempo que le dieron a la lectura de esta

tesis y a sus comentarios resultó un mejor trabajo.
A los doctores del departamento de computación especialmente a los que fueron mis profesores, pero también a los que no lo fueron, ya que todos me dejaron alguna enseñanza.

A mis tías Chita y Chave por estar al pendiente de mis estudios en esta etapa y a mis demás primos y tíos que me ofrecieron su apoyo y soporte.
A todas mis amigos que he conocido a lo largo de mi vida y que durante esta etapa me brindaron su apoyo especialmente a Karina, Geovani, Eli, Emma, Any, Chucho, Cesar O., Miguel, Vane, Benjamin, Abel, Mario, Fabiola, Abel, Ed, Pao, Cesar Reyna, Ulises y Cristobal.

A mis amigos de la maestría Alex, Temoc, Cuau, Genaro, Eric, Anita, Herón, Laurita, Armando, Fer, Alberto, Edgar y demás compañeros del CINVESTAV que hicieron de esta una de las mejores etapas de mi vida. También quiero agradecer al grupo EVOCINV por los consejos para desarrollar mi trabajo.

Agradezco especialmente a la Sra. Sofia Reza que siempre estuvo al pendiente de mi y me brindo su ayuda cuando la necesite, siempre estando al pendiente de mis estudios, también quiero agradecer a Feli y demás personal administrativo que facilitaron mi estancia en el departamento de computación.

Agradezco al Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, por haberse convertido en mi segundo hogar y haberme brindado una educación de excelencia.

Agradezco al Consejo Nacional de Ciencia y Tecnología CONACYT por el apoyo financiero recibido, mediante el cual pude realizar mis estudios de maestría.

El trabajo presentado en esta tesis, se derivó del proyecto CONACyT titulado "Escalabilidad y nuevos esquemas híbridos en optimización evolutiva multiobjetivo" (Ref. 103570), cuyo responsable es el Dr. Carlos Artemio Coello Coello.

Durante la realización de esta tesis realicé una estancia corta de investigación en la unidad Tamaulipas del Cinvestav, la cual fue financiada por el proyecto CONACyT titulado "Algoritmos para la Canonización de Covering Arrays" (Ref. 99276), cuyo responsable es el Dr. Eduardo Arturo Rodríguez Tello.

Índice general

1. Introducción **1**

1.1. Motivación 1

1.2. Planteamiento del problema 2

1.3. Objetivo 2

1.4. Contribuciones 3

1.5. Organización de la tesis 3

2. Bioinformática **5**

2.1. Áreas de investigación de la bioinformática 8

2.1.1. Bases de datos biológicas 8

2.1.2. Alineamiento de secuencias 13

2.1.3. Predicción de genes y promotores 26

2.1.4. Filogenética molecular 27

2.1.5. Bioinformática estructural 28

2.1.6. Genómica funcional 29

2.2. Redes Reguladoras de Genes 30

2.2.1. Métodos para la inferencia de redes reguladoras de genes . . . 32

3. Computación evolutiva 39

3.1. Algoritmos evolutivos . . . 39

3.1.1. Componentes de los algoritmos evolutivos . . . 41

3.1.2. Paradigmas principales de los algoritmos evolutivos . . . 45

3.2. Optimización . . . 46

3.3. Evolución diferencial . . . 47

3.3.1. Variantes de la evolución diferencial . . . 53

4. GPGPU y estado del arte 55

4.1. GPGPU . . . 55

4.2. Conceptos de cómputo paralelo . . . 57

4.2.1. Clasificación . . . 57

4.2.2. Granularidad . . . 61

4.3. GPU . . . 62

4.4. Historia de los GPUs . . . 62

4.5. CUDA . . . 65

4.6. Arquitectura de los GPUs . . . 66

4.7. Uso de CUDA . . . 68

4.8. Estado del arte . . . 71

4.8.1. Paralelización de algoritmos evolutivos . . . 71

4.8.2. Cómputo evolutivo en GPUs . . . 72

5. Propuesta de un AE basado en GPUs para la inferencia de modelos de GRN 79

5.1. Evaluación de la aptitud para la inferencia de modelos de GRN mediante AEs . . . 79

5.2. Paralelización del algoritmo 82
5.3. Implementación en CUDA 83
5.3.1. Reserva de memoria 83
5.3.2. Generación de la población inicial en paralelo 85
5.3.3. Evaluación de la población en paralelo 88
5.3.4. Operadores de la evolución diferencial en paralelo 88
5.3.5. Selección en paralelo 89

6. Diseño Experimental y Análisis de Resultados 91
6.1. Infraestructura 92
6.2. Diseño experimental 92
6.3. Validación de la implementación 93
6.4. Resultados 96
6.5. Discusión de resultados 98

7. Conclusiones y trabajo futuro 99
7.1. Conclusiones 99
7.2. Trabajo Futuro 101

A. Métodos numéricos para resolver sistemas de ecuaciones diferenciales 103
A.1. Método de Euler 104
A.2. Método de Runge-Kutta 105

B. Requerimientos de CUDA 107
B.1. Procesador gráfico con arquitectura CUDA 107
B.2. El controlador de desarrollo de Nvidia 109
B.3. El conjunto de herramientas de CUDA (*CUDA Toolkit*) 109

B.4. Un compilador de C estándar 110

C. Publicaciones Realizadas **113**

Índice de figuras

2.1. Un ejemplo de un emparejamientos de secuencias, donde se muestra la diferencia entre un alineamiento global y uno local; en el global se incluyen todos los residuos de las dos secuencias. La región con mayor semejanza está encerrada en el recuadro. El alineamiento local sólo incluye las porciones de las dos secuencias en las que se encontró una máxima semejanza. En la figura, los " : " significan que los residuos son idénticos y "." indica que los residuos son similares. 35
2.2. Ejemplo del método de la matriz de puntos 35
2.3. Dogma central de la biología molecular. 36
2.4. Representación gráfica de una red reguladora de genes. 37

3.1. Diagrama de flujo de un algoritmo evolutivo. 42
3.2. Ejemplo de la distribución de soluciones en la población inicial de la evolución diferencial. 50
3.3. Representación gráfica de la recombinación en la evolución diferencial. 50
3.4. Representación gráfica de la mutación en la evolución diferencial. . . 51

4.1. Clasificación de Flynn - SISD. 59

4.2. Clasificación de Flynn - MISD. 59
4.3. Clasificación de Flynn - SIMD. 60
4.4. Clasificación de Flynn - MIMD. 60
4.5. Arquitectura de GPU CUDA. 69

5.1. Diagrama de flujo de la propuesta de la tesis. 84
5.2. Reservación de memoria en el GPU. Cada individuo guarda los datos correspondientes a las matrices G y H y alos vectores α y β. 86

6.1. Instancia de pruebas Tominaga. 94
6.2. Datos objetivo y estimados para el gen 1. 94
6.3. Datos objetivo y estimados para el gen 2. 96

Índice de tablas

2.1. Principales bases de datos biológicas disponibles en Internet. 34

4.1. Clasificación de Flynn. 57

4.2. Las 5 supercomputadoras con mayor poder de cómputo a junio de 2011. 70

5.1. Datos de ejemplo. 80

5.2. En la tabla en algoritmo 1, se paralelizan todos los operadores del AE, mientras que en el algoritmo 2, sólo se parelliza la evaluación de los individuos 83

6.1. Especificación técnica del GPU utilizado para la realización de la tesis. 92

6.2. Especificación técnica del CPU utilizado para la realización de la tesis. 93

6.3. Intervalo de búsqueda de los parámetros del sistema S. 93

6.4. Instancia de pruebas de Tominaga [2]. 95

6.5. Comparación de los tiempos computacionales entre las versión secuencial y la versión implementada en CUDA para 100 generaciones. . . . 97

6.6. Comparación de los tiempos computacionales entre las versión secuencial y la versión implementada en CUDA para 500 generaciones. . . . 97

6.7. Comparación de los tiempos computacionales entre las versión secuencial y la versión implementada en CUDA para 1000 generaciones. . . 98

B.1. Características por versión de capacidad de cómputo. 108

Capítulo 1

Introducción

Las redes reguladoras de genes representan las interacciones que ocurren entre ellos. Esto es, el nivel de influencia que tienen uno o un grupo de genes sobre otro gen o grupos de genes. Estas interacciones pueden provocar que un gen sea inhibido o promovido (incremente su nivel de expresión). En esta tesis se presenta un algoritmo que puede realizar la inferencia de un modelo matemático que describa las interacciones ocurridas en los genes. Para realizar esto, se utilizó la capacidad de cómputo paralelo de los GPUs.

1.1. Motivación

La inferencia de modelos de redes reguladoras de genes, tiene una gran aplicación en diferentes de áreas, entre las que podemos destacar, el desarrollo de nuevos tratamientos para enfermedades que en la actualidad no tienen cura. También tienen aplicaciones en el área del diseño biológico inteligente, ya que los genes se podrían manipular para obtener el funcionamiento deseado en un organismo biológico (por ejemplo, el uso de bacterias para limpiar los mares de accidentes petroleros). Sin

embargo, los métodos actuales basados en el enfoque de los algoritmos evolutivos, requieren una gran cantidad de evaluaciones para aproximar modelos aceptables, lo cual hace prohibitivo su uso, excepto para instancias muy pequeñas. Debido a esta posible aplicación y a las disminuciones tan considerables de tiempo de cómputo que se han obtenido al implementar algoritmos en GPUs, ahora se ve el uso de GPUs como una alternativa viable para obtener modelos aceptables en un tiempo razonablemente corto.

1.2. Planteamiento del problema

La inferencia de modelos de redes reguladoras de genes consiste en un problema de ingeniería inversa en el cual se intentan describir, mediante un modelo matemático, las interacciones ocurridas en una red reguladora de genes, a partir de las mediciones hechas a un grupo de genes. Se plantea la solución de la inferencia de modelos de redes reguladoras de genes usando un algoritmo evolutivo implementado en GPUs.

1.3. Objetivo

El objetivo principal de esta tesis es realizar el algoritmo de la evolución diferencial en una GPU utilizando CUDA[1]. Este algoritmo deberá ser capaz de resolver el problema de la inferencia de redes reguladoras de genes mediante el uso de sistemas S.

[1]CUDA son las siglas de *Compute Unified Device Architecture*, una arquitectura para computo de alto rendimiento desarrollada por Nvidia que se mostrará en el capítulo 4

1.4. Contribuciones

Esta tesis presenta, de acuerdo a la literatura consultada, la primera implementación de evolución diferencial en GPUs para resolver el problema de la inferencia de modelos de redes reguladoras de genes.

1.5. Organización de la tesis

La tesis se encuentra organizada de la siguiente manera

- **Capítulo 2. Bioinformática:**
  En el segundo capítulo se presenta una introducción a la bioinformática, y se describen sus principales áreas actuales de investigación:

  - Bases de datos biológicas
  - Alineamiento de secuencias
  - Predicción de genes y promotores
  - Filogenética molecular
  - Bioinformática estructural
  - Genómica funcional
  - Redes reguladoras de genes

  En la parte final del capítulo se describen las redes reguladoras de genes, dando nociones de ellas desde el punto de vista biológico, e indicando porqué es importante entenderlas. Adicionalmente, se presentan algunos de los métodos disponibles para modelarlas.

- **Capítulo 3. Computación evolutiva:**

  En el tercer capítulo de esta tesis se presentan los conceptos básicos de la computación evolutiva, sus componentes fundamentales y sus principales paradigmas. Posteriormente, se describen los conceptos elementales de optimización, se define lo que es un problema de optimización y se mencionan algunos de los métodos para resolver este tipo de problemas.

- **Capítulo 4. GPGPU y estado del arte:**

  En el cuarto capítulo se muestran los principales conceptos de cómputo paralelo. Posteriormente, se presenta una introducción a los GPUs, una breve reseña historica de éstos y se finaliza el capítulo presentando la tecnología de CUDA, utilizada en esta tesis. El capítulo finalza mostrando el estado del arte de las diferentes áreas que están involucradas en la realización de esta tesis.

- **Capítulo 5. Propuesta de un algoritmo evolutivo basado en GPUs para la inferencia de modelos de redes reguladoras de genes:**

  Se describe el método para la inferencia de modelos de redes reguladoras de genes utilizando algoritmos evolutivos mediante sistemas S. El capítulo finaliza con una descripción de la propuesta implementada en GPUs utilizando CUDA.

- **Capítulo 6. Resultados:**

  En este capítulo se revisaran los resultados obtenidos de la implementación.

- **Capítulo 7. Conclusiones y trabajo futuro:**

  En este último capítulo se enuncian las conclusiones principales de esta tesis y el posible trabajo futuro, para extender el trabajo aquí presentado.

Capítulo 2

Bioinformática

La bioinformática es la aplicación de las ciencias computacionales al manejo y análisis de datos biológicos, A la bioinformática, también se le conoce como biología computacional ó biocomputación [3]. La bioinformatíca es un área de investigación interdisciplinaria que hace referencia a diferentes campos de estudio entre los que destacan ciencias computacionales, matemáticas aplicadas, estadística, inteligencia artificial, química y bioquímica.

La bioinformática estudia el desarrollo de métodos computacionales y técnicas estadísticas para resolver problemas prácticos y teóricos derivados del almacenamiento, extracción y distribución de información relativa a macromoléculas biológicas como el ADN (ácido desoxirribonucleico), ARN (ácido ribonucleico) y proteínas. Aunque, como ya se mencionó anteriormente, los términos bioinformática y biología computacional suelen considerarse sinónimos, el NIH (*National Institute of Health*) los define de una manera más explícita [3]:

- **Bioinformática** es la investigación, desarrollo o aplicación de herramientas computacionales y aproximaciones para la expansión del uso de datos biológicos, médicos, conductuales o de salud, incluyendo aquellas herramientas que sirvan

para adquirir, almacenar, organizar, analizar o visualizar tales datos.

- **Biología computacional** es el desarrollo y aplicación de métodos teóricos y de análisis de datos, modelado matemático y técnicas de simulación computacional al estudio de sistemas biológicos, conductuales y sociales.

El objetivo principal de la bioinformática es lograr entender mejor las células y la manera en que funcionan a nivel molecular. Mediante el análisis de secuencias moleculares y datos estructurales la investigación en bioinformática puede generar nuevas ideas y proporcionar una perspectiva "global" de la célula.
La razón por la que las funciones de las células pueden ser entendidas mejor mediante el análisis de datos de las secuencias es porque el flujo de la información genética es dictado por el "*dogma central*" de la biología molecular, el cual indica que el ADN se transcribe en ARN, el cual a su vez se transcribe en proteínas. Las funciones celulares se llevan a cabo principalmente por proteínas cuyas capacidades están determinadas por sus secuencias. Por lo tanto, la solución de problemas funcionales usando secuencias y, en ocasiones, enfoques estructurales han probado ser una tarea fructífera.
La bioinformática consta de dos subcampos que se complementan entre sí:

- El desarrollo de herramientas computacionales y bases de datos.
- La aplicación de estas herramientas y bases de datos para generar nuevo conocimiento biológico que ayude a entender de una mejor manera los sistemas vivos.

La bioinformática no sólo se ha vuelto esencial para la investigación genómica y la biología molecular, sino que ha impactado de una manera importante en áreas de la biotecnología y las ciencias biomédicas. Ha tenido aplicaciones en áreas como: en el diseño de nuevas medicinas, el análisis forense y la agricultura. Estudios computacionales de las interacciones entre las proteínas proporcionan una base racional para la

rápida identificación de nuevas drogas sintéticas.

El conocimiento de las estructuras tridimensionales de las proteínas permite diseñar moléculas de forma que sean capaces de unirse al sitio receptor de una proteína objetivo (como un antígeno) con gran afinidad y especificidad. Este enfoque basado en bioinformática, reduce los recursos necesarios para desarrollar medicamentos más efectivos y con menos efectos secundarios que con el enfoque de prueba y error.

Sin embargo, a pesar del gran potencial de la bioinformática hay que estar conscientes de sus limitaciones, para evitar sobreestimar esta disciplina. La biología experimental y la bioinformática son actividades independientes pero complementarias. La bioinformática depende de la biología experimental para producir datos primarios de análisis y, a su vez, la bioinformática proporciona la interpretación de los datos experimentales e importantes pistas para seguir con la investigación experimental.
Las predicciones hechas de la bioinformática no son pruebas formales de ningún concepto, y no reemplazan los métodos de experimentación tradicional. La calidad de las predicciones hechas por la bioinformática depende de la calidad de los datos y la sofistificación de los algoritmos utilizados.
La bioinformática no es, de manera alguna, un campo maduro. La mayoría de los algoritmos que en ella se utilizan carecen de la capacidad y sofisticación para reflejar la realidad verdadera, y éstos a menudo realizan predicciones incorrectas que no tienen ningún sentido cuando sus soluciones se colocan en un contexto biológico. Los resultados dependen también del poder de cómputo disponible. Muchos algoritmos exactos pero exhaustivos no se pueden utilizar debido a la gran demanda de cómputo que requieren, haciéndolos en ocasiones inviables debido al tiempo necesario para producir un resultado.

Es por eso que se prefiere utilizar algoritmos que sólo dan aproximaciones pero que son menos costosos, computacionalmente hablando. Dentro de la bioinformática existen muchas áreas de investigación como por ejemplo:

- Alineamiento de secuencias
- Predicción de genes
- Ensamble del genoma
- Alineamiento de estructuras de proteínas
- Predicción de estructuras de proteínas
- Modelado de la evolución
- Predicción de la expresión génica
- Genómica funcional

En la siguiente sección describiremos con más detalle algunas de las principales áreas de la bioinformática.

2.1. Áreas de investigación de la bioinformática

2.1.1. Bases de datos biológicas

En la actualidad podemos encontrar diferentes tipos de modelos de bases de datos (BD) (archivos planos, modelo relacional y orientado a objetos) para almacenar información biológica. Aunque modelos como los del tipo relacional presentan ventajas

para el tratamiento de la información biológica, las bases de datos en archivos planos son muy comunes, y su uso se suele justificar diciendo que los biólogos no necesitan demasiados conocimientos en el diseño de bases de datos para poder utilizarlas. Además, el manejo de los resultados al realizar una búsqueda es fácilmente comprensible por ellos.

De acuerdo a su contenido, las bases de datos pueden ser catalogadas en tres tipos [4]:

- Bases de datos primarias
- Bases de datos secundarias
- Bases de datos especializadas

Bases de datos primarias

Las bases de datos primarias contienen datos biológicos originales. Las tres principales bases de datos que contienen la información original o datos crudos de las secuencias de ADN producidas y sometidas por investigadores de todo el mundo son: GENBANK [5] del NCBI[1], DDBJ [6] DNA Data Bank of Japan del CIB[2] y el EMBLBANK o EMBL Nucleotide Sequence [7] del EMBL-EBI[3] las cuales se encuentran disponibles libremente en internet. La mayoría de la información de estas bases de datos es ingresada directamente por los autores con un nivel mínimo de anotación.

Actualmente, una condición para la publicación de trabajos científicos, ya sea para revistas o conferencias de bioinformática, consiste en someter la información de las

[1]El Centro Nacional para la Información Biotecnológica o National Center for Biotechnology Information (NCBI), una rama de los Institutos Nacionales de Salud (National Institutes of Health o NIH) en Estados Unidos.

[2]Centro de Información Biológica (Japón).

[3]Instituto Europeo de Bioinformática, el cual pertenece al Laboratorio Europeo de Biología Molecular (European Molecular Biology-European Bioinformatics Institute).

secuencias a alguna de estas BD. Esto para garantizar el libre acceso a la información de los datos moleculares fundamentales que se hayan usado en los trabajos de investigación. Estas tres BD se pueden usar de manera colaborativa y juntas conforman la INSDC (*International Nucleotide Sequence Database Collaboration*).
Aunque existe consistencia en cuanto a la información de datos disponible entre estas tres BD, esto es, la información de una secuencia de nucleótidos es la misma para las tres BD, cada una de ellas tiene un formato diferente para representar esta información. Esto genera un problema obvio, si bien existen excepciones como por ejemplo, el caso de la información de la estructura tridimensional de macromoléculas biológicas, en el cual sólo existe una base de datos centralizada, la PDB (*Protein Data Bank*). Esta base de datos almacena la información de las coordenadas atómicas de las macromoléculas tanto para proteínas como para ácidos nucleicos. Para determinar estos datos se utilizan técnicas como la cristalografía de rayos X y la resonancia magnética nuclear. Esta base de datos utiliza un archivo de texto plano para representar el nombre de la proteína, los autores, detalles experimentales, estructuras secundarias, cofactores y coordenadas atómicas.

Bases de datos secundarias

Las anotaciones de información en las BD primarias es a menudo muy limitada. Para transformar esta información original o en estado crudo en un conocimiento biológico más sofisticado, se requiere una gran cantidad de post-procesamiento de los datos. Es por eso que se requieren las BD secundarias que contienen información computacionalmente procesada de las BD primarias. La base de datos Uniprot [8] es un claro ejemplo de una base de datos secundaria la cual provee anotaciones detalladas de las secuencias como son: estructura, función y a qué familia de proteína pertenecen. La información de UniProt se deriva principalmente de TrEMBL, una base de datos

de secuencias traducidas de ácidos nucleicos almacenadas en la base de datos EMBL.

Cada una de las anotaciones son realizadas cuidadosamente por personas expertas, por lo que son de buena calidad. Dentro de las anotaciones de la proteína que podemos encontrar están: función, estructura de dominios, sitios catalíticos, cofactor vinculante, modificación post-traducción, información de vía metabólica, asociación con enfermedades, y similitud con otras secuencias. Mucha de la información anotada es obtenida de la literatura científica y es ingresada por los curadores de la base de datos. La información con anotaciones provee un valor agregado a los registros de secuencias originales. También hay anotaciones con referencias cruzadas hacia otros sitios de interés. Además, características tales como la baja redundancia de datos han ocasionado que esta base de datos sea muy popular entre los biólogos.

También existen otras BD secundarias que se relacionan con las proteínas de acuerdo a su función o estructura. Por ejemplo, las BD Pfam y Blocks contienen información sobre alineamientos de proteínas, de motivos derivados y patrones, las cuales pueden ser usadas para clasificar familias de proteínas e inferencia de funciones de las proteínas. La base de datos DALI es otro ejemplo de base de datos secundaria en la cual se encuentra la información de las estructuras de proteínas vital para su clasificación y para la identificación de relaciones evolutivas entre proteínas.

Bases de datos especializadas

Las BD especializadas normalmente sirven a una comunidad científica en particular o se centran en un organismo en particular. El contenido de estas bases pueden ser secuencias u otros tipos de información. Las secuencias contenidas en estas BD pueden traslaparse con las de una BD primaria pero también pueden tener información adicional proveída por los autores. Muchas BD genómicas que pertenecen a una

taxonomía en específico caen dentro de las BD especializadas.

Algunos ejemplos son la FlyBase [9] y la WormBase [10], entre otras. Además, existen BD especializadas que contienen datos originales derivados de análisis funcionales. Por ejemplo, el GenBank EST y la Micro Array Expression Database del Instituto Europeo de Bioinformática son BD disponibles con información sobre expresión de genes.

Interconexión entre las bases de datos biológicas

Debido a que las BD primarias son repositorios y distribuidores centrales de información sobre secuencias y estructuras, es necesario que las BD secundarias y especializadas se conecten con las BD primarias. Debido a que el usuario normalmente necesita información tanto de BD primarias, como secundarias, ya que muchas veces la información contenida en una sola BD es insuficiente, existen referencias cruzadas y ligas entre las BD para facilitar la navegación por la información sobre una entrada de interés.

El principal problema para ligar las BD biológicas es la incompatibilidad de los formatos que se utilizan entre éstas, ya que algunas pueden ser archivos planos, otras, relacionales u orientadas a objetos. Una alternativa adoptada para poder ligar las BD es el uso de CORBA o XML.

Desventajas de las bases de datos biológicas

Han sido identificados algunos problemas existentes en las BD biológicas, de entre los cuales podemos mencionar los siguientes:

- Uno de los problemas en las BD biológicas es el exceso de confianza en la información de secuencias y anotaciones, sin tomar en cuenta la fiabilidad de la información.

- A menudo se ignora el hecho de que hay muchos errores en las BD de secuencias, debido a errores de secuenciación.

- Algunos de estos errores causan desplazamientos que hacen que toda la identificación del gen se dificulte o que la traducción de la proteína sea imposible.

- Estos errores pueden propagarse a otras BD, siendo un problema grave.

Existen sistemas diseñados para obtener información de las BD, haciendo más amigable estas consultas. Algunas de ellos son Entrez y SRS (Sequence Retrieval Systems), los cuales proveen acceso a múltiples BD para la recuperación integrada de búsquedas, además de permitir el realizar consultas complejas mediante operadores booleanos.

En la tabla 2.1 se muestran algunas de las principales BD biológicas disponibles en Internet.

2.1.2. Alineamiento de secuencias

El alineamiento de secuencias es una de la principales tareas de la bioinformática. Es un paso importante hacia el análisis funcional y estructural de las nuevas secuencias descubiertas. Debido al crecimiento exponencial de nuevas secuencias, la comparación de secuencias se ha vuelto más importante ya que es posible inferir funciones e información evolutiva sobre una nueva proteína comparándola con proteínas existentes en la BD.

El alineamiento de secuencias es el proceso mediante el cual diferentes secuencias son comparadas buscando patrones comunes y estableciendo correspondencias residuo-residuo. El alineamiento de pares de secuencias es la base para la búsqueda de similitudes en bases de datos y el alineamiento múltiple de secuencias.

Alineamiento de pares de secuencias

Las proteínas y el ADN son producto de la evolución y los elementos básicos que componen a estas macromoléculas (ácidos nucleicos y aminoácidos) forman secuencias lineales que determinan la estructura primaria de las moléculas. Estas moléculas pueden ser consideradas como fósiles moleculares que contienen la información de millones de años de evolución. Durante este periodo, las moléculas han sido sometidas a cambios aleatorios llamados mutaciones que las hacen diferir de otras. Y aunque a lo largo del tiempo estas secuencias acumulan mutaciones que las hacen diverger, aún pueden conservarse rastros de la evolución en ciertas porciones de las secuencias, haciendo posible la identificación de ancestros comunes. Esto debido a que algunos de estos residuos comunes juegan un papel clave a nivel funcional o estructural y por eso han sido preservados por la selección natural, de tal forma que los residuos con menor importancia para la estructura o función tienden a mutar más fácilmente. Por ejemplo, las residuos activos de una familia de enzimas tienden a conservarse debido a que son los responsables de funciones catalíticas. Por lo tanto, comparando las secuencias mediante alineamientos pueden ser identificados patrones de conservación o de variación. El grado de conservación en el alineamiento de secuencias revela la relación evolutiva de secuencias diferentes, mientras que la variación entre dos secuencias refleja los cambios que han ocurrido durante la evolución mediante substituciones, inserciones y eliminaciones.

Identificar relaciones evolutivas entre secuencias ayuda a caracterizar las funciones de secuencias desconocidas. Cuando un alineamiento de secuencias revela una similitud significativa entre un grupo de secuencias, entonces se puede considerar que pertenecen a la misma familia. Si un miembro de esa familia tiene una estructura o funciones conocidas, entonces esa información puede ser transferida para aquellas que no han sido caracterizadas experimentalmente. Por lo tanto, el alineamiento de secuencias

puede ser usado como una base para la predicción de una estructura o función de secuencias que no han sido caracterizadas.

Los alineamientos de secuencias proveen información sobre la relación existentes entre dos secuencias. Si dos secuencias comparten similitudes significativas, es altamente improbable que esta similitud haya sido obtenida mediante mutaciones aleatorias, de tal forma que es más probable que estas dos secuencias provengan de un origen evolutivo común. Cuando el alineamiento de secuencias es realizado correctamente, éste refleja la relación evolutiva entre dos secuencias:

- regiones que son alineadas, pero no idénticas representan sustituciones de residuos.
- regiones donde los residuos de una secuencia no corresponden a ninguno de los de las otras secuencias, corresponden a inserciones o eliminaciones ocurridos durante la evolución.

Un concepto importante en el análisis de secuencias es la homología de secuencias. Se dice que dos secuencias tienen una relación homóloga si ambas descienden de un origen evolutivo común. A diferencia de la similitud entre secuencias, la homología mide el porcentaje de residuos que son similares en cuanto a propiedades fisicoquímicas tales como el tamaño, carga, e hidrofobicidad. La homología de secuencias es una conclusión o inferencia acerca de la relación ancestral común hecha cuando dos secuencias comparten un grado suficiente de similitud. Por otro lado, la similitud es el resultado directo de la observación de un alineamiento de secuencias cuantificándolo mediante porcentajes.

Otros conceptos relacionados en la comparación de secuencias es la similitud e identidad de secuencias. En el caso de secuencias de nucleótidos, estos dos términos son sinónimos pero en secuencias de proteínas son términos muy diferentes. En el alineamiento de secuencias de proteínas, la identidad de secuencias se refiere al porcentaje

de coincidencias de los mismos residuos de aminoácidos entre dos secuencias alineadas, mientras que la similitud de secuencias se refiere al porcentaje de residuos alineados que tienen características fisicoquímicas similares y que pueden ser substituídos entre sí.

Hay dos maneras para calcular la similitud e identidad: Una involucra el uso del total de las longitudes de las dos secuencias y la otra la normaliza en función de la secuencia más corta. El primer método utiliza la siguiente fórmula:

$$S = [(L_s \times 2) \,/\, (L_a + L_b)] \times 100 \tag{2.1}$$

Donde S es el porcentaje de similitud de las secuencias, L_S es el número de residuos alineados con características similares, y L_a, L_b son las longitudes totales de cada secuencia.
La identidad de las secuencias puede ser calculada de la siguiente manera:

$$I = [(L_i \times 2) \,/\, (L_a + L_b)] \times 100 \tag{2.2}$$

donde L_i es el número de residuos alineados idénticos.

El objetivo de un alineamiento de pares de secuencias consiste en encontrar el mejor emparejamiento de dos secuencias, de tal forma que se obtenga una máxima correspondencia entre los residuos. Para lograr este objetivo, una secuencia debe ser desplazada en relación a la otra para encontrar la posición donde se encuentra el máximo número de coincidencias. Existen dos estrategias de alineamiento que son usadas a menudo:

- Alineamiento global: Donde se presupone que las secuencias son similares en toda su longitud. En este caso, el alineamiento se lleva a cabo a lo largo de las

secuencias para encontrar el mejor posible. Este método es más aplicable para cadenas similares con longitudes semejantes

- Alineamiento local: En esta estrategia no se presupone que las dos secuencias tienen similitud en toda su longitud, sino que localizan regiones locales en las que existe un gran nivel de similitud entre las dos secuencias y se alinean estas regiones sin preocuparse por el resto de las secuencias. Este enfoque puede ser utilizado para alinear secuencias más divergentes con el objetivo de encontrar patrones que se hayan conservado en el ADN o en las secuencias de proteínas. Las dos secuencias pueden ser de diferente tamaño a diferencia del alineamiento global. En la figura 2.1 se muestra la diferencia entre un alineamiento global y uno local.

Los algoritmos de alineamientos tanto locales como globales, son fundamentalmente similares. Solo difieren en la estrategia de optimización utilizada para alinear residuos similares. Ambos tipos de algoritmos pueden estar basados en uno de los siguientes métodos:

- Matriz de puntos
- Programación dinámica
- Método de la palabra

El método de la matriz de puntos es una forma gráfica de representar el alineamiento de dos secuencias. Se utiliza una matriz bidimensional en la que las secuencias a comparar se escriben en los ejes vertical y horizontal y la comparación se realiza poniendo un punto en el caso en el que los residuos de las dos secuencias posean similitud. Cuando se han terminando de colocar los puntos para los residuos de las dos secuencias, se trazan líneas uniendo los puntos de forma diagonal. Estas diagonales

muestran regiones de alineamiento. Si hay interrupciones en una línea diagonal, es porque en esos residuos existe una inserción o eliminación. Si hay líneas paralelas diagonales es porque existen regiones repetitivas de las secuencias. En la figura 2.2 se muestra un ejemplo de una matriz de puntos.

Uno de los métodos más populares es el uso de programación dinámica para determinar el alineamiento de dos secuencias. Al igual que con el método de la matriz de puntos, en este caso se crea una matriz bidimensional, pero se localiza un alineamiento de una forma más cuantitativa ya que en lugar de utilizar puntos, utiliza puntuaciones para contar las coincidencias y divergencias de los residuos de las secuencias. Buscando la puntuación más alta en la matriz se puede obtener el mejor alineamiento de una manera acertada.
El método de programación dinámica consiste primero en inicializar la matriz, construir la matriz de puntaje con base en reglas predefinidas. Puede penalizarse el que dos residuos no coincidan y también se pueden utilizar valores de puntuación que tomen en cuenta características biológicas conocidas. Tal es el caso de la matriz PAM y BLOSUM [11].

En el caso de alineamientos globales, el algoritmo más conocido es el Needleman-Wunsch [12] que se muestra en el algoritmo 1.

Algoritmo 1: Cálculo de la matriz global Needleman-Wunsch.

Entrada: Se define el valor de penalización d

Los valores para la función de similitud de caracteres S( )

Las secuencias A y B

Salida: Matriz de puntuaciones F

for $i \leftarrow 0$ **to** $longitud(A)$ **do**
  $F(i,0) \leftarrow d * i$
end

for $j \leftarrow 0$ **to** $longitud(B)$ **do**
  $F(0,j) \leftarrow d * j$
end

for $i \leftarrow 0$ **to** $longitud(A)$ **do**
  for $j \leftarrow 0$ **to** $longitud(B)$ **do**
    $COINCDE \leftarrow F(i-1,j-1) + S(A_i, B_j)$
    $ELIMINA \leftarrow F(i-1,j-1) + S(A_i, B_j)$
    $INSERTA \leftarrow F(i-1,j-1) + S(A_i, B_j)$
    $F(i,j) \leftarrow max(COINCIDE, ELIMINA, INSERTA)$
  end
end

Después de que se ha calculado la matriz de puntuaciones F, el resultado es la puntuación devuelta por el mejor alineamiento posible. Para obtener las secuencias alineadas se efectúa el procedimiento descrito en el algoritmo 2.

Algoritmo 2: Algoritmo para obtener las secuencias de acuerdo al alineamiento global Needleman-Wunsch.

Entrada: Matriz de puntuaciones F
Salida: Secuencias alineadas GA Y GB

$GA \leftarrow$ “”
$GB \leftarrow$ “”
$i \leftarrow longitud(A)$
$j \leftarrow longitud(B)$
while $i > 0$ *and* $j > 0$ **do**
  $puntaje \leftarrow F(i, j)$
  $diagonal \leftarrow F(i-1, j-1)$
  $arriba \leftarrow F(i, j-1)$
  $izquierda \leftarrow F(i-1, j)$
  if $puntaje == diagonal + S(A_i, B_j)$
  then
    $GA \leftarrow A_i + GA$
    $GB \leftarrow B_j + GB$
    $i \leftarrow i - 1$
    $j \leftarrow j - 1$
  else
    if $puntaje == izquierda + d$
    then
      $GA \leftarrow A_i + GA$
      $GB \leftarrow$ “ $-$ ” $+ GB$
      $i \leftarrow i - 1$
  end
  otherwise $puntaje == arriba + d$
    $GA \leftarrow$ “ $-$ ” $+ GA$
    $GB \leftarrow B_j + GB$
    $i \leftarrow i - 1$
  endsw
end
while $i > 0$ **do**
  $GA \leftarrow A_i + GA$
  $GB \leftarrow$ “ $-$ ” $+ GB$
  $i \leftarrow i - 1$
end
while $j > 0$ **do**
  $GA \leftarrow$ “ $-$ ” $+ GA$
  $GB \leftarrow B_j + GB$
  $j \leftarrow j - 1$
end

Para alineamientos locales, el algoritmo más conocido es el de Smith-Waterman [13].

Algoritmo 3: Algoritmo de alineamiento local Smith-Waterman.

Entrada: Se define el valor de penalización d

Los valores para la función de similitud de caracteres S( )

Las secuencias A y B

Salida: Matriz de puntuaciones F

El algoritmo es parecido al de alineamiento global con algunas modificaciones

for $i \leftarrow 0$ **to** $longitud(A)$ **do**
| $F(i, 0) \leftarrow 0$

end

for $j \leftarrow 0$ **to** $longitud(B)$ **do**
| $F(0, j) \leftarrow 0$

end

Si la puntuación de sub-alineamiento se vuelve negativa, reiniciar la búsqueda

Se llena la matriz de acuerdo a las siguientes reglas.

$$F(i,j) = \text{máx} \begin{cases} 0, \\ F(i-1, j-1) + S(A_i, B_j), \\ F(i-1, j) - d, \\ F(i, j-1) - d \end{cases}$$

Para obtener el alineamiento local se realiza un rastreo hacia atrás comenzando del máximo de $F(i, j)$ en toda la matriz F hasta el primer 0

Búsqueda de similitud en bases de datos

Una de las principales aplicaciones de los algoritmos de alineamientos de pares de secuencias es el poder realizar búsquedas en las bases de datos biológicas. El proceso

de realizar una búsqueda consiste en realizar una comparación de pares de secuencia entre una que se va a consultar y las existentes en la BD consultada. Dentro de los requerimientos para la implementación de búsquedas de secuencias en las bases de datos podemos encontrar:

- **Sensibilidad**: Que podemos definir como la habilidad para encontrar el mayor número de secuencias coincidentes posibles (verdaderos positivos).
- **Selectividad** : también llamado *especificidad* el cual se refiere a la habilidad para excluir resultados incorrectos (falsos positivos).
- **Velocidad** : Se refiere al tiempo que toma recuperar los resultados de la base de datos.

En la búsqueda dentro de bases de datos, así como también en muchas otras áreas de la bioinformática, existen dos tipos fundamentales de algoritmos. El primero es el *exhaustivo*, el cual utiliza un algoritmo para encontrar la mejor y más exacta solución para un problema en particular examinando todas las posibles combinaciones matemáticas posibles (la programación dinámica es un ejemplo). El otro tipo es el *heurístico*, el cual es una estrategia computacional para encontrar soluciones empíricas o aproximadas. Esencialmente, las heurísticas reducen el espacio de búsqueda de acuerdo a algún criterio que, por lo general, involucra procesos estocásticos.

Existen dos algoritmos importantes en la búsqueda de bases de datos:

BLAST [14] (*Basic Alignment Search Tool*) es un algoritmo heurístico de alineamiento de secuencias de tipo local, desarrollado por Stephen Altschul en 1990. Blast hace uso de los algoritmos de Smith-Waterman para realizar alineamientos locales utilizando matrices de puntuación (BLOSUM y PAM) para calificar los alineamientos.

El algoritmo consta de tres etapas principales:

1. **Inicialización**: Se buscan palabras pequeñas en las secuencias de la base de datos que corresponden a fragmentos significativos de la secuencia de consulta. Se consideran significativas las palabras que tengan puntuación mayor a un umbral T y que se encuentren al menos a una distancia A de otra palabra. Definiendo estos parámetros y el tamaño de palabra a buscar se puede ajustar la sensibilidad del resultado.

2. **Extensión**: El alineamiento encontrado se extiende hacia ambos lados de las palabras usando el algoritmo de Smith-Waterman. Esta extensión se realiza hasta que la puntuación del alineamiento llega a un límite establecido.

3. **Evaluación**: Una vez que se ha realizado la extensión de todas las palabras, cada uno de los alineamientos es evaluado para determinar su significación estadística. Se eliminan alineamientos inconsistentes y los alineamientos resultantes son llamados pares de alta puntuación (HPS). De acuerdo a un parámetro dado, se discriminan los alineamientos que se consideran menos significativos.

FASTA[15] FASTA fue la primera herramienta de búsqueda de similitud en bases de datos desarrollada en 1985 por Lipman y Pearson. Utiliza una estrategia de picadillo o *hashing* para encontrar coincidencias para un conjunto pequeño de residuos idénticos llamados k-tuplas (2 residuos en proteínas, 6 en nucleótidos). Ver algoritmo 4.

Algoritmo 4: Algoritmo FASTA.

Crear fragmentos de las secuencias llamados k-tuplas

Identificar las k-tuplas coincidentes entre 2 secuencias utilizando hashing

Mapear los resultados en una matriz bidimensional (diagonales)

Se asignan puntuaciones a las regiones con más densidad de diagonales utilizando una matriz de sustitución.

Los segmentos vecinos con altas puntuaciones en la misma diagonal son unidos para formar un alineamiento.

El alineamiento se refina utilizando el algoritmo de Smith-Waterman.

Se efectúa una validación estadística.

Tanto BLAST como FASTA han obtenido buenos resultados en las búsquedas de bases de datos generales. La diferencia principal entre ellos se encuentra en la inicialización; mientras que BLAST utiliza una matriz de substitución para encontrar coincidencias, FASTA identifica pequeños fragmentos idénticos utilizando *hashing*. FASTA utiliza fragmentos de búsqueda más pequeños por lo que es más sensible pero también más lento que BLAST, mientras que BLAST tiene más selectividad que FASTA porque reduce falsos positivos. Otra ventaja de BLAST es que al final, el algoritmo regresa un conjunto de posibles alineamientos, mientras que FASTA regresa sólo uno.

Alineamiento múltiple de secuencias

Una extensión natural del alineamiento de pares de secuencias es el alineamiento múltiple de secuencias, el cual consiste en alinear múltiples secuencias relacionadas para lograr la mejor coincidencia entre ellas. La ventaja del alineamiento múltiple de secuencias es que permite revelar más información biológica que en un grupo de

alineamiento de pares. Dentro de las aplicaciones de los alineamientos múltiples de secuencias podemos encontrar:

- Análisis filogénetico
- Predicción de estructura secundaria y terciaria de proteínas.

El alineamiento múltiple consiste en acomodar las secuencias de tal forma que el máximo número de residuos de cada secuencia coincida con una función de puntuación. La función de puntuación está basada en la suma de la puntuación de todos los posibles pares de secuencias en un alineamiento múltiple utilizando una matriz de puntuación particular.
Existen algoritmos exhaustivos los cuales realizan el alineamiento de todos las posibles combinaciones. Este tipo de algoritmos suele limitarse a pequeños conjuntos de secuencias (menos de diez).
Existen también algoritmos heurísticos los cuales pueden clasificarse de la siguiente manera :

- **Alineamiento progresivo**: Se van ensamblando progresivamente alineamientos.
- **Alineamiento iterativo**: Consiste en encontrar un alineamiento de baja calidad el cual se va mejorando gradualmente hasta que ya no sea posible.
- **Alineamiento basado en bloques**: Debido a que los anteriores se basan en alineamientos globales, en este caso, se busca alinear pequeños bloques entre las secuencias.

2.1.3. Predicción de genes y promotores

La predicción computacional de genes es uno de los principales pasos para el análisis de la secuencia del genoma. Para genomas de organismos procariotas, los cuales se caracterizan por una alta densidad de genes y poca interrupción entre ellos, la predicción de genes es más fácil que para los genomas de los organismos eucariotas. Los algoritmos actuales para la predicción de genes procariotas, los cuales están basados en modelos ocultos de Markov (GeneMark [16], Glimmer [17], FGENESB [18]) han alcanzado una precisión razonablemente buena. Sin embargo, para el caso de los organismos procariotas todavía persisten varias dificultades en la predicción de genes. Las dificultades principalmente se deben a la baja densidad de genes y a la forma en que se encuentran divididos los genes en el genoma. Los algoritmos actuales pueden dividirse en tres categorías: basados en Ab-initio, basados en homología y basados en consenso. Para los basados en Ab-inito, los algoritmos que usan modelos ocultos de Markov han obtenido mejores resultados para diferenciar los límites de intrones-exones[4]. La principal limitación es la dependencia del entrenamiento de los modelos estadísticos, lo que hace que el método sea específico del organismo. Los métodos basados en consenso en combinación con modelos ocultos de Markov pueden mejorar la precisión de los resultados. Este método, sin embargo, está limitado por la disponibilidad de la información.

Se espera que con los avances en el conocimiento de cómo se encuentra dividido el genoma de los eucariotas, la predicción de genes se vuelva más confiable en el futuro cercano. Para organismos eucariotas algunos programas disponibles son: GRAIL [19], FGENES [18] y GENSCAN [20].

[4]Regiones intercaladas del ADN, dentro de los exones se encuentra codificada la información para producción de proteinas, la función de los intrones aún no se encuentra bien determinada.

2.1.4. Filogenética molecular

La filogenética molecular es el estudio de las relaciones evolutivas existentes entre los organismos utilizando datos moleculares como el ADN y las secuencias de proteínas. Se basa en una serie de suposiciones p.ej., un árbol evolutivo es siempre binario y todas las posiciones de las secuencias evolucionan de manera independiente. Las ramas del árbol definen su topología. El número de posibles topologías del árbol depende del número de taxones y se incrementa de manera extremadamente rápida a medida que el número de taxones crece. Un árbol basado en la secuencia genética no siempre se correlaciona con la evolución de la especie; por eso se debe tener precaución en la extrapolación de los resultados de los arboles fiilogenéticos. Un árbol genético puede ser con raíz o sin raíz. La mejor manera de establecer la raíz del árbol es utilizar un individuo externo de entre los individuos a analizar. El primer paso en la construcción de un árbol filogenético es decidir si se utilizarán secuencias de ADN o secuencias de proteínas. Cada uno de éstos tiene ventajas, así como también limitaciones. Las secuencias de proteínas son preferibles en la mayoría de los casos. Sin embargo, para el estudio de la evolución reciente, las secuencias de ADN son mejores. El segundo paso es realizar un alineamiento de secuencias múltiple. La obtención de la alineación correcta es crítica para la correcta construcción del árbol filogenético. El único detalle de los alineamientos múltiples para el análisis filogenético es que regularmente se requiere truncar manualmente regiones de alineamiento que son ambiguas. El siguiente paso es seleccionar un modelo de substitución adecuado que provea estimaciones de los verdaderos sucesos evolutivos tomando en cuenta múltiples eventos de sustitución. Distancias evolutivas correctas son usadas tanto en los métodos de construcción de árboles basados en distancia como en los basados en probabilidad. Los modelos más comúnmente usados para la sustitución de nucleótidos son los propuestos por Jukes-Cantor [21] y Kimura [22]. Los modelos de sustitución comúnmente usados en el caso

de aminoácidos son los PAM [23] y JTT [24]

2.1.5. Bioinformática estructural

Las proteínas son las responsables de la mayoría de las funciones celulares. El conocimiento de la estructura de las proteínas es esencial para entender su funcionamiento y comportamiento. Las proteínas son polipéptidos formados por la unión de aminoácidos mediante enlaces peptídicos. El plegamiento de un polipéptido puede ser descrito por ángulos de rotación alrededor de los enlaces de la cadena principal. Estos ángulos son ϕ y ψ. El grado de rotación depende de la conformación de la proteína. Los ángulos ϕ y ψ de una proteína pueden ser especificados en una gráfica de Ramachandran. Existen cuatro niveles de estructuras de proteínas:

- **Estructura primaria**: Esta estructura se refiere a la secuencia de residuos de aminoácidos.
- **Estructura secundaria**: La estructura secundaria de las proteínas es el plegamiento regular local entre residuos aminoacídicos cercanos de la cadena polipeptídica. Se adopta gracias a la formación de enlaces de hidrógeno entre las cadenas laterales (radicales) de aminoácidos cercanos en la cadena que incluyen los hélices α y las hojas plegadas β.
- **Estructura terciaria**: La estructura terciara es la conformación tridimensional de la cadena de polipéptidos.
- **Estructura cuaternaria**: La estructura cuaternaria es un complejo arreglo de múltiples cadenas polipetídicas.

Las estructuras de las proteínas son estabilizadas por interacciones electrostáticas, enlaces por puentes de hidrógeno e interacciones van der Waals. Las proteínas pueden

ser clasificadas como globulares o de membrana. Las globulares existen en solventes a través de interacciones hidrofílicas con moléculas solventes. Las de membrana existen en lípidos de membrana y se estabilizan por medio de interacciones hidrofóbicas con las moléculas de lípidos. Las estructuras de las proteínas pueden ser determinadas mediante el uso de cristalografía de rayos x y espectroscopia NMR.

La visualización de representaciones de estructuras de proteínas es el primer paso para entender su funcionamiento. Existen diferentes algoritmos utilizados para la visualización de proteínas: el método intermolecular, que implica la transformación de coordenadas atómicas de las estructuras para obtener la superposición óptima; el método intramolecular que construye una matriz de distancias entre residuos y compara la matriz con otra molécula; y el método que combina ambos enfoques.

2.1.6. Genómica funcional

El campo de la genómica abarca dos áreas principales, genómica estructural y genómica funcional. La genómica estructural principalmente se ocupa con estructuras del genoma enfocándose en el estudio del mapeo y ensamble del genoma, así como también en la anotación y comparación del genoma.

La genómica funcional se basa en gran parte en resultados experimentales, enfocándose en las funciones del gen a nivel de todo el genoma usando enfoques de gran throughput (cantidad de información). El enfoque adoptado aquí es en el "*throughput*", el cual es el análisis simultáneo de todos los genes en el genoma. Esta característica es de hecho la que separa la genómica de la biología molecular tradicional, que estudia un solo gen a la vez. El análisis de esta gran cantidad de información es también llamado análisis transcriptoma, que es el análisis de expresión del conjunto completo de moléculas de ARN en una célula bajo ciertas condiciones.

El análisis transcriptoma facilita el entendimiento de como los genes trabajan en con-

junto para formar caminos metabólicos, regulativos y de señalización en la célula. Revela patrones de co-expresión y co-regulación de los genes y permite determinar funciones en genes que no habían sido caracterizados previamente.
El análisis transcriptoma usando microarreglos de ADN, ESTs y SAGE forman la base de la genómica funcional y es la clave para el entendimiento de las interacciones de los genes y su regulación a nivel del genoma completo.

Más adelante se explicarán con detalle los microarreglos de ADN.

2.2. Redes Reguladoras de Genes

Es conveniente introducir conceptos básicos de biología para definir lo que son las redes reguladoras de genes.

En nuestro planeta el ADN de todos los seres vivos están compuestos de los mismos elementos básicos: los 4 nucleótidos (adenina, citocina, guanina y tiamina). Las cadenas de ADN contienen la información codificada para producir cada proteína o molécula de ARN presente en un organismo [25].

Cada célula de un organismo biológico tiene unas moléculas llamadas cromosomas, las cuales se encuentran constituídas por cadenas de ADN. Esta cadena de ADN puede ser dividida en pequeños fragmentos conocidos como genes, los cuales contienen la información necesaria par construir una molécula de ARN o proteína.

Dentro de cada gen existen regiones llamadas exones y otras intrones. La diferencia entre estas regiones es que se sabe que los exones contienen la información codificada para la generación de proteínas, mientras que la función específica de los intrones no se ha logrado determinar.

Dentro de la célula existe un mecanismo que reconoce cuando inicia un gen o grupo de genes. Esto se hace mediante una sección del ADN llamada promotor. El promotor de un gen es la parte de la secuencia de ADN que indica que se debe de iniciar una transcripción de ADN a ARN. Una vez que se reconoce el inicio de un gen, este gen es copiado en una molécula de ARN. Al ARN resultante se le conoce como ARN mensajero (ARNm) y tiene exactamente la misma secuencia de nucleótidos que la secuencia del gen, con la diferencia de que en el ARN no se encuentra la tiamina (T) ya que es sustituida por el uracilo (U). A este proceso se le conoce como transcripción. El ARNm es utilizado en los orgánulos celulares llamados ribosomas para la fabricación de una proteína. Los ribosomas están hechos de proteínas y una forma de ARN llamado ribosomal (ARNr). Los ribosomas toman como entrada 2 moléculas: una de ARNm y otra de ARNt (ARN transferente). Las moléculas de ARNt son las que implementan el código genético. A este proceso se le llama transducción, y mediante él se convierte el código de ARN en una proteína. Este proceso constituye el dogma central de la biología molecular como se muestra en la figura 2.3.

El dogma central de la biología molecular 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 del ADN. Además, propone que la expresión de la información sólo va en una dirección. Es decir, que el ADN es transcrito a ARN mensajero, y que este es traducido a proteínas, y que estas realizan las funciones metabólicas de las células. El dogma también postula que sólo el ADN puede replicarse y por tanto, reproducirse y transmitir información genética a la descendencia. El dogma fue propuesto por Francis Crick en 1970 [26].

Cuando los genes se expresan, se ven reflejados en el fenotipo del individuo. Estos genes son los encargados de producir proteínas, las cuales realizan funciones metabólicas.

Las redes reguladoras de genes (GRN) por sus siglas en inglés de *gene regulatory networks*, describen la forma en que interactúan los genes. Es decir, representan qué tanto afectan a la expresión de cierto gen. Esto puede ocurrir a diferentes niveles ya que es posible que varios genes creen una proteína que es necesaria para que otro gen se exprese o cierta proteína realice una función metabólica que inhiba la expresión de algún gen o grupo de genes. Estas relaciones se pueden ver, por tanto, como una interacción a tres niveles (genes, proteínas y funciones metabólicas). Es por eso que este tipo de sistemas es sumamente complejo (ver figura 2.4, en la cual se muestra la representación gráfica de una red reguladora de genes).

2.2.1. Métodos para la inferencia de redes reguladoras de genes

Debido a que es un tema de investigación que ha atraído gran interés dentro de la comunidad científica son diferentes los enfoques mediante los cuales se ha tratado de inferir el comportamiento que muestran las interacciones entre los genes.
Entre los métodos computacionales que han sido adoptados para la inferencia de modelos de redes reguladoras de genes podemos encontrar:

- **Modelos Booleanos:**
  Es el más sencillo de los modelos, el cual fue propuesto originalmente por Kauffman [27]. Las redes booleanas están basadas en el supuesto de que interruptores binarios de encendido y apagado, en pasos discretos de tiempo, pueden describir aspectos importantes de la regulación de genes.

- **Redes de Petri:** Las redes de Petri han sido utilizadas con éxito para simular

las redes metabólicas. Las redes de Petri permiten realizar una representación cuantitativa de los procesos ocurridos en las redes reguladoras de genes.

- **Modelos basados en ecuaciones diferenciales:**
  Tanto las redes booleanas, como las redes de Petri pueden revelar propiedades importantes de las redes reguladoras de genes. Sin embargo, estos dos tipos de modelos son poco detallados para captar importantes aspectos de la dinámica de las redes. Los modelos basados en ecuaciones diferenciales permiten obtener descripciones más detalladas de la dinámica de las redes, modelando explícitamente los cambios de concentración de los niveles de expresión en el tiempo.

Debido a lo discutido anteriormente, en esta tesis se usará un modelo basado en ecuaciones diferenciales para representar redes reguladoras de genes.

Bases de datos	Descripción del contenido	URL
AceDB	AceDB es una base de datos originalmente desarrollada para el proyecto del genoma *C.elegans*	www.acedb.org
DDBJ	Base de datos de nucleótidos de Japón	www.ddbj.nig.ac.jp
EMBL-BANK	Base de datos de nucleótidos del Instituto Europeo de Bioinformática	http://www.ebi.ac.uk/embl/index.html
Entrez	Portal del NCBI que contiene una gran variedad de bases de datos biológicas	www.ncbi.nlm.nih.gov/Entrez/
ExPASY	Base de datos proteómica	http://us.expasy.org/
FlyBase	Base de datos que contiene el genoma de la mosca *Drosophila melanogaster*.	http://flybase.bio.indiana.edu/
PDB	Es la base de datos de estructura terciaria y secundaria de proteínas.	http://www.rcsb.org/pdb/
GENBANK	Es la base de datos de nucleótidos del NCBI.	http://www.ncbi.nlm.nih.gov/Genbank/
HIV databases	Son las bases de datos referentes al Virus de Inmunodeficiencia Humana.	http://www.hiv.lanl.gov/content/index
OMIM	Significa Online Mendelian Inheritance in Man. Es un catálogo de genes humanos relacionados con informaciones genéticas.	http://www.ncbi.nlm.nih.gov/omim/
HIV databases	Son las bases de datos referentes al Virus de Inmunodeficiencia Humana.	http://www.hiv.lanl.gov/content/index
ArrayExpress	Base de datos de experimentos de genómica funcional	http://www.ebi.ac.uk/arrayexpress/

Tabla 2.1: Principales bases de datos biológicas disponibles en Internet.

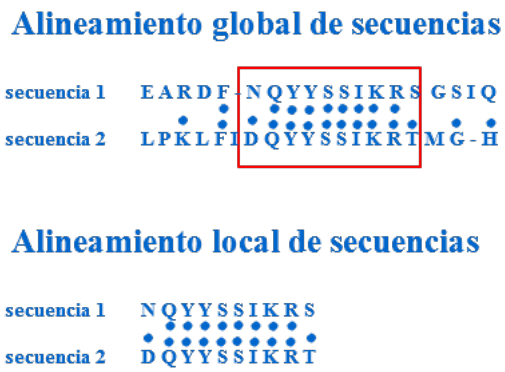

Figura 2.1: Un ejemplo de un emparejamientos de secuencias, donde se muestra la diferencia entre un alineamiento global y uno local; en el global se incluyen todos los residuos de las dos secuencias. La región con mayor semejanza está encerrada en el recuadro. El alineamiento local sólo incluye las porciones de las dos secuencias en las que se encontró una máxima semejanza. En la figura, los " : " significan que los residuos son idénticos y "." indica que los residuos son similares.

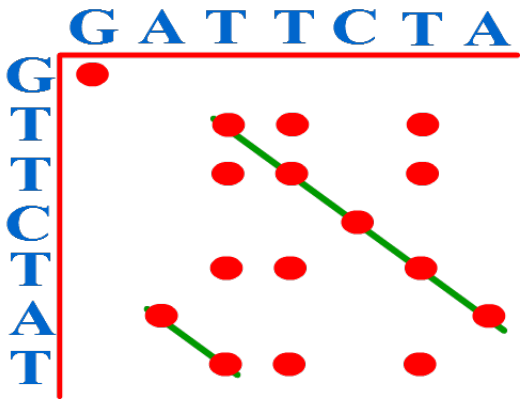

Figura 2.2: Ejemplo del método de la matriz de puntos .

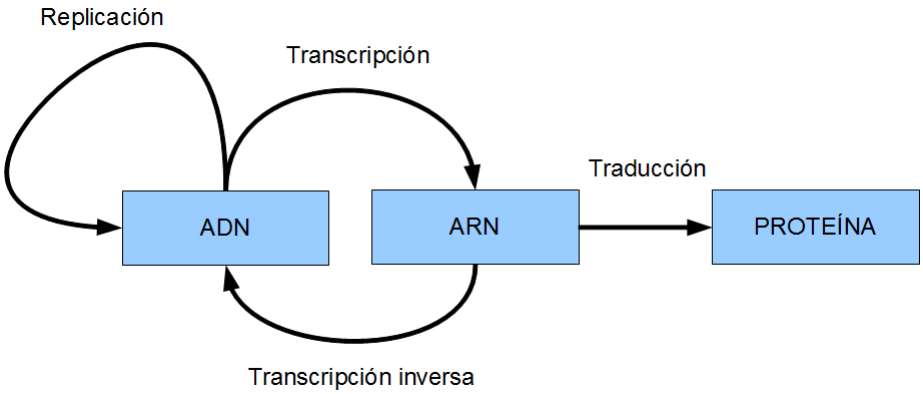

Figura 2.3: Dogma central de la biología molecular.

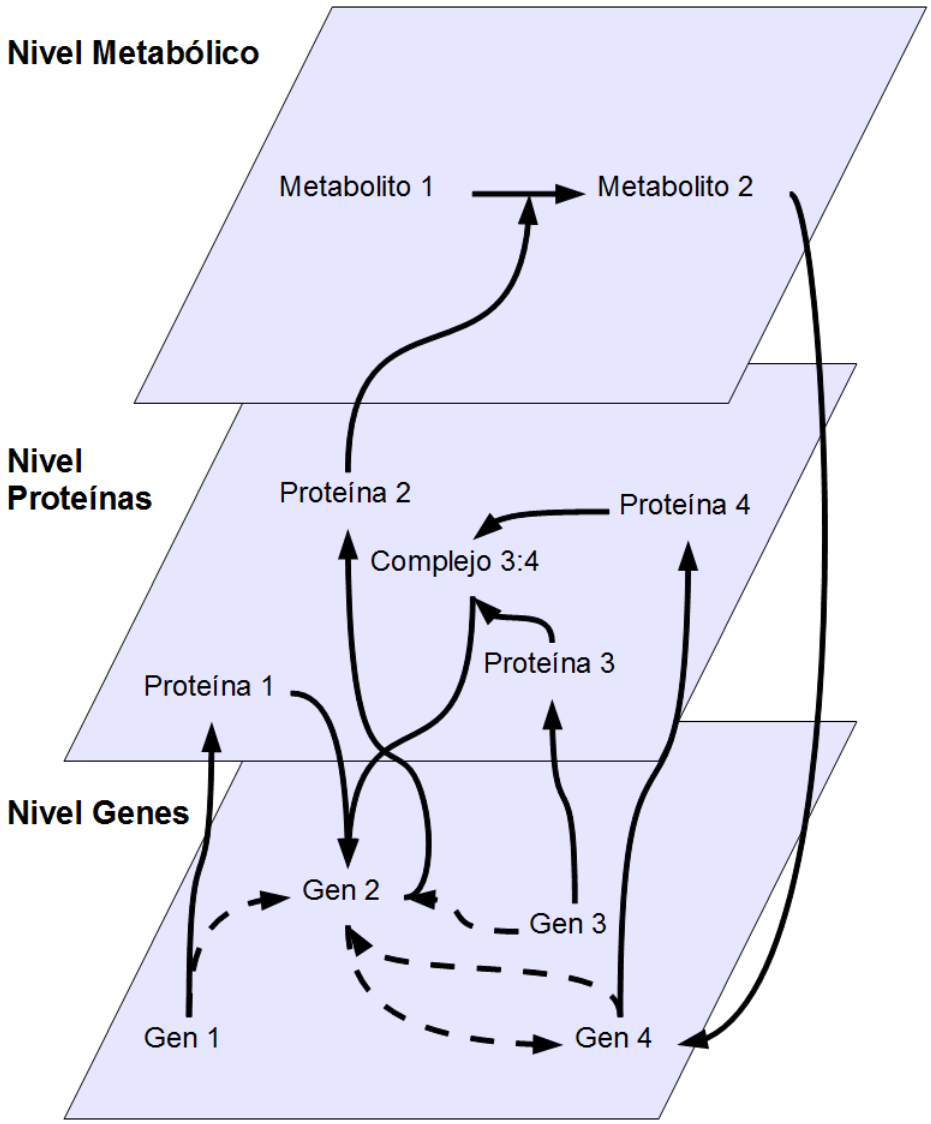

Figura 2.4: Representación gráfica de una red reguladora de genes.

Capítulo 3

Computación evolutiva

La computación evolutiva es una rama de la computación que usa conceptos de la teoría Neo-Darwiniana de la evolución natural para resolver problemas de optimización y clasificación. El Neo-Darwinismo extiende el concepto de selección natural (o supervivencia del más apto) de Charles Darwin con la teoría del germoplasma (que explica la transmisión de información hereditaria) de August Weissmann y con las leyes de la herencia (que originaron la genética) de Gregor Mendel [28]. El neo-Darwnismo establece que la vida en nuestro planeta es el resultado de un conjunto de procesos estocásticos que interactúan entre sí, generación tras generación. Estos procesos son la reproducción, mutación, competencia y selección [29] y son los responsables de toda la diversidad de vida de nuestro planeta.

3.1. Algoritmos evolutivos

Los algoritmos evolutivos (AEs) surgieron en la década de los 1960s, tratando de emular los procesos de la evolución biológica. De manera general, un AE realiza el siguiente proceso: se genera una población inicial, en la cual están codificadas las

posibles soluciones al problema. Para cada una de las soluciones se les evalúa la función objetivo y, posteriormente, se les asigna un valor de aptitud. Mediante algún método, se selecciona un subconjunto de la población que serán los padres de los individuos de la siguiente generación. A los padres se les aplica el operador de recombinación para generar nuevos individuos. Estos nuevos individuos son sometidos al operador de mutación, y se les evalúa su función objetivo con el fin de obtener su valor de aptitud. Estos nuevos individuos son evaluados en la función objetivo con el fin de obtener su aptitud. Después, se utiliza un mecanismo de selección que puede ser del tipo “ , ” o “ + ”. En el caso de la selección “ + ” los mejores individuos de la unión de padres e hijos sobreviven a la siguiente generación, mientras que en el caso de la selección “ , ” sólo los mejores hijos sobreviven a la siguiente generación sin competir con sus padres, que son siempre reemplazados. Este proceso se repite hasta que se cumpla con la condición de terminación del algoritmo [30].

El pseudocódigo de un AE se presenta en el algoritmo 5 y en el diagrama de flujo

de la figura 3.1 .

Algoritmo 5: Algoritmo evolutivo.

Entrada: N: Tamaño de la población, parámetros del AE

Salida: Población Final

$g \leftarrow 0$;

Generar la población inicial $P_g = \{\overrightarrow{x_1}, \overrightarrow{x_2}, ..., \overrightarrow{x_N}\}$;

Obtener la aptitud de cada individuo: $f\left(\overrightarrow{x_1}\right), f\left(\overrightarrow{x_2}\right), ..., f\left(\overrightarrow{x_N}\right)$;

while No se cumpla el criterio de término **do**

Seleccionar padres: $P_{padres} \subset P_g$;

Aplicar operador de recombinación : $P'_g \leftarrow \text{recombinar}(P_{padres})$;

Aplicar operador de mutación : $P''_g \leftarrow \text{mutar}(P'_g)$;

Obtener la aptitud de cada individuo en P''_g;

Obtener la nueva población: $P_{g+1} \leftarrow \text{seleccionar}(P''_g \cup P_g)$;

$g \leftarrow g + 1$;

end

3.1.1. Componentes de los algoritmos evolutivos

Para poder implementar un algoritmo evolutivo, los siguientes componentes deben ser especificados [30]:

- **Representación** (definición de los individuos):

  La representación de los individuos es el enlace entre el "mundo real" y los algoritmos evolutivos, ya que establece una relación entre las variables de decisión del problema a resolverse y el espacio de búsqueda donde opera el algoritmo evolutivo. En la terminología de computación evolutiva se llama *genotipo* a la forma en la que una solución es representada y *fenotipo* al valor de la solución en sí. Por tanto, la búsqueda en los algoritmos evolutivos se realiza en el espacio

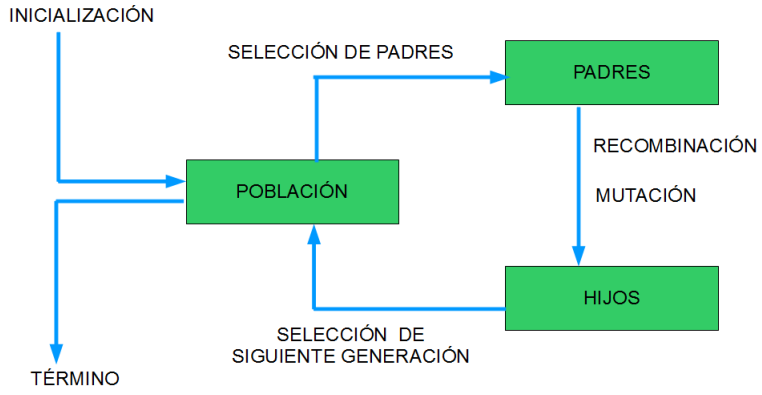

Figura 3.1: Diagrama de flujo de un algoritmo evolutivo.

genotípico y el cálculo de la función de aptitud se realiza en el espacio fenotípico. Tradicionalmente, la representación más utilizada en los algoritmos evolutivos es la binaria, por su universalidad. Sin embargo, muchas otras representaciones son posibles (p.ej., entera, árboles, matrices, etc.), destacando entre ellas la real, en la cual cada individuo consiste de un vector de números reales (en este caso, el espacio genotípico es igual al fenotípico).

- **Función de evaluación** (función de aptitud)

  La función de evaluación (comúnmente llamada función de aptitud) sirve para determinar la calidad de una solución. Es decir, nos indica qué tan buena es una solución con respecto a las demás. Todos los individuos de la población son evaluados con esta función, la cual sirve para guiar la búsqueda, pues en ella se basa el mecanismo de selección.

- **Población**

  La población representa a un conjunto de posibles soluciones al problema que queremos resolver. El tamaño de la población es un parámetro que define el usuario. Es importante que una población contenga soluciones diferentes (a eso se le denomina *diversidad*) a fin de muestrear más adecuadamente el espacio de búsqueda.

- **Mecanismo de selección de padres**

  El mecanismo de selección de padres es un proceso mediante el cual se distingue a los buenos individuos de los malos, y que individuos son seleccionados para generar una nueva población. Existen diferentes técnicas de selección de padres, aunque las técnicas más aceptadas son las probabilísticas en las cuales entre mayor sea la aptitud de un individuo mayor es su probabilidad de ser seleccionado como padre para la siguiente generación. Bajo este esquema, sin embargo, no se descarta la posibilidad de que un individuo de baja aptitud sea seleccionado, lo cual favorece la diversidad y puede evitar la convergencia prematura.[1]

- **Operadores de variación**

  El papel de los operadores de variación es el de generar nuevas soluciones o individuos a partir de individuos previos. Existen principalmente dos tipos de operadores que pueden diferenciarse en términos de su aridad, es decir, el número de individuos que el operador recibe de entrada.

  - **Operador de mutación:** El operador de variación unario llamado mutación, recibe un individuo y produce una variación estocástica en este. El

[1] La convergencia prematura ocurre cuando la población de un algoritmo evolutivo converge a una solución que no es la óptima y esta solución ya no puede mejorarse. La convergencia prematura ocurre normalmente tras haber transcurrido pocas generaciones.

papel principal del operador es garantizar que todo el espacio de búsqueda este conectado.

- **Operador de recombinación:** El operador de recombinación, o de cruza, mezcla dos individuos para generar uno o dos nuevos individuos. Al igual que la mutación, la recombinación es un proceso estocástico, en lo referente a qué partes de cada padre se combinarán para formar a los hijos correspondientes. El principio detrás de la recombinación es simple: se espera que de dos individuos que tienen buena aptitud, las características que los hacen ser buenos individuos se mezclen y produzcan un nuevo individuo mejor a sus padres.

- **Elitismo**

  Consiste en pasar intacto al mejor individuo de la población actual a la siguiente. Este mecanismo es necesario para garantizar convergencia [31].

- **Inicialización**

  En la mayoría de los algoritmos evolutivos, la inicialización, es decir, la generación de la primera población, se realiza de manera aleatoria. Sin embargo, para problemas particulares, la generación de la población inicial se puede realizar mediante el uso de algún otro método o heurística (p.ej., un algoritmo avaricioso [32]). Sin embargo, esta forma de generar los individuos iniciales es muy específica del problema y no puede ser aplicada de manera general. Por ello, la mayoría de las veces se opta por utilizar la generación aleatoria de individuos en la primera generación, buscando cubrir de manera uniforme el espacio de búsqueda.

- **Condición de paro**

  Debido a la naturaleza estocástica de los algoritmos evolutivos, no se puede ase-

gurar que siempre convergerán al óptimo, por lo cual pueden adoptarse distintos criterios de paro:

- **Tiempo máximo de ejecución:** El algoritmo detendrá el proceso de optimización cuando se alcance una cierta cantidad de tiempo (ciclos de reloj de una computadora)
- **Número máximo de evaluaciones de la función objetivo:** Si durante el proceso de búsqueda se ha alcanzado un número máximo de evaluaciones de la función objetivo (definido por el usuario).
- **Estancamiento de las soluciones:** Si durante una cierta cantidad de tiempo, el algoritmo ya no presenta mejora en la calidad de las soluciones.
- **Máximo número de generaciones:** Cuando el algoritmo ha alcanzado un número máximo de generaciones establecido inicialmente. Este es el mecanismo más utilizado en la práctica.
- **Diversidad de la población:** Se puede dar por terminada la búsqueda si la diversidad de la población cae dentro de un umbral. Esto es, debido a que, la población ha convergido a un valor y, dado que los individuos de la población son muy similares entre sí, los operadores de variación no pueden realizar modificaciones suficientes a los individuos para poder continuar con un proceso de búsqueda fructífero.

3.1.2. Paradigmas principales de los algoritmos evolutivos

Son tres los paradigmas principales dentro de los algoritmos evolutivos [33]:

- Estrategias evolutivas
- Algoritmos genéticos

- Programación evolutiva

Además, se han propuesto otra serie de heurísticas bioinspiradas que también son consideradas como algoritmos evolutivos :

- Algoritmos meméticos
- Optimización mediante cúmulos de partículas
- Evolución diferencial

En la sección 3.3, en la página 47, explicaremos a mayor detalle el funcionamiento de la evolución diferencial ya que fue el algoritmo evolutivo adoptado en esta tesis.

3.2. Optimización

Una aplicación muy popular de los algoritmos evolutivos es la optimización numérica. La optimización, en general, puede ser definida como el proceso mediante el cual se tiene como objetivo el encontrar el valor máximo o mínimo de una función en todo el espacio de búsqueda. La optimización es una rama de las matemáticas que tiene muchas aplicaciones en la vida real, sobre todo en diseño e ingeniería. De manera general, un problema de optimización consta de los siguientes componentes:

- **Variables de decisión:** Son los valores que pueden tener las variables para resolver el problema.
- **Función objetivo:** Es la expresión matemática que sirve para evaluar las variables de decisión con el fin de poder determinar si una solución es mejor que otra.
- **Restricciones:** Son las ecuaciones de igualdad o desigualdad que limitan los valores que las variables del problema pueden tomar.

Un problema de optimización se enuncia de la siguiente manera: Encontrar los valores de las n variables $[x_1, x_2, ..., x_n]$, denotadas por el vector $\vec{x}$, que optimicen (o sea, minimicen o maximicen) la función objetivo $f(\vec{x})$. Debido a que minimizar $f(\vec{x})$ es equivalente a maximizar $-f(\vec{x})$, el problema de optimización puede ser escrito siempre como uno de minimización:

$$\text{Encontrar } \vec{x} = [x_1, x_2, ..., x_n]^T \text{ que minimice } f(\vec{x}), \tag{3.1}$$

donde $\vec{x}$ es un vector n-dimensional llamado *vector de diseño,* $f(\vec{x})$ es la *función objetivo*

Los problemas de optimización han sido estudiados tradicionalmente en el área conocida como programación matemática y se han desarrollado diferentes métodos para resolverlos. Dentro de los métodos de programación matemática más conocidos se encuentran el método de Newton y el método de Fletcher-Reeves, entre muchos otros [34]. Estos métodos, sin embargo, suelen requerir que la función a optimizarse cumpla con ciertas características (p.ej., ser diferenciable), lo cual no siempre es posible. Otros métodos requieren un punto inicial de búsqueda y su desempeño suele depender mucho del mismo [34]. Las metaheurísticas, en contraste, requieren poca información específica del problema y son más generales.

3.3. Evolución diferencial

La evolución diferencial (ED) es uno de los paradigmas más recientes dentro de los algoritmos evolutivos. Originalmente se propuso como una heurística para la optimización de funciones no lineales y no diferenciables, desarrollada por Rainer Storn y Kenneth Price en 1995 [35, 36]. Su primera versión buscaba resolver el problema de ajuste de polinomios de Chebyshev.

La idea principal de la ED la perturbación de vectores mediante la operación de la diferencia entre dos variables, lo cual busca estimar el gradiente en una región. La evolución diferencial utiliza números reales para representar los valores de los vectores sobre el espacio de búsqueda. Este método se ha vuelto muy popular debido a su eficacia, robustez, facilidad de implementación y uso.

A continuación describimos los pasos del algoritmo de evolución diferencial: [37]

1. Inicialización:

   Primero, la población $P_{x,0}$ de N vectores D-dimensionales $\vec{x}_{i,0} = [x_{1,i,0}, ..., x_{D,i,0}]$, $i = 1, ..., N$ es generada aleatoriamente dentro de los límites $[x_{\text{mín}}, x_{\text{máx}}]$ como se muestra en la figura 3.2.

2. Mutación:

   Con respecto a cada vector $\vec{x}_{i,g}$ en la población actual, un vector $\vec{v}_{i,g}$ es generado como producto de la mutación. Esto se realiza agregando un vector de diferencia escalado, y seleccionado aleatoriamente un vector de la población actual. El vector diferenciado se genera de acuerdo a la ecuación ((3.2)) (ver figura 3.4).

   $$\vec{v_i}, g = \vec{x}_{r1,g} + F(\vec{x}_{r2,g} - \vec{x}_{r3,g}) \tag{3.2}$$

   donde F es un factor de escala de la mutación $F \in (0, 1+)$ e $i \neq r1 \neq r2 \neq r3$ donde $r1$, $r2$ y $r3$ son 3 individuos seleccionados aleatoriamente.

3. Recombinación:

   Con respecto a cada vector $\vec{x}_{i,g}$ de la población actual, un vector de prueba $\vec{u}_{i,g}$ es generado por la cruza del vector $\vec{x}_{i,g}$ con su vector $\vec{u}_{i,g}$ correspondiente bajo la probabilidad del porcentaje de cruza establecido en los parámetros del

algoritmo ($Cr \in [0, 1]$). Este procedimiento se define en la ecuación (3.3).

$$u_{j,i,g} = \begin{cases} v_{j,i,g} & \text{si } \text{rand}_j[0, 1] \leq Cr \text{ o } j = j_{rand} \\ x_{j,i,g} & \text{de lo contrario} \end{cases} \tag{3.3}$$

4. Selección y reemplazo:

   Si el vector de prueba $\vec{u}_{i,g}$ tiene mejor valor de la función objetivo que el de su vector $\vec{x}_{i,g}$ correspondiente, entonces este reemplaza al vector $\vec{x}_{i,g}$ para la siguiente generación. De lo contrario, el vector $\vec{x}_{i,g}$ permanece igual para la siguiente generación.

$$x_i, \vec{g} + 1 = \begin{cases} \vec{u}_{i,g} & \text{si } f(\vec{u}_{i,g}) \leq f(\vec{x}_{i,g}) \\ \vec{x}_{i,g} & \text{de lo contrario} \end{cases} \tag{3.4}$$

En el algoritmo 6 se muestra el algoritmo completo de la evolución diferencial.

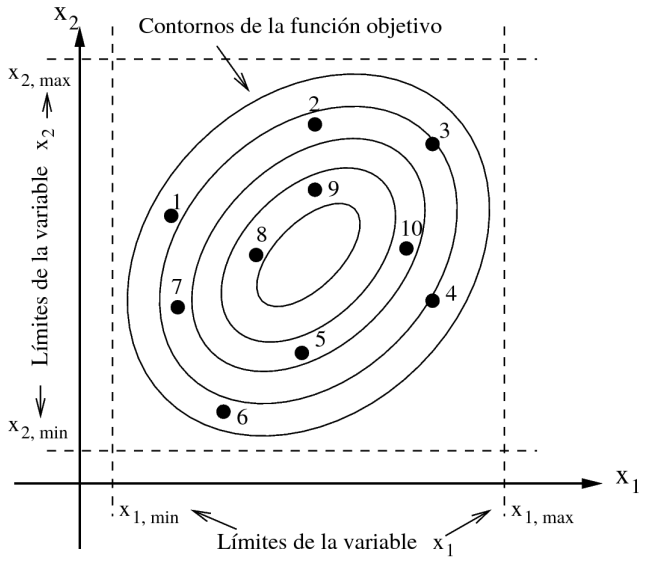

Figura 3.2: Ejemplo de la distribución de soluciones en la población inicial de la evolución diferencial.

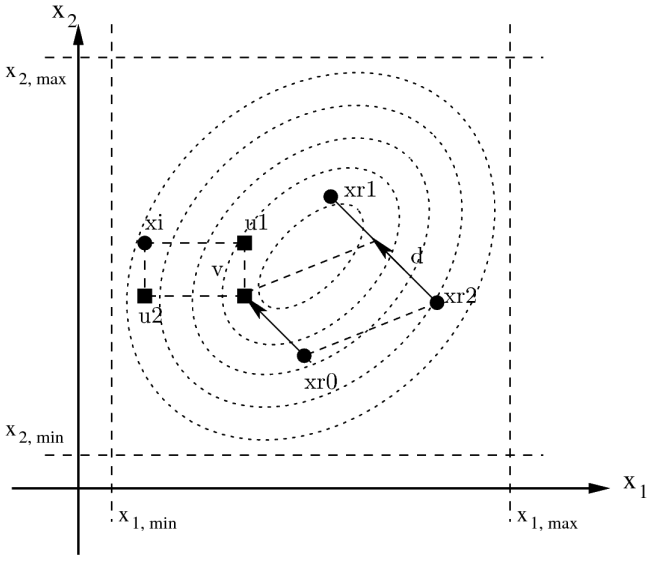

Figura 3.3: Representación gráfica de la recombinación en la evolución diferencial.

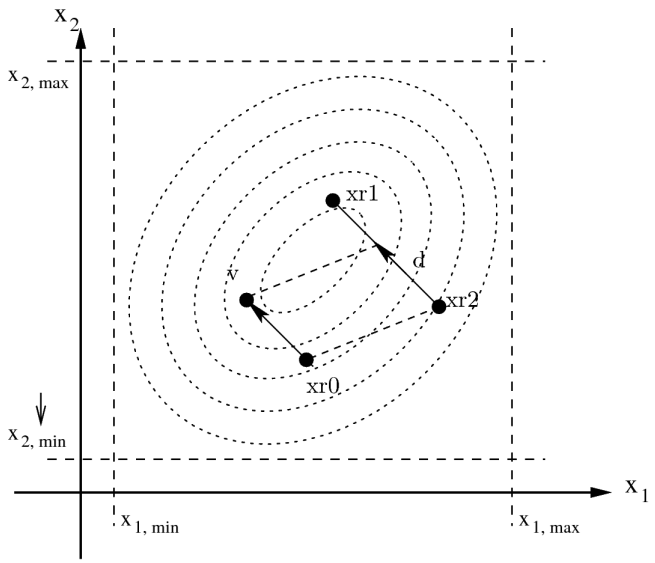

Figura 3.4: Representación gráfica de la mutación en la evolución diferencial.

Algoritmo 6: Algoritmo de la evolución diferencial.

Entrada: D: Número de variables que tiene el problema

N_p: Tamaño de la población

Cr : Probabilidad de Cruza

F : Constante de escalamiento

gmax: Número máximo de generaciones

Salida: Mejor solución encontrada

while No se cumpla el criterio de término*s* **do**

  for $i \leftarrow 0$ **to** N_p **do**

    $r0, r1, r2 \in [0, N_p - 1]$ seleccionados aleatoriamente

      donde $r0 \neq r1 \neq r2 \neq i$

    $j_{rand} \in [0, D-1]$ seleccionada al azar

    for $j \leftarrow 0$ **to** D **do**

      if $\text{rand}_j \leq Cr$ *ó* $j_{rand} = j$ **then**

        $u_{G,i,j} = x_{G,r0,j} + F(x_{G,r1,j} - x_{G,r2,j})$

      end

      else $u_{G,i,j} = x_{G,ri,j}$

    end

    if $f(\vec{U}_{G,i}) \leq f(\vec{X}_{G,i})$ **then**

      $\vec{X}_{G+1,i} = \vec{U}_{G,i,j}$

    end

    else $\vec{X}_{G+1,i} = \vec{X}_{G,i,j}$

  end

end

3.3.1. Variantes de la evolución diferencial

Existen diversos operadores de mutación para la evolución diferencial. Para clasificarlos, se utiliza la notación **DE/*x*/*y*/*z***, donde:

- *DE*: Hace referencia a evolución diferencial (*DE, Differential Evolution*)
- x: Representa el vector a perturbar; puede ser el mejor vector de la generación anterior (*best*) o cualquier vector elegido aleatoriamente (*rand*).
- y: Es el número de vectores considerados para la mutación del vector x. Para la perturbación de un simple vector *y=1*, se eligen aleatoriamente 3 vectores diferentes; la diferencia de dos de ellos se suma al tercero. Para perturbar dos vectores diferentes *y=2*, se eligen aleatoriamente 5 vectores distintos y la diferencia de pesos para cada par de cualquiera de los cuatro primeros se suma al quinto.

- z: Denota el esquema de cruza que se va a utilizar (exp: exponencial; bin:binomial). En la cruza exponencial, la primera vez que el valor aleatorio de la cruza supera al parámetro Cr, se suspende el ciclo y las variables restantes ya no se alteran. En la cruza binomial, la cruza se realiza siempre que el valor aleatorio sea menor al parámetro Cr

A continuación se enlistan las principales variantes de la evolución diferencial [38]:

1. DE/best/1/exp
2. DE/rand/1/exp

3. DE/rand-to-best/1/exp
4. DE/best/2/exp
5. DE/rand/2/exp
6. DE/best/1/bin
7. DE/best/1/bin
8. DE/rand-to-best/1/bin
9. DE/best/2/bin
10. DE/rand/2/bin

En esta tesis se utiliza el esquema de DE/rand/2/bin debido a que es sencillo de implementar y que es el esquema adoptado con más frecuencia en los trabajos relacionados que se revisaron para la elaboración de esta tesis.

Capítulo 4

GPGPU y estado del arte

4.1. GPGPU

En las últimas décadas ha habido un gran avance tecnológico, permitiendo integrar un mayor número de componentes electrónicos en un solo chip, posibilitando tener computadoras más veloces. Sin embargo, la velocidad de procesamiento alcanzada hasta el momento, no es suficiente para resolver problemas de gran tamaño como el de algunas aplicaciones científicas o comerciales. Y aunque la ley de Moore [39] establece que el número de transistores se duplicará aproximadamente cada 18 meses, los chips basados en silicio están llegando a su límite físico debido a los restricciones de las leyes electromagnéticas y termodinámicas. Para poder superar estas limitaciones, una solución viable es el uso de varios procesadores trabajando de manera coordinada en un solo problema, dividiendo las tareas que componen al problema entre éstos.

El cómputo paralelo consiste en el uso activo de varias unidades de procesamiento de manera simultánea para resolver un problema dado. Una tarea es dividida en múltiples subtareas usando la técnica de *divide y vencerás*, y cada una de ellas es

manejada en una unidad de procesamiento diferente.

Muchas aplicaciones actualmente requieren mucho mayor poder de cómputo del que una computadora secuencial es capaz de ofrecer. El cómputo paralelo ofrece la distribución del trabajo entre diferentes unidades de procesamiento, resultando en un mucho mayor poder de cómputo y rendimiento del que se puede obtener mediante un sistema tradicional de un solo procesador. El desarrollo del cómputo paralelo ha sido influenciado e impulsado por diferentes factores, de entre los que destacan los siguientes [40] :

- Los requerimientos de cómputo cada vez mayores, tanto en las aplicaciones de negocios, como en las aplicaciones científicas. Entre los problemas que requieren mucho mayor poder de cómputo se encuentran aquellos relacionados con las ciencias de la vida, aeroespaciales, sistemas de información geográfica, análisis y diseño de ingeniería, etc.

- Las limitaciones físicas de las arquitecturas secuenciales [41], debido a que aunque cada vez se reduce más el tamaño de los transistores para obtener una mayor capacidad de computo, la industria se esta acercando cada vez más al límite físico posible.

- La dificultad en mejorar el hardware de *pipelining*[1] y *superescalar*[2] además requiere de complicadas técnicas para el desarrollo de compiladores eficientes.

- El uso del cómputo paralelo se encuentra en un estado maduro y puede ser explotado comercialmente, debido a que ya existe un avance significativo en la

[1]Consiste en descomponer la ejecución de cada instrucción en varias etapas para procesar una instrucción diferente en cada etapa y trabajar con varias instrucciones a la vez.

[2]Es un tipo de micro arquitectura de procesador capaz de ejecutar más de una instrucción por ciclo de reloj.

	Una instrucción	**Múltiples instrucciones**
Un dato	SISD	MISD
Múltiples datos	SIMD	MIMD

Tabla 4.1: Clasificación de Flynn.

investigación y desarrollo de herramientas y ambientes para este fin.

- El avance significativo en la tecnología de redes de interconexión, el cual permite comunicaciones cada vez más rápidas.

4.2. Conceptos de cómputo paralelo

4.2.1. Clasificación

Existen diferentes clasificaciones de arquitecturas paralelas. Una de las más conocidas es la clasificación propuesta por Flynn [42] (ver tabla 4.1). Flynn clasifica las computadoras paralelas en cuatro tipos diferentes basándose en el número de instrucciones que se ejecutan y en el número de datos sobre los que se aplican estas instrucciones:

- **SISD** ***(Single Instruction, Single Data)***

  En este caso hablamos de una computadora secuencial en la que no hay paralelismo en las instrucciones, ni en el flujo de los datos. Un ejemplo son las computadoras tradicionales con un solo procesador (ver figura 4.1).

- **MISD** ***(Multiple Instruction, Single Data)***

  Esta arquitectura es poco común. En ella, múltiples instrucciones se aplican a un solo flujo de datos. Esta arquitectura se usa en situaciones de paralelismo redundante en donde se requieren sistemas tolerantes a fallas (ver figura 4.2).

- **SIMD *(Single Instruction, Multiple Data)***

  La arquitectura SIMD es cuando una misma instrucción es ejecutada sobre un conjunto de datos distintos. En esta arquitectura se cuenta con varios procesadores capaces de ejecutar la misma instrucción simultáneamente, sobre un segmento de datos que previamente fue asignado a cada procesador. Esta arquitectura es muy utilizada en las aplicaciones científicas debido a que en ellas suele requerirse un gran número de operaciones sobre matrices y vectores. Ejemplos de este tipo de arquitectura son las unidades de procesamiento gráfico (GPUs) o las máquinas vectoriales (ver figura 4.3).

- **MIMD *(Multiple Instruction, Multiple Data)***

  En las computadoras MIMD se realiza la ejecución simultánea de un grupo de instrucciones sobre distintos flujos de datos. Los sistemas distribuidos se ubican dentro de este tipo de arquitectura. A su vez, este tipo de arquitectura puede subdividirse en *MIMD de memoria compartida*, en el caso de que todos los procesadores puedan acceder a todas las regiones de la memoria disponible o *MIMD de memoria distribuida* en el caso de que cada procesador posea su propia región de memoria y no pueda acceder directamente a los datos contenidos en la memoria de otro procesador (ver figura 4.4).

En las últimas décadas han sido propuestos muchos modelos y lenguajes de programación para realizar implementaciones de algoritmos paralelos [43].

Dentro de los lenguajes más extendidos y usados podemos encontrar MPI (*Message Passing Interface*), para cómputo escalable en clusters y OpenMP (*Open Multi-Processing*), para sistemas de memoria compartida. MPI adopta un modelo consistente en un cluster de computadoras donde los nodos no comparten memoria, y en el cual tanto el intercambio, como la interacción de los datos entre los nodos debe realizarse explícitamente a través del uso de paso de mensajes. MPI ha sido exitoso

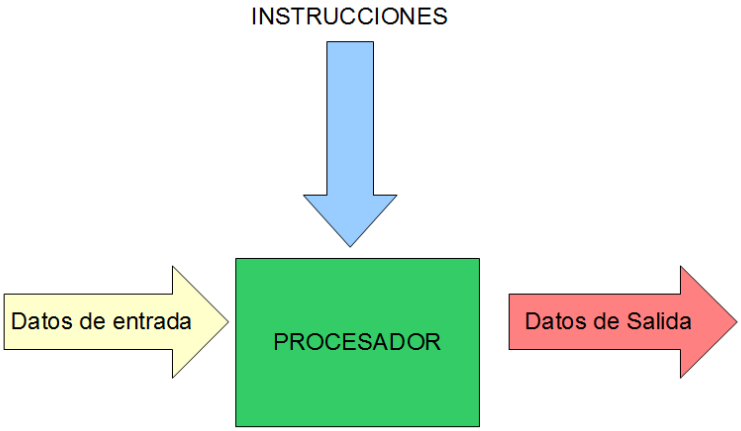

Figura 4.1: Clasificación de Flynn - SISD.

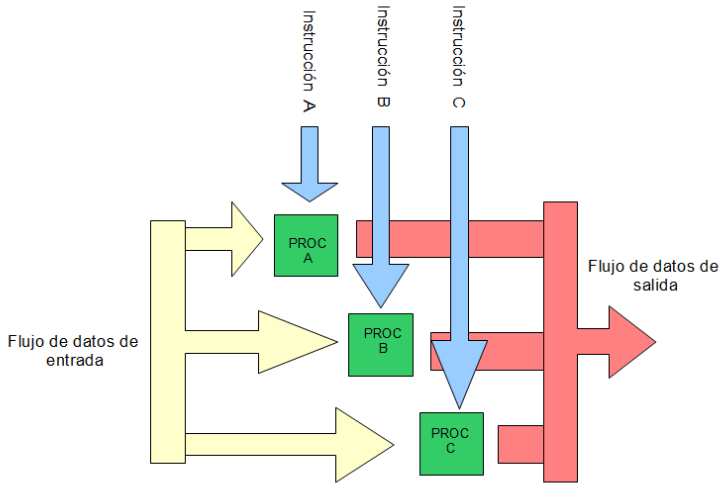

Figura 4.2: Clasificación de Flynn - MISD.

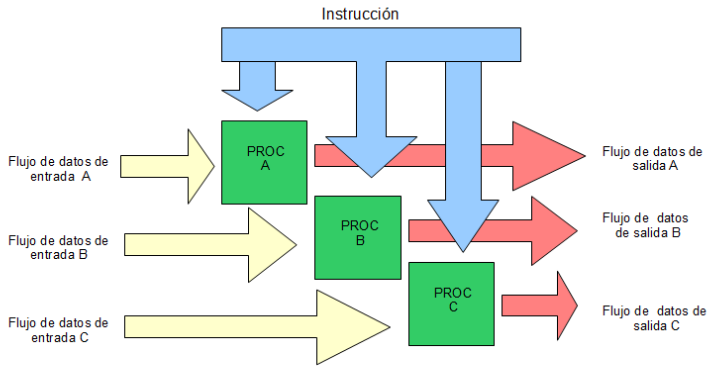

Figura 4.3: Clasificación de Flynn - SIMD.

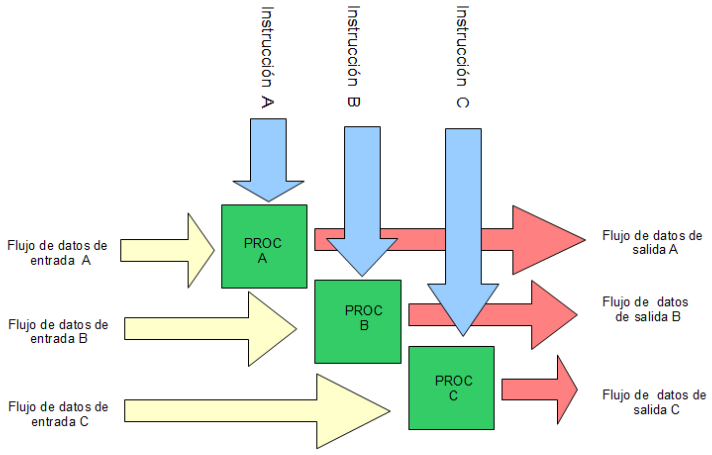

Figura 4.4: Clasificación de Flynn - MIMD.

en el cómputo de alto rendimiento, particularmente en aplicaciones científicas implementadas en clusters. Por otro lado, OpenMP es una API[3] de programación para cómputo paralelo de memoria compartida, en la que se agregan directivas al lenguaje de programación [44].

4.2.2. Granularidad

La granularidad se refiere al coeficiente entre el tiempo para una operación de comunicación básica entre una operación de cómputo [45]. Esto es, el tamaño relativo de las unidades de cómputo que son ejecutadas al mismo tiempo. La granularidad se mide de acuerdo al grosor o fineza con que se dividen las tareas en un sistema de computo paralelo. Principalmente se clasifican en dos tipos de granularidad:

- Grano fino .- Se trabajo a nivel de funciones, ciclos o instrucciones, en los que se requiere un alto grado de interacción y comunicación entre las unidades de procesamiento. Por lo general, consisten de una gran cantidad de procesadores simples y relativamente lentos.

- Grano grueso .- En el caso de grano grueso una aplicación o grupo de procesos o aplicaciones, se dividen en procesos o tareas practicamente independientes que se ejecutan al mismo tiempo y la comunicación entre las unidades de procesamiento es menor. Por lo general, consisten de pocos procesadores, aunque con mayor capacidad de computo que los de grano fino.

Los GPUs son unidades de procesamiento de grano fino, ya que, se conforman de una gran cantidad de unidades de procesamiento, aunque, con relativamente poco

[3] *Application Programming Interface* (API): Es un conjunto de bibliotecas de programación mediante las cuales se puede acceder a servicios y representan un método de abstracción en la programación a la hora de comunicar procesos.

poder de cómputo (comparado con los procesadores del CPU), y además, los GPUs, requieren un alto grado de coordinación en el cómputo paralelo.

4.3. GPU

Una unidad de procesamiento para gráficos o GPU por sus siglas en inglés (*graphics processing unit*) es un procesador que inicialmente estaba destinado exclusivamente a implementar en hardware directivas para desplegar gráficos en una computadora. Sin embargo, actualmente es posible desarrollar cómputo de propósito general o GPGPU (*General-Purpose Computation on Graphics Hardware*) usando GPUs. Los GPUs se consideran una arquitectura SIMD en la clasificación de Flynn vista anteriormente.

4.4. Historia de los GPUs

Entre finales de la década de los 1980s y principios de la década de los 1990s, el crecimiento de la popularidad de los sistemas operativos con interfaces gráficas avanzadas (principalmente *Microsoft Windows*) ayudó a crear un mercado para un nuevo tipo de procesador: los aceleradores de despliegue de gráficos en dos dimensiones para computadoras personales. Estos dispositivos proporcionaban visualización asistida por hardware que realizaba operaciones de mapas de bits para facilitar el despliegue de ventanas y demás facilidades gráficas requeridas por los nuevos sistemas operativos.

En la misma época, la empresa *Silicon Graphics* popularizó el uso de gráficos tridimensionales en mercados tales como aplicaciones para el gobierno y la defensa y visualización científica.

Dicha empresa también proporcionó técnicas para la creación de efectos visuales para uso cinematográfico. En 1992, Silicon Graphics abrió la interfaz de programación de su hardware mediante la liberación de la biblioteca OpenGL. La idea era que

OpenGL se usaría como el estándar para programar aplicaciones gráficas interactivas tridimensionales, independientemente de la plataforma de hardware adoptada.

Posteriormente, a mediados de los 1990s, la demanda de aplicaciones que requerían el uso de gráficos en 3D aumentó rápidamente, preparando el escenario para desarrollos significativos. Primero, se dio la salida de juegos inmersivos de primera persona como *DOOM*, *Duke Nukem* y *Quake*, lo cual motivó una búsqueda de entornos tridimensionales más realistas para los juegos de computadora. Por ende, fue este desarrollo tan acelerado de la industria de los videojuegos lo que ayudó a adoptar más rápidamente los gráficos 3D en las computadoras personales. Al mismo tiempo, empresas como *MATROX*, *ATI* y *NVIDIA*, entre otras, comenzaron a desarrollar aceleradores gráficos que fueron lo suficientemente económicos como para poder atraer al usuario común.

La salida de la tarjeta gráfica NVIDIA *GeForce 256* en el año de 1999 empujó aún más las capacidades del hardware gráfico de consumo. Por primera vez [46], se podía realizar directamente en el procesador gráfico el cómputo necesario para efectuar las tareas de transformación e iluminación[4], mejorando de este modo el potencial de aplicaciones visualmente más interesantes, dado que, la transformación y la iluminación eran partes integrales de las operaciones de renderización[5] de OpenGL. La GeForce 256 marcó el inicio de una progresión natural en la que se incrementaría el número de operaciones de renderización realizados directamente en los procesadores gráficos.

Desde el punto de vista del cómputo paralelo, la liberación de la tarjeta gráfica GeForce 3 por parte de NVIDIA en el 2001 representó uno de los avances más importantes de la tecnología de GPUs, ya que fue el primer chip de la industria en

[4]La transformación se refiere a la tarea de conversión de coordenadas en el espacio 3D virtual a la pantalla 2D. La iluminación se refiere a simular efectos de luz sobre los gráficos desplegados.

[5]La renderización es el proceso por el cual se generan imágenes a partir de modelos matemáticos.

implementar DirectX 8.0 [40]. DirectX es un estándar de programación desarrollado por Microsoft para facilitar la programación de aplicaciones multimedia. Esta versión requería que el hardware compatible fuera capaz de realizar tanto la programación del *pixel shading*[6] como la programación de los vectores. Por primera vez, los desarrolladores de aplicaciones tenían cierto control sobre los cálculos que se realizaban dentro del GPU. La salida de GPUs que poseían operaciones programables atrajo la atención de investigadores para usar hardware gráfico de tal manera que problemas generales parecieran problemas de renderización para el GPU.

Los GPUs de principios de este siglo fueron diseñados para producir un color por cada pixel de la pantalla usando unidades aritméticas programables conocidas como *pixel shaders*. Estos usan las coordenadas de posición(x,y) de la pantalla y otra información de entrada para producir un color de salida. La otra información podrían ser los colores de entrada, coordenadas de textura, u otros atributos que serían pasados al *shader* cuando se ejecutara. Pero como las operaciones realizadas a los colores y texturas de entrada eran completamente controladas por el programador, los investigadores observaron que estos "colores de entrada" podría ser, en realidad, cualquier tipo de dato.

De tal forma que si la entrada eran en realidad datos numéricos que significaban algo más que color, se podrían entonces programar los *pixel shaders* para realizar cálculos arbitrarios sobre los datos. Los resultados, serían entregados de nuevo al GPU como el "color resultado" del pixel, de tal forma que este "color" sería el resultado de los cálculos encargados al GPU, los cuales podrían ser leídos por el programador. En esencia, el GPU estaba siendo "engañado" para realizar tareas que no eran de renderización como si en realidad lo fueran. Esta técnica parecía muy creativa, aunque

[6]El *pixel shade*r es un programa de sombreado que sirve para manipular un pixel pudiendo aplicar diferentes efectos sobre una imagen (por ejemplo sombras y exposiciones a la luz).

también era muy complicada.

Debido al alto rendimiento de procesamiento aritmético de los GPUs, los primeros resultados de estos experimentos preveían un futuro promisorio al cómputo de propósito general mediante GPUs. Sin embargo, el modelo de programación seguía siendo demasiado restrictivo para un gran número de posibles aplicaciones, ya que existían fuertes restricciones en los recursos disponibles, debido a que los datos de entrada estaban limitados a un número determinado de colores y texturas. Hubo también serias limitaciones sobre dónde y cómo el programador podía escribir los resultados en la memoria, de tal forma que algunos algoritmos en los que se requería escribir el resultado en alguna posición específica de la memoria (p.ej., operación de *scatter*) no podrían ser implementados en los GPUs. Por otra parte, era casi imposible predecir el manejo del GPU para datos de punto flotante, por lo que la mayoría de las aplicaciones científicas no podían hacer uso del GPU. Finalmente, cuando el programa no terminaba de una forma correcta o simplemente no terminaba, no existía un método razonablemente bueno para realizar la depuración del código que se estaba ejecutando en el GPU.

Otro obstáculo para hacer más aceptable la utilización de GPUs para cómputo de propósito general, era que la única manera de interactuar con el GPU era mediante el uso de funciones de OpenGL o DirectX, por lo que el programador tendría que aprender estas bibliotecas sujetándose a las restricciones de estos lenguajes para poder realizar cualquier tipo de cómputo.

4.5. CUDA

En noviembre de 2006, NVIDIA dio a conocer su tarjeta gráfica GeForce 8800 GTX, que fue la primera GPU que soportaba DirectX 10. La GeForce 8800 GTX fue

también la primera GPU en ser construída con la arquitectura CUDA de Nvidia. Esta arquitectura incluyó nuevos componentes diseñados exclusivamente para el cómputo con GPU y con el objetivo de resolver las limitaciones que impedían a los procesadores gráficos anteriores ser útiles para el cómputo de propósito general.

4.6. Arquitectura de los GPUs

A diferencia de generaciones anteriores en las que se repartieron los recursos de computación en *vertex* y *pixel shaders*, la arquitectura CUDA incluye un shader *pipeline* unificado, lo que permite que cada una de las unidades aritméticas y lógicas (*ALUs* por sus sigas en inglés) pueda ser transformada a través un programa de manera que pueda realizar cómputo de propósito general. Debido a que NVIDIA pretendió que esta nueva familia de procesadores gráficos fuera utilizada para cómputo de propósito general, cada una de las ALUs fue construida cumpliendo los requisitos de la IEEE para aritmética de punto flotante de precisión simple y fueron diseñados para usar un conjunto de instrucciones para cómputo de propósito general y no específicamente para los gráficos. Además, las unidades de ejecución del GPU pueden acceder arbitrariamente a la lectura y escritura de la memoria, así como a un *caché* conocido como memoria compartida.

En la figura 4.5 se muestra la arquitectura típica de un GPU moderno [40]. Está organizado en un arreglo de multiprocesadores de alto flujo de hilos *(highly threaded streaming multiprocessors)* (*SMs*). En la figura 4.5, los SMs forman un bloque constructor (*building bloc*k). El número de bloques constructores varía de una generación a otra de GPUs CUDA. A su vez, cada SM en la figura tiene un número propio de procesadores de flujo (*SP o Streamming Processors*) que comparten instrucciones lógicas e instrucciones de cache. Cada GPU actualmente cuenta con hasta 4 Gigabytes

de (GDDR) DRAMs que es llamada memoria global. Estas GDDR DRAMs son diferentes a las DRAMs que tenemos en las tarjetas madres del CPU en el sentido de que son esencialmente para la memoria del *buffer* que se utiliza en los gráficos. Para aplicaciones gráficas, los *SMs* poseen imágenes de video, e información de renderización para textura de tres dimensiones. Para aplicaciones de paralelismo masivo, el mayor ancho de banda compensa, la latencia[7] necesaria.

La G80 que introdujo la arquitectura CUDA tiene un ancho de banda de memoria de 86.4 GB/s. Una aplicación de CUDA puede transferir datos desde la memoria del sistema con una tasa de 4 GB/s y la misma tasa para regresar los datos a la memoria del sistema. En total, hay una tasa combinada de transferencia de 8 GB/s. El ancho de banda de comunicación es mucho más lento que el ancho de banda. Sin embargo, el ancho de banda del *PCI Express* (puerto mediante el cual se encuentra conectado el GPU al sistema) es comparable con el ancho de banda del *front-side bus*[8] a la memoria del sistema. De tal forma que el ancho de banda de comunicación en realidad no es una limitante. También se espera que el ancho de banda crezca en el futuro a medida que el tamaño de la memoria del sistema vaya creciendo.

El chip de paralelismo masivo G80 tiene 128 SPs (16 SMs, cada uno con 8 SPs) Con 128 SPs, se obtiene un total de 500 gigaflops. Además, las unidades especiales permiten realizar operaciones de punto flotante como la raíz cuadrada, así como otras funciones trascendentes. Con 240 SPs, la tarjeta gráfica GT200 excede el teraflop de velocidad de procesamiento. Debido a que cada SP puede dividirse masivamente en hilos, cada SP puede ejecutar miles de hilos por aplicación. Una chip típicamente ejecuta entre 5000 y 12000 hilos simultáneamente. Haciendo una comparación con los procesadores tradicionales actuales de Intel que soportan de 2 a 8 hilos simultáneos de grano grueso, el chip G80 soporta hasta 769 hilos por SM, lo que permite hasta un

[7]Se denomina latencia a los retardos ocurridos en el acceso a la memoria.
[8]Bus principal de comunicación entre el CPU y la memoria del sistema

total de 12000 hilos simultáneos de grano fino para este chip. Una tarjeta más actual como la GT200 soporta 1024 hilos por SM y hasta cerca de 30000 hilos simultáneos. Resulta evidente que el nivel de paralelismo que soporta un GPU está aumentando rápidamente. Es muy importante por eso tratar de alcanzar los niveles ideales de paralelismo de un GPU al realizar una aplicación, a fin de aprovechar de una mejor manera la capacidad de procesamiento del GPU.

4.7. Uso de CUDA

Para alcanzar un mayor número de posibles desarrolladores, NVIDIA decidió adoptar el lenguaje de programación C al cual agregó un número relativamente pequeño de instrucciones. Pocos meses después del lanzamiento de la GeForce 8800 GTX, NVIDIA hizo público el compilador para este lenguaje llamado CUDA C. De esta manera, CUDA C se convirtió en el primer lenguaje diseñado por una compañía de GPUs para facilitar el cómputo de propósito general en los GPUs. Ademas de crear el lenguaje CUDA C, NVIDIA también provee un controlador de hardware especializado para explotar la arquitectura de NVIDIA.

Desde su debut en el 2007, una gran variedad de industrias han desarrollado aplicaciones en CUDA C, logrando beneficios que regularmente representan órdenes de magnitud de mejora en cuanto al rendimiento, comparados con las implementaciones del estado del arte previamente utilizadas. Algunas de las áreas que se han beneficiado del uso de CUDA son las siguientes:

- Aceleración en la renderización de gráficos 3D [47]
- Aceleración en la conversión entre formatos de video [48]

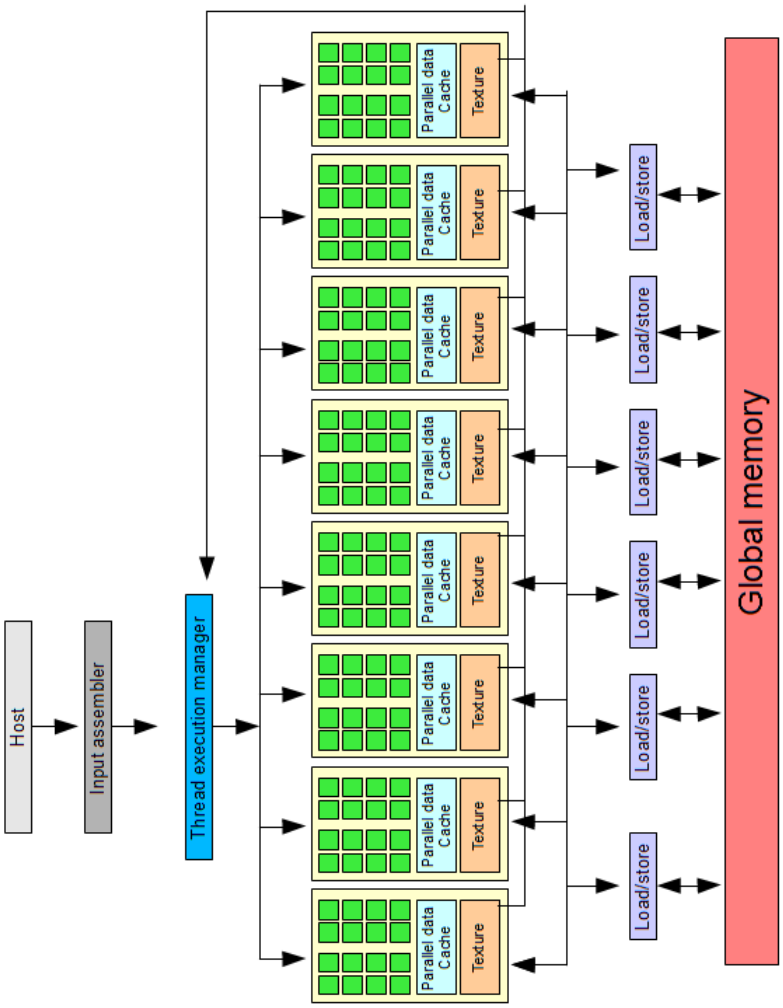

Figura 4.5: Arquitectura de GPU CUDA.

Rank	Nombre	País	Cluster de GPUs	Max TFlops
1	K computer	Japón	No	8162.00
2	Tianhe-1A	China	Si	2566.00
3	Jaguar	Estados Unidos	No	1759.00
4	Nebulae	China	Si	1271.00
5	Tsubame 2.0	Japón	Si	1192.00

Tabla 4.2: Las 5 supercomputadoras con mayor poder de cómputo a junio de 2011.

- Aceleración en el cifrado, descifrado y compresión de datos [49].
- Simulaciones de análisis médicos, por ejemplo, realidad virtual basada en imágenes de resonancia magnética [50].
- Simulaciones físicas de dinámica de partículas para fluidos [51]
- Alineamientos de secuencias de ADN [52]

Otro dato importante sobre el impacto que el cómputo de propósito general con GPUs está teniendo en el cómputo de alto rendimiento es que actualmente 3 de las 5 computadoras con mayor capacidad de cómputo en el mundo son clusters con GPUs [53] como se puede ver en la tabla 4.2.

Debido a que el problema que trata de resolver esta tesis es muy costoso computacionalmente, se plantea la alternativa de utilizar computo de alto rendimiento mediante GPUs.

En el apéndice B se mencionan los requerimientos minimos para poder desarrollar aplicaciones con NVIDIA CUDA.

4.8. Estado del arte

Debido a que este es un trabajo multidisciplinario se debe de revisar qué es lo que se ha realizado en las diferentes áreas que están involucradas. Principalmente, se revisará trabajo previo que se ha realizado en relación a algoritmos evolutivos paralelos y a la inferencia de modelos de redes reguladoras de genes usando algoritmos evolutivos.

4.8.1. Paralelización de algoritmos evolutivos

Los algoritmos evolutivos pueden presentar básicamente tres tipos de paralelización [54]:

- **Evaluación paralela de la población**

  Debido al carácter poblacional de los algoritmos evolutivos, y a que la evaluación de un individuo es independiente a la de otro, se puede realizar un tipo de paralelización en la que cada unidad de procesamiento sea capaz de evaluar a un individuo diferente. Con ello, se obtiene naturalmente una mejora en la aceleración de la ejecución total del algoritmo. Este tipo de paralelización es la más sencilla e intuitiva.

- **Evaluación distribuida de una solución**

  Puede ocurrir que en un algoritmo evolutivo, la función de evaluación sea muy costosa computacionalmente por lo que, otra alternativa es paralelizar las evaluaciones de los individuos. Este tipo de paralelización es altamente dependiente de la función de evaluación ya que la paralelización es particular para cada problema. Puede ocurrir que la evaluación de la solución no sea paralelizable y en esos casos este tipo de paralelización no puede ser utilizado.

- **El modelo de islas**

El modelo de islas es el más extendido dentro de los algoritmos evolutivos. Consiste en dividir a la población en subpoblaciones de menor tamaño, dejándolas evolucionar de manera independiente como si se tratara de un algoritmo evolutivo independiente. Después de que ciertas condiciones sean satisfechas, los individuos de las subpoblaciones son mezclados con el fin de lograr una búsqueda cooperativa que evite, hasta cierto punto, una convergencia prematura. Sin embargo, este esquema requiere que se especifiquen una serie de parámetros:

- *Topología de intercambio*: La topología especifica con qué islas es posible realizar un intercambio.
- *Número de migrantes*: Este parámetro indica cuántos individuos son intercambiados.
- *Política de selección de migrantes*: Este parámetro indica la forma en que serán seleccionados los individuos que migrarán (p.ej., los mejores).
- *Política de integración o reemplazo*: Este parámetro indica qué va a ocurrir con los nuevos individuos que se incorporan de otras poblaciones. Se pueden manejar criterios tales como solo aceptarlos si son individuos mejores que los existentes en la población, aceptarlos si son diferentes a la población, eliminar a todos los individuos inferiores al nuevo elemento, etc.
- *Criterio de migración*: Este parámetro establece cuando ocurre la migración. Por ejemplo, se realizará de manera periódica, de manera determinista o tal vez cuando ocurran ciertas condiciones dadas.

4.8.2. Cómputo evolutivo en GPUs

El uso de GPUs para cómputo evolutivo ha empezado a gozar de cierta popularidad debido a que los algoritmos evolutivos son altamente paralelizables en modelos

SIMD. A continuación, se presentan algunos trabajos realizados con AEs, utilizando GPUs. Deepak [55] propone el uso de GPUs para realizar una jerarquización estocástica para resolver problemas de optimización multiobjetivo. Dentro de los resultados que muestra, menciona haber conseguido una aceleración de 5000x para poblaciones muy grandes (10000 individuos). Dentro de otros trabajos relacionados con el cómputo evolutivo y los GPUs, Thé Van Luong [54] propone un esquema de islas para GPUs que parece ser muy eficiente ya que el factor de aceleramiento crece proporcionalmente al número de islas que se consideran. De este trabajo se infiere que el modelo de islas parece funcionar muy bien sobre GPUs.

Evolución diferencial sobre GPUs

Veronese [56] realizó la primera implementación de un algoritmo de evolución diferencial sobre CUDA. Su trabajo es muy intuitivo y consiste en paralelizar la evaluación de los individuos de tal forma que lo que realiza es asignar un hilo por individuo en la población para que ésta se evalue de forma simultánea. En el trabajo muestra las funciones en CUDA para realizar esta paralelización, y se indica que deben realizarse los cambios de datos de la memoria del dispositivo (tarjeta gráfica) a la memoria del procesador (computadora donde se encuentra instalada la tarjeta gráfica), de tal forma que las evaluaciones se realicen en los núcleos del GPU. En las pruebas que se presentan (funciones de prueba de minimización estándar) en el trabajo se obtuvo una aceleración de procesamiento de 10-35.5; esto es, se ejecutó 35.5 veces más rápido la versión paralela sobre GPUs que la versión secuencial.

Esto representa una aceleración muy significativa que antes sólo era posible mediante grandes clusters de computadoras.

Algoritmos evolutivos para la inferencia de modelos de redes reguladoras de genes

Los sistemas de ecuaciones diferenciales son una forma de modelar el comportamiento de los genes y, en este caso, se utiliza un tipo de sistema de ecuaciones diferenciales en particular: los *sistemas S*.

Los *sistemas S* son sistemas de ecuaciones diferenciales que pueden ser expresados de la siguiente forma:

$$\frac{dXi(t)}{d(t)} = \alpha_i \prod_{j=1}^{N} X_j(t)^{g_{ij}} - \beta_i \prod_{j=1}^{N} X_j(t)^{h_{ij}}, \tag{4.1}$$

donde N es el número de genes que están involucrados en la red. Los términos g_{ij} y h_{ij} representan el grado de interacción de X_j sobre X_i. El primer término representa todas las influencias que incrementan X_i, mientras que el segundo término representa todas las influencias que disminuyen X_i. Los parámetros α_i y β_j son valores no negativos llamados constantes de escalamiento y los parámetros g_{ij} y h_{ij} son valores reales llamados órdenes cinéticos.

Estos modelos han sido utilizados con frecuencia debido a que pueden representar muy bien el dinamismo del sistema. Sin embargo, el problema con ellos es el gran número de parámetros que tienen que ser calculados, $2N(N+1)$. Debido a que este número crece de manera cuadrática, el problema se vuelve más complejo conforme crece el número de genes a inferir. Otra característica deseable en la inferencia del sistema, es que los valores de los parámetros g_{ij} y h_{ij} sean cero para diferentes valores de i y j, ya que eso significa que el sistema no se encuentra altamente conectado, y, por tanto, las redes son poco densas.

El objetivo consiste en encontrar los valores para un conjunto de parámetros que puedan producir los mismos valores obtenidos de manera experimental (datos reales). Esto es equivalente a un problema de optimización donde se buscan los valores tales que minimicen el error cuadrático medio entre los datos obtenidos del experimento biológico y los generados por el sistema S.

Para poder realizar la optimización mediante algoritmos evolutivos, en un individuo se codifican los parámetros α_i, β_j, g_{ij} y h_{ij} que representan a un *Sistema S*. De tal forma, cada individuo de la población representa un sistema de ecuaciones diferenciales que se resuelve al evaluar al individuo. Esto se puede realizar con un método de Runge Kutta, aunque existen también otros métodos númericos que pueden usarse para el mismo fin [57]. En el apendice A se muestran los metodos utilizados en esta tesis.

Una vez que se tiene resuelto el sistema, el individuo es evaluado y se le compara con los datos reales. Regularmente, se utiliza la siguiente expresión para determinar la aptitud del individuo:

$$f = \sum_{i=1}^{N} \sum_{t=1}^{T} \left(\frac{X_{i,ca,t} - X_{i,exp,t}}{X_{i,exp,t}} \right)^2 \tag{4.2}$$

donde N es el número de genes involucrados, T es el número de mediciones, $X_{i,ca,t}$ representa los valores calculados por el sistema generado para X_i, y $X_{i,exp,t}$ representa los valores obtenidos por el experimento biológico para X_i.
Entonces, el problema consiste en minimizar el valor de la ecuación 4.2 de tal forma que el sistema de ecuaciones diferenciales puedan definir de una manera más precisa la naturaleza de la red.

Existe cierto trabajo realizado con respecto a la inferencia de redes reguladoras de genes usando algoritmos evolutivos [58, 59, 60, 61, 62, 63, 64, 65].

Se han utilizado diferentes heurísticas tales como: algoritmos genéticos con representación binaria [58], estrategias evolutivas [66], algoritmos genéticos con representación real [67] y evolución diferencial [68, 65, 62].

También se han adoptado enfoques multiobjetivo [69, 63, 60, 70, 61, 71] y se han combinado algoritmos evolutivos con algunas otras técnicas tales como redes neuronales [72] y buscadores locales [59, 62, 73, 64].

Todos estos trabajos siguen el mismo esquema principal que consiste en el cálculo de parámetros de los sistemas S, de tal forma que se reduzca el error entre el sistema descrito por la solución y los datos reales. En algunos trabajos se involucra no sólo el error cuadrático medio sino que también se busca que no existan muchas conexiones entre los genes ya que se sabe que las redes reguladoras de genes son sistemas dispersos, por lo que algunos autores también involucran esta característica en la función de evaluación [2, 74, 65, 62].
Otros autores han adoptado enfoques multiobjetivo en los que buscan, mediante un objetivo, disminuir el error entre los datos generados y los datos reales y en el otro objetivo buscan que la red sea dispersa [61]. Dentro de la bibliografía revisada hasta el momento la evolución diferencial parece ser la heurística más prometedora para este problema ya que en las comparaciones reportadas por la literatura resulta ser la técnica más efectiva [75, 62].
En cuanto al conjunto de datos de prueba, la mayoría de los trabajos usan datos independientes. Sin embargo, hay algunos trabajos que recuperan los datos de trabajos anteriores [75]. Es con estos datos con los que se va a trabajar en esta tesis. Son pocos

los autores que reportan haber usado datos de genes reales.

Capítulo 5

Propuesta de un AE basado en GPUs para la inferencia de modelos de GRN

En este capítulo se presenta la descripción del algoritmo propuesto en esta tesis. Primero se muestra un ejemplo ilustrativo del proceso requerido para evaluar la aptitud de un individuo. Posteriormente se muestra como se paralelizo el algoritmo, y, en la parte final de este capítulo se describe a detalle como se implemento el algoritmo en el GPU.

5.1. Evaluación de la aptitud para la inferencia de modelos de GRN mediante AEs

En esta sección se ilustrará mediante un pequeño ejemplo el proceso que se debe seguir para evaluar la aptitud de un individuo o calidad de un conjunto de parametros de un modelo S para la inferencia de modelos de redes reguladoras de genes:

1. Supongamos que los datos de la tabla 5.1 fueron obtenidos de los microarreglos ADN, es decir los niveles de expresión de dos genes $X_1(t)$ y $X_2(t)$ para tres

t	$X_1(t)$	$X_2(t)$
0	0.7	0.3
0.4	1.7258	0.9566
0.8	1.5934	0.6153

Tabla 5.1: Datos de ejemplo.

instantes de tiempo diferentes. El objetivo es usar un sistema S para modelar la interacción existente entre estos dos genes

2. Posteriormente, se genera un individuo aleatoriamente, es decir un conjunto de parámetros que describan un sistema S. Supongamos que se generó el siguiente individuo con los parámetros mostrados a continuación:

$$G = \begin{bmatrix} 0 & 2.5 \\ -2.5 & 0 \end{bmatrix} \qquad H = \begin{bmatrix} -1 & 0 \\ 0 & 2 \end{bmatrix} \qquad \alpha = \begin{bmatrix} 3 \\ 3 \end{bmatrix} \qquad \beta = \begin{bmatrix} 3 \\ 3 \end{bmatrix} \tag{5.1}$$

3. Con este conjunto de valores podemos representar el sistema S que corresponda a ese individuo sustituyendo los valores de los parámetros de la ecuación (4.1). De tal forma, obtenemos el siguiente sistema de ecuaciones diferenciales:

$$\begin{aligned} \frac{dX_1(t)}{d(t)} &= 3X_1(t)^0 X_2(t)^{2.5} - 3X_1(t)^{-1} X_2(t)^0 \\ \frac{dX_2(t)}{d(t)} &= 3X_1(t)^{-2.5} X_2(t)^0 - 3X_1(t)^0 X_2(t)^2 \end{aligned} \tag{5.2}$$

$$\begin{aligned} \frac{dX_1(t)}{d(t)} &= 3X_2(t)^{2.5} - 3X_1(t)^{-1} \\ \frac{dX_2(t)}{d(t)} &= 3X_1(t)^{-2.5} - 3X_2(t)^2 \end{aligned} \tag{5.3}$$

4. Después, se resuelve el sistema de ecuaciones diferenciales mediante un método numérico, por ejemplo, el método de Runge Kutta. Supongamos que al resolver el sistema obtenemos la siguiente solución:

t	$X_1(t)$	$X_2(t)$
0	0.7	0.3
0.4	2.3	1.5
0.8	1.7	0.5

5. Al resolver el sistema de ecuaciones diferenciales, se obtiene la aptitud del individuo. Esto se hace cuantificando el error cuadrático medio entre la solución del sistema y los datos objetivo de la tabla 5.1. Para eso utilizamos la ecuación 4.2 (en la pag. 75).

6. En caso de que el valor de la aptitud fuera cero entonces se habría encontrado el modelo que puede reproducir el comportamiento de la red reguladora de genes. Por lo tanto, este problema se puede ver como uno de minimización donde se busca minimizar el error cuadrático medio.

7. De esta manera, hemos obtenido la aptitud para un individuo. Este proceso se debe realizar con cada individuo de la población. Posteriormente, se ejecutan los pasos restantes de un AE.

5.2. Paralelización del algoritmo

Como se vio en el capitulo 4 los algoritmos evolutivos se han paralelizado utilizando paralelismo de grano fino (cuando sólo la evaluación de la aptitud de los individuos es paralelizada) o mediante paralelismo de grano grueso (cuando se utiliza el modelo de islas). Sin embargo en esta tesis,se pretende paralelizar en el GPU el algoritmo per se, y no solo paralelizar la evaluación de los individuos. Ya que en un modelo tradicional, las operacions del AE (cruza, mutación y selección), no se realizan de manera paralela, y es sólo la evaluación de la población la que se busca paralelizar.

Lo que se busca al paralelizar todo el algoritmo en el GPU, es reducir la cantidad de datos que es transferida entre en GPU y el CPU, ya que, siguiendo el modelo tradicional, en cada generación al momento de realizar la evaluación de los individuos de la población, se tendría que transferir toda la información de cada uno de los individuos al GPU, evaluar al individuo, y despues transferir la aptitud de cada individuo del GPU al CPU. y proceder con el resto de los operadores del AE.

Además, al paralelizar los operadores se busca obtener una ganancia adicional, ya que, aunque aparentemente no son procesos computacionalmente muy costosos, y como son independientes entre sí, si podría obtenerse una ganancia al realizar estas operaciones en paralelo.

Para validar esta hipotesís se realizó el siguiente experimento:

Se implementó un algoritmo en el que sólo se paraleliza la evaluación de los individuos de la población y otro en el que se paraleliza todos los operadores del AE en el GPU. Se realizaron 15 ejecuciones para cada conjunto de parametros que se

Número de individuos	Número máximo de generaciones	Tiempo promedio de ejecución algoritmo 1	Tiempo promedio de ejecución algoritmo 2
2048	1000	6.34s	9.49 s
2048	2000	9.44s	18.83 s
2048	3000	10.32s	28.27 s
4096	1000	7.12s	16.29 s
4096	2000	10.24s	32.48 s
4096	3000	12.21s	48.64 s

Tabla 5.2: En la tabla en algoritmo 1, se paralelizan todos los operadores del AE, mientras que en el algoritmo 2, sólo se parelliza la evaluación de los individuos

muestran en la tabla 5.2 y se reportan los tiempos de ejecución promedio. Como se puede observar paralelizar todo el algoritmo representa una ganacia significativa con respecto a sólo paralelizar la evaluación de los individuos

5.3. Implementación en CUDA

A continuación se describe como se llevo a acabo la implementación del algoritmo propuesto en esta tesis.

Se desarrolló una implementación en GPUs de la evolución diferencial que resuelve el problema de la inferencia de modelos de redes reguladoras de genes. En el diagrama de flujo de la figura 5.1 se muestran los procesos que se realizan en nuestra propuesta. Cada uno de estos procesos se describirán a detalle a continuación.

5.3.1. Reserva de memoria

Antes de comenzar con los procesos, se decidió adoptar la técnica de reservar el espacio en memoria que se ocupará en el GPU, a fin de minimizar la transferencia de datos entre el CPU y el GPU, ya que la transferencia de datos es la principal

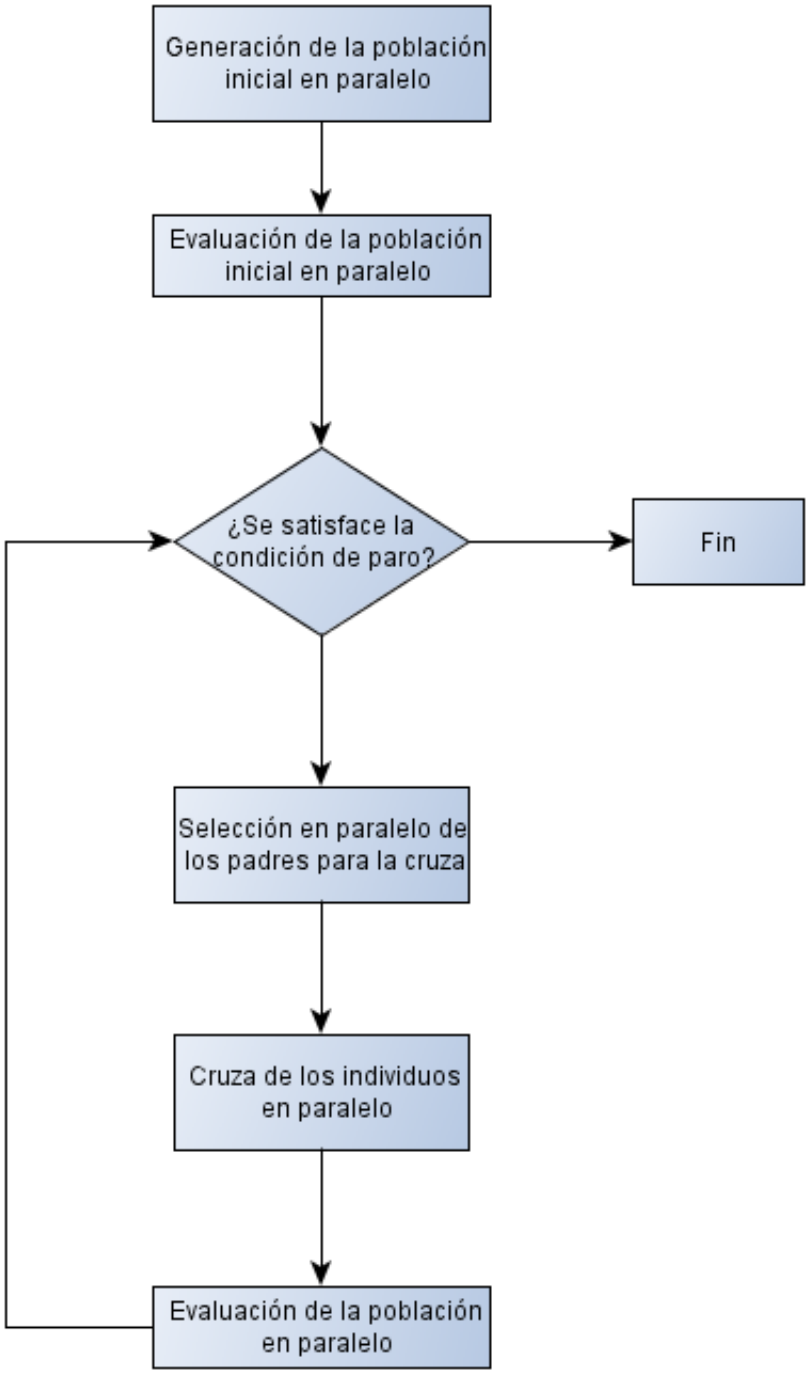

Figura 5.1: Diagrama de flujo de la propuesta de la tesis.

causa de que ocurra un "cuello de botella" en las aplicaciones en GPUs, ocasionando en consecuencia, un bajo rendimiento. La memoria que se reserva es en la región de la memoria global del GPU, para que los datos se mantengan durante toda la ejecución del programa. Los datos se almacenan de manera continua, de tal forma que todos los datos de cada individuo se encuentran contiguos en la memoria. La figura 5.2 muestra una representación gráfica de cómo se reserva la memoria al inicio de la ejecución del programa. El código CUDA necesario para reservar memoria se puede ver en el código 5.1. La función `cudaMalloc` se encuentra dentro de la librería estándar de CUDA y sirve para reservar espacio en memoria en el GPU. Su prototipo es el siguiente `cudaError_t cudaMalloc ( void ** devPtr,size_t size)` donde `devPtr` es el apuntador a la región de memoria en el GPU y `size` es el tamaño requerido en bytes. La función puede regresar un `cudaSuccess` en caso de que la función haya terminado de manera exitosa o `cudaErrorMemoryAllocation` en caso de que ocurra algún error. En el código 5.1 se reserva el espacio necesario para la población. Para cada individuo se reserva espacio para sus parámetros, para la matriz H, para la matriz G, para el vector α, para el vector β, para la aptitud del individuo y para guardar la solución del sistema S de cada individuo.

Código 5.1: Reservación de la memoria.

```
cudaMalloc((void**)&Nd_pop_G, sizeof(float)*ngenes*ngenes*POPSIZE);
cudaMalloc((void**)&Nd_pop_H, sizeof(float)*ngenes*ngenes*POPSIZE);
cudaMalloc((void**)&Nd_pop_alph, sizeof(float)*ngenes*POPSIZE);
cudaMalloc((void**)&Nd_pop_beta, sizeof(float)*ngenes*POPSIZE);
cudaMalloc((void**)&Nd_pop_fitness, sizeof(float)*POPSIZE);
cudaMalloc((void**)&Nd_pop_X, sizeof(float)*ngenes*npoints*POPSIZE);
```

5.3.2. Generación de la población inicial en paralelo

La población inicial se genera en paralelo directamente en el GPU mediante la librería CURAND, de reciente incorporación en CUDA. CURAND posee varias funciones para la generación de números aleatorios con diferentes características. Para la

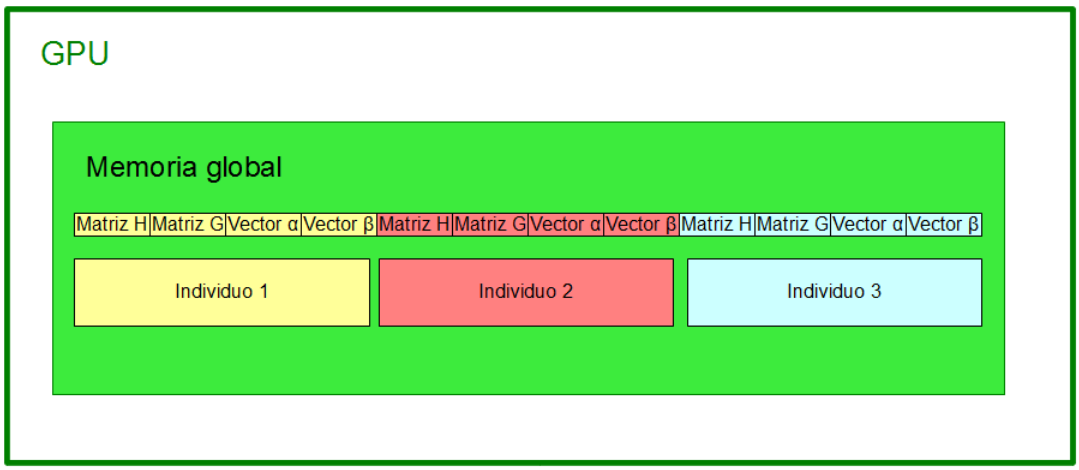

Figura 5.2: Reservación de memoria en el GPU. Cada individuo guarda los datos correspondientes a las matrices G y H y alos vectores α y β.

generación de los individuos se utilizó la función:

`curandGenerateUniform( curandGenerator_t generator, float *outputPtr, size_t num) .`

Esta función genera números de punto flotante entre 0.0 y 1.0 uniformemente distribuidos. En los argumentos de la función encontramos `curandGenerator_t generator` el cual recibe un algoritmo disponible para la generación de números aleatorios (en esta tesis se utilizó el algoritmo por defecto de la biblioteca "XORWOW"). También recibe el argumento `outputPtr`, que es un apuntador al espacio en memoria en que se escribirá el resultado de la generación de números aleatorios y `num` que indica el tamaño en bytes de los números aleatorios que serán generados.

Código 5.2: Generación de números aleatorios.

```
curandGenerateUniform(gen,popG,ngenes*ngenes*POPSIZE);
curandGenerateUniform(gen,popH,ngenes*ngenes*POPSIZE);
curandGenerateUniform(gen,popalpha,ngenes*POPSIZE);
curandGenerateUniform(gen,popbeta,ngenes*POPSIZE);
```

Una vez realizada la generación de los números aleatorios, es necesario ajustar los valores dentro de los límites del espacio de búsqueda para cada una de las variables.

Para realizar esto en paralelo lanzamos un *kernel*.

Un **kernel** es como se le denomina a una función que se ejecuta en el GPU y se define como **___global___ void fun()**, donde **fun** es el nombre del kernel. Para invocar un kernel, se debe de indicar cuántos hilos de ejecución serán utilizados por éste. Los hilos de ejecución del GPU son agrupados mediante bloques. Cada bloque de hilos posee características tales como compartir memoria, permitir sincronización entre hilos, todo esto dentro del mismo bloque de hilos (no se puede realizar comunicación entre hilos de diferentes bloques, así como tampoco se permite la sincronización entre hilos pertenecientes a diferentes bloques). Para invocar un kernel se realiza de la siguiente manera: **fun<< j,k >>()** , donde j, indica el número de bloques que utiliza el kernel y k, el número de hilos por bloque que se utiliza en el kernel. El GPU utilizado para esta tesis permite hasta 1024 hilos por bloque, y un máximo de 65535 bloques, por lo que se pueden invocar 65535×1024 hilos en un kernel.

En cada cada hilo de ejecución del kernel realizará el ajuste de los valores entre los límites del espacio de búsqueda. Esto se realiza como se muestra en el código 5.3.

Código 5.3: Ajuste de valores dentro del espacio de búsqueda.

```
fit_vals<<<grid_size,block_size>>>(popG, mingij, maxgij);
fit_vals<<<grid_size,block_size>>>(popH, minhij, maxhij);
fit_vals<<<grid_size,block_size>>>(popalpha, minalph, maxalph);
fit_vals<<<grid_size,block_size>>>(popbeta, minbeta, maxbeta);
```

De esta manera, realizamos en paralelo el ajuste de valores invocando un kernel por variable en el que cada variable es ajustada por un hilo. Esta es la forma mediante la cual se genera la población inicial dentro del espacio de memoria que fue reservado previamente.

5.3.3. Evaluación de la población en paralelo

Se implementó un esquema de evaluación de la población en paralelo. Para este fin, se asignó a cada individuo un hilo de ejecución del GPU de tal forma que se tienen tantos hilos de ejecución como individuos en la población. Cada hilo realiza la evaluación de aptitud de un individuo, esto es, dentro de cada hilo se resuelve un sistema de ecuaciones diferenciales mediante el método de Runge Kutta de cuarto orden. Cada hilo toma la región correspondiente a un individuo dentro de la memoria global del GPU, donde se encuentra la población y copia los valores correspondientes del individuo a la memoria local del hilo. De esta forma es más rápido acceder a la memoria, obteniéndose beneficios de rendimiento. Cada región copiada contiene los parámetros que corresponden al sistema S a resolverse. Posteriormente, una vez resuelto el sistema, se calcula el error de la solución del sistema comparado con los datos objetivo para así obtener la aptitud de cada individuo. En el código 5.4 se muestra cómo se invoca al kernel que realiza la evaluación en paralelo.

Código 5.4: Evaluación de los individuos de la población.

```
llamark<<<(unsigned int)(POPSIZE/512) + 1,512>>>
(d_pop_t, d_pop_G, d_pop_H, d_pop_alph, d_pop_beta, d_pop_X, dB, d_pop_w,
d_pop_k, npoints, ngenes, h, d_pop_fitness);
```

5.3.4. Operadores de la evolución diferencial en paralelo

Se implementaron los operadores de la evolución diferencial de tal forma que se calcularán en paralelo:

Selección de individuos para la cruza.

La selección de los individuos que formarán parte de la cruza se realiza de manera paralela de la siguiente forma. Debido a que los 4 individuos seleccionados de-

ben de ser seleccionados aleatoriamente y diferentes entre sí, se utiliza la función `currandGenerate` de la librería CURAND, la cual genera un conjunto de números enteros aleatorios que representan los índices de los individuos en la población. Sin embargo, se debe comprobar que estos números sean diferentes entre sí.

Cruza paralela

Una vez que se han seleccionado n conjuntos de índices de la población, donde cada conjunto representa una cruza a realizar, cada conjunto de números es asignado a un hilo, para que la operación de cruza se realice de forma simultánea en cada hilo de ejecución.

5.3.5. Selección en paralelo

Finalmente, una vez que se ha realizado la cruza se comparará si el nuevo individuo es mejor que el individuo padre. Si ese es el caso, entonces, es reemplazado. Una vez más, esto puede ser realizado de manera paralela asignando una comparación por hilo de ejecución. Esto se realiza como se muestra en el código 5.5

Código 5.5: Selección de individuos en paralelo.

```
tid = threadIdx.x + (blockIdx.x * blockDim.x);
        if(nuevo_ind.fitns[tid]<viejo_ind.fitns[tid])
                viejo_ind = nuevo_ind
```

Las variables `threadIdx.x, blockIdx.x y blockDim.x` son reservadas por CUDA y sirven para identificar cada hilo.

Este proceso se repite hasta que la condición de paro se alcance. La condición de paro puede ser un número máximo de generaciones o un valor de aptitud deseable que es cuando el error de los datos obtenido es cercano a cero.

Capítulo 6

Diseño Experimental y Análisis de Resultados

En este capítulo se detallarán los resultados obtenidos de la implementación realizada para un conjunto de genes que se ha utilizado anteriormente en la literatura relacionada.

En la primera parte de este capítulo se describirá la infraestructura con la que se contó para la realización de esta tesis, así como el equipo con el cual fueron validados los resultados. Posteriormente, se establecerá el conjunto de instancias de prueba usadas para realizar la validación de la implementación desarrollada. Después, se presentarán los resultados obtenidos analizándose tanto la calidad de las soluciones obtenidas como el tiempo requerido para hallarlas, a fin de estimar las aceleraciones logradas.

GPU utlilizado	
Modelo	GeForce GTX 460
Versión del controlador NVIDIA utilizado	4.0
Cuda Capability	2.1
Cantidad total de memoria global:	1023 MBytes (1072889856 bytes)
Número de núcleos CUDA:	336 CUDA Cores
Velocidad de reloj del GPU	1.53 GHz
Velocidad de reloj de memoria:	1850.00 Mhz
Ancho del bus de memoria:	256 bits
Tamaño del cache L2:	524288 bytes
Cantidad total de memoria constante:	65536 bytes
Cantidad total de memoria compartida por bloque:	49152 bytes
Número total de registros por bloque:	32768
Máximo número de hilos por bloque:	1024
Tamaño máximo de las dimensiones del bloque:	$1024 \times 1024 \times 64$
Tamaños máximo de las dimensiones del grid:	$65535 \times 65535 \times 65535$
Soporta copiado y ejecución concurrente:	Si
Soporta ejecución concurrente de kernels:	Si

Tabla 6.1: Especificación técnica del GPU utilizado para la realización de la tesis.

6.1. Infraestructura

El GPU con el que se contó para la realización de esta tesis es una tarjeta GeForce GTX 460 cuyas características, se enlistan en la tabla 6.1.

Para el caso de la versión secuencial se utilizó el procesador de la computadora donde se encuentra alojada la tarjeta gráfica. El equipo al que se hace referencia cuenta con las especificaciones técnicas mostradas en la tabla 6.2.

6.2. Diseño experimental

La instancia mediante la cual se probó la implementación realizada ha sido utilizada previamente en la literatura relacionada [75, 2]. La instancia corresponde al

CPU utlilizado	
Modelo del procesador	Intel Core 2 Quad Q8400
Velocidad del procesador	2.66 Ghz
Total de memoria RAM disponible	2 GB de memoria RAM

Tabla 6.2: Especificación técnica del CPU utilizado para la realización de la tesis.

Variable	Intervalo de búsqueda
α_i	[0,20.0]
β_i	[0,20.0]
g_{ij}	[-4.0,4.0]
h_{ij}	[-4.0,4.0]

Tabla 6.3: Intervalo de búsqueda de los parámetros del sistema S.

conjunto de datos de dos genes generados artificialmente por Tominaga [2]. El conjunto de datos se muestra en la tabla 6.4 y consiste en un conjunto de 50 niveles de expresión de dos genes. La representación gráfica del conjunto de datos se puede observar en la figura 6.1.
El intervalo de búsqueda de las variables se encuentra definido en [2] y son los que se muestran en la tabla 6.3.

Para los parámetros del algoritmo de la evolución diferencial se utilizaron los valores de: $F = 0.5$ y porcentaje de cruza $Cr = 0.8$.

6.3. Validación de la implementación

Para validar la correctitud de nuestra implementación se ejecutaron varias pruebas en las que, como se puede observar en las figuras 6.2 y 6.3, se pudieron reproducir los valores de los niveles de expresión de los genes en su totalidad. En el caso de las imágenes, el valor de aptitud de la solución es de 0.0162×10^{-5}.

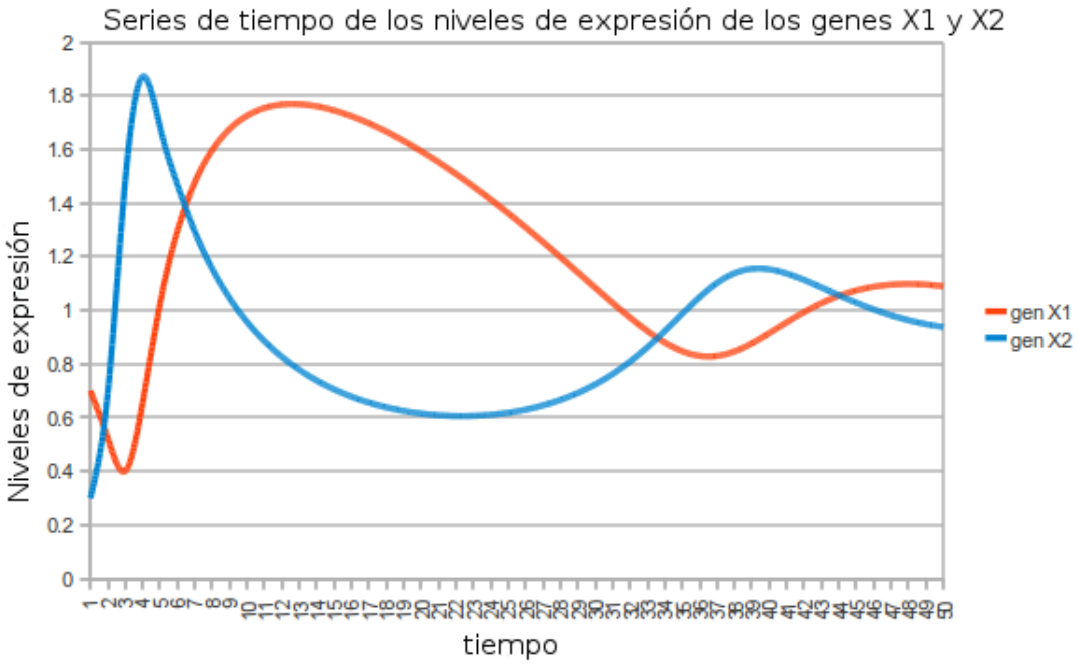

Figura 6.1: Instancia de pruebas Tominaga.

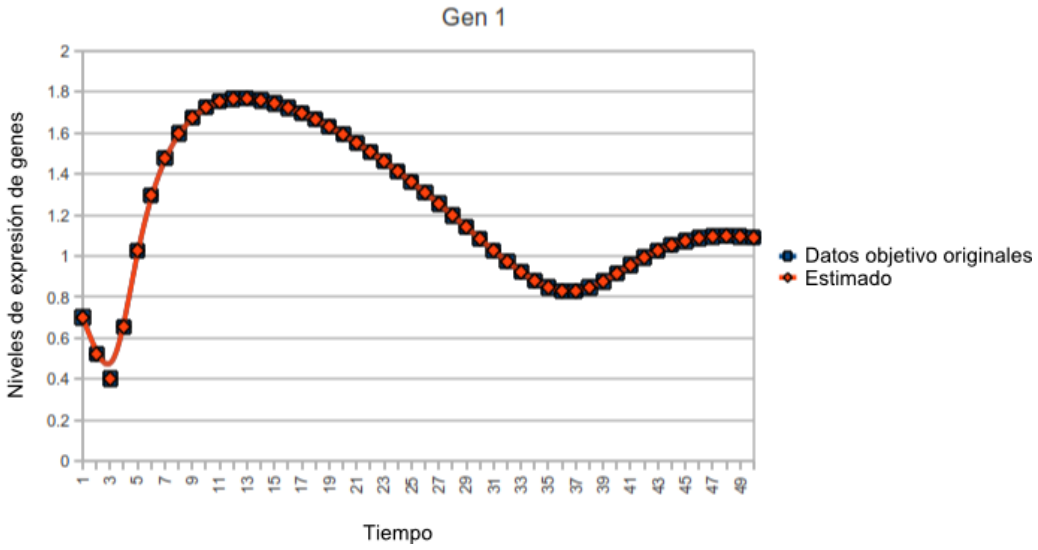

Figura 6.2: Datos objetivo y estimados para el gen 1.

Instancia Tominaga	
0.7	0.3
0.5234791780272952	0.6918046028294869
0.4008077213301025	1.482555609372431
0.6555485472242668	1.871422957275465
1.0257875494081694	1.6827720724709134
1.2960679904234484	1.4682349488962445
1.477598487755887	1.2930385999270209
1.5978645197220018	1.1546483753242396
1.6763976296003589	1.0448119036425079
1.7258282263986326	0.9566453423432874
1.7543438531221716	0.885085805135011
1.767370926605649	0.8264838274479623
1.768598813143668	0.7781961379296346
1.760597645586304	0.7382869691002015
1.7451974089560358	0.705322836748924
1.7237262516504475	0.6782333037517578
1.6971642221609151	0.6562162286723536
1.6662450461082534	0.6386728885658537
1.6315252946152539	0.625163433173152
1.5934327426961854	0.6153765057927888
1.5523013508646561	0.6091090643111415
1.5083977603315861	0.6062538689830885
1.4619427354263863	0.6067930449950939
1.4131302242959918	0.610796736087083
1.362146447425601	0.6184262040455949
1.3091915957562024	0.6299407586221824
1.254507347389887	0.6457074296490357
1.1984145726913806	0.6662108416237226
1.1413673531644155	0.6920573070438562
1.0840315921222095	0.7239597455646412
1.0273978706557476	0.7626753262422268
0.9729347898805855	0.8088418907505543
0.9227701428695675	0.8626248702542696
0.8798313427119128	0.9230775414997686
0.8477622770133088	0.9872486823978858
0.8303115801808706	1.0495198705674362
0.8300131797364644	1.1022532985018723
0.8466606348209675	1.1384332161902564
0.8768005284351923	1.1548253649404725
0.914899867852558	1.1529397846787393
0.955289633852685	1.1373332642447038
0.9935458423505262	1.1132386374847107
1.0268841492393288	1.0850971480748486
1.0539221781930959	1.056121895230003
1.0742513395805768	1.0284164979777894
1.0880546006167062	1.0032588516778742
1.0958364301453447	0.9813696456324301
1.0982550987840816	0.9631142729591788
1.0960282857909982	0.9486390642852057
1.0898855132684124	0.9379567676613328

Tabla 6.4: Instancia de pruebas de Tominaga [2].

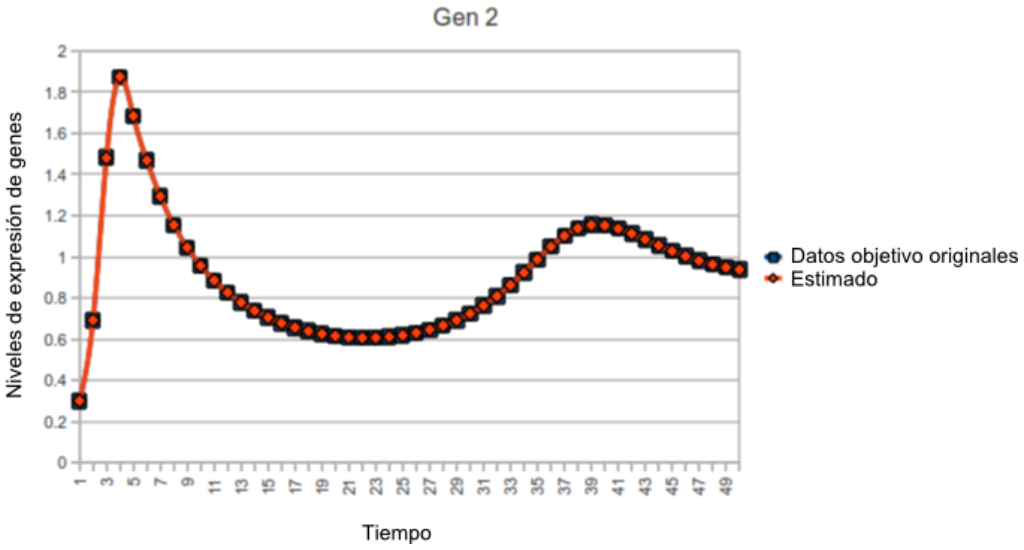

Figura 6.3: Datos objetivo y estimados para el gen 2.

6.4. Resultados

Para probar la eficacia de la implementación se probó con diferentes combinaciones de número máximo de generaciones y tamaños de población. Se realizaron 20 ejecuciones de cada conjunto de pruebas y los resultados se muestran en las tablas 6.5, 6.6 y 6.7:

- 100 generaciones con tamaños de población de 32, 64, 128, 256, 512, 1024, 2048 y 4096 individuos (ver tabla 6.5)

- 500 generaciones con tamaños de población de 32, 64, 128, 256, 512, 1024, 2048 y 4096 individuos (ver tabla 6.6)

- 1000 generaciones con tamaños de población de 32, 64, 128, 256, 512, 1024, 2048 y 4096 individuos (ver tabla 6.7)

# indv	# Gen	tiempo (s) CPU	Aptitud CPU	tiempo (s) GPU	Aptitud en el GPU	Aceleración
32	100	0.68	4.39158	0.64	5.03668	×1.06
64	100	1.36	5.17165	0.63	4.39576	×2.15
128	100	2.66	5.23209	0.62	3.50573	×4.29
256	100	5.33	5.17518	0.63	5.09926	×8.46
512	100	10.71	5.15556	0.66	5.20999	×16.22
1024	100	21.56	4.45661	0.65	4.78358	×33.16
2048	100	42.72	4.15864	0.65	4.42818	×65.72
4096	100	84.63	2.72947	0.66	4.837	×128.22

Tabla 6.5: Comparación de los tiempos computacionales entre las versión secuencial y la versión implementada en CUDA para 100 generaciones.

# indv	# Gen	tiempo (s) CPU	Aptitud CPU	tiempo (s) GPU	Aptitud en el GPU	Aceleración
32	500	3.53	1.20813	3.07	0.1357531	×1.14
64	500	7.09	0.0238907	3.14	0.0110196	×2.25
128	500	13.97	0.0242551	3.15	0.0172489	×4.43
256	500	27.86	0.0127262	3.17	0.00386007	×8.78
512	500	55.6	0.0175084	3.12	0.00935788	×17.82
1024	500	110.86	0.0200963	3.18	0.00658496	×34.86
2048	500	234.31	0.0184026	3.14	0.0043593	×74.62
4096	500	438.5	0.0143981	3.16	0.00700156	×138.76

Tabla 6.6: Comparación de los tiempos computacionales entre las versión secuencial y la versión implementada en CUDA para 500 generaciones.

# indv	# Gen	tiempo (s) CPU	Aptitud CPU	tiempo (s) GPU	Aptitud en el GPU	Aceleración
32	1000	7.32	1.89028	6.24	0.000593101	×1.17
64	1000	14.64	0.003957	6.28	0.000440774	×2.33
128	1000	29.31	2.48775e-05	6.3	0.000226659	×4.65
256	1000	58.21	2.82812e-05	6.29	8.89689e-05	×9.25
512	1000	116.57	2.84008e-05	6.36	8.85665e-05	×18.32
1024	1000	232.72	3.69976e-05	6.29	0.000360468	×36.99
2048	1000	463.56	6.2814e-05	6.31	8.13485e-05	×73.46
4096	1000	919.65	5.3545e-05	6.35	8.016232e-05	×144.82

Tabla 6.7: Comparación de los tiempos computacionales entre las versión secuencial y la versión implementada en CUDA para 1000 generaciones.

6.5. Discusión de resultados

Los resultados indican que la implementación en GPU propuesta en esta tesis produce reducciones significativas en el tiempo necesario para resolver este problema, con respecto a la ejecución secuencial en un CPU. Para la configuración más costosa, en la que se cuenta con 4096 individuos y 1000 generaciones, se obtuvo una aceleración de más de 145 veces con respecto a la versión secuencial. De tal forma, que en la versión implementada en el CPU el tiempo necesario para finalizar la ejecución del programa es de 919.65 segundos, mientras que en la versión en GPU fue de 6.35 segundos. Cabe señalar que los valores de aptitud entre las dos versiones son muy similares (en ambos casos se converge a un valor cercano a cero), lo que indica que se han modelado de manera correcta los datos de prueba.

También es importante señalar que Tominaga [2] reporta que para una configuración de 1000 individuos y 1000 generaciones su implementación requirió 4085 segundos para su ejecución, obteniendo un valor muy cercano en cuanto a la aptitud con respecto a nuestra implementación (8×10^{-5}).

Capítulo 7

Conclusiones y trabajo futuro

7.1. Conclusiones

Los algoritmos evolutivos son técnicas metaheurísticas de búsqueda y optimización que se utilizan cuando otras alternativas no son viables. Éste tipo de técnicas resultan particularmente útiles en problemas de optimización donde los algoritmos de programación matemática presentan diversas limitantes. Por ejemplo, cuando las funciones objetivo no son diferenciables.

Dentro de los algoritmos evolutivos, la evolución diferencial (ED) ha mostrado ser un algoritmo robusto y eficaz en espacios de búsqueda continuos. Por esta razón se eligió a este algoritmo evolutivo como motor de búsqueda para el trabajo aquí reportado. Dentro de las variantes de la ED, el esquema DE/rand/2/bin fue el utilizado debido a la simplicidad en su implementación ya que ha sido utilizado en trabajos anteriores relacionados con el problema de esta tesis.

En esta tesis abordó el problema de la inferencia de modelos de las redes reguladoras de genes. Las redes reguladoras de genes son las interacciones existentes entre un conjunto de genes, los cuales afectan sus niveles de expresión entre sí. Éste es un

problema muy complejo y costoso (computacionalmente hablando). Sin embargo, el impacto que puede tener el entender la manera en que se desarrollan estas interacciones es muy importante para diversas áreas, como en el desarrollo de tratamientos de enfermedades en los que los genes no están funcionando de la manera en que deberían, manifestándose de más (por ejemplo, la producción excesiva de células), o manifestándose de menos (p. ej.,la falta de producción de alguna proteína necesaria para la supervivencia del organismo).

Existen diversos métodos para modelar las redes reguladoras de genes, pero debido a las ventajas que presentan los sistemas de ecuaciones diferneciales, se optó por su uso.
Se optó por adoptar los sistemas S (un tipo de sistemas de ecuaciones diferenciales). Esto, sin embargo, tiene el inconveniente de que se requiere un gran número de parámetros para describir un sistema S. Para encontrar los parámetros que modelen los niveles de expresión que reproduzcan los obtenidos mediante el uso de microarreglos de ADN, se utilizó un algoritmo evolutivo en el que la función objetivo fue minimizar el error entre los datos producidos mediante un conjunto de parámetros, y los obtenidos experimentalmente. La solución a este problema requiere de un gran número de evaluaciones de la función objetivo para obtener errores cercanos a cero, que son los aceptables en la práctica. Debido a que el tiempo computacional requerido para obtener un modelo aceptable es muy alto y crece conforme se incrementa la dimensión del problema, en esta tesis se propuso el uso de cómputo de alto rendimiento basado en GPUs para resolver esta desventaja.

Los resultados presentados en esta tesis indican que es posible obtener reducciones significativas en el tiempo requerido para resolver este problema (se lograron reducciones de hasta 400 veces el tiempo de cómputo, con respecto a la versión secuencial) usando GPUs.

Las principales contribuciones de esta tesis son:

- Se mostró que el uso de GPUs es una forma económica y eficiente de realizar computo de alto rendimiento.
- Se mostró que los algoritmos evolutivos pueden implementarse eficientemente en GPUs.
- Esta parece ser primera utilización de evolución diferencial implementada en GPUs para resolver este problema.
- Se obtuvo una importante reducción en el tiempo de cómputo necesario para la inferencia de las redes reguladoras de genes.

7.2. Trabajo Futuro

A continuación se presentan algunas posibles áreas de trabajo futuro:

- Implementar un buscador local acoplado a la evolución diferencial, que acelere su convergencia.
- Implementar otros métodos numéricos más eficientes para la resolución de sistemas de ecuaciones diferenciales.
- Implementar paralizaciones en la solución de sistemas de ecuaciones diferenciales
- En general, es posible realizar optimizaciones al código para reducir la latencia de acceso a la memoria del GPU.
- Emplear varias y no sólo una GPU para aumentar la paralización de la solución del problema.

- Finalmente, también es deseable probar el algoritmo con redes reguladoras de genes de mayor tamaño, las cuales requieren varios días de tiempo de cómputo, e incluso uso de supercómputo, cuando se usan algoritmos secuenciales.

Apéndice A

Métodos numéricos para resolver sistemas de ecuaciones diferenciales

En muchos de los problemas de ciencias e ingeniería se requiere el uso de ecuaciones diferenciales para poder modelar un sistema.

Las ecuaciones diferenciales sirven para modelar situaciones físicas que ocurren en las ciencias naturales, ingeniería y otras disciplinas donde existe una razón de cambio o funciones desconocidas con respecto a variables independientes. Estos modelos pueden ser sencillos, involucrando sólo una ecuación diferencial para la función desconocida, o puede tenerse un modelo más complejo en el que se involucra un sistema de ecuaciones diferenciales para varias funciones desconocidas. Las leyes mecánicas del movimiento de los cuerpos, cuando se expresan de forma matemática, se describen mediante ecuaciones diferenciales. Usualmente, esas ecuaciones están acompañadas de una condición que especifica el estado inicial del sistema. A esto se le conoce como la condición inicial que, junto con la ecuación inicial, forman lo que se conoce como problema del valor inicial. Debido a que la solución de un problema de valor inicial es muy complicada de obtener mediante métodos analíticos, se han propuesto diversos

métodos numéricos [76] para dar una solución aproximada al problema. A continuación, se presentan los 2 métodos para la solución de ecuaciones diferenciales con los que se trabajaron en esta tesis.

A.1. Método de Euler

El método de Euler es el más simple de los métodos numéricos disponibles para la resolución de ecuaciones diferenciales. Aunque no es muy utilizado en la práctica, contiene las ideas fundamentales en las que se basan los demás métodos.

Tomando en cuenta la longitud de paso h constante entre los puntos x_i. Esto es :

$$x_i = x_0 + ih \tag{A.1}$$

y

$$x_0 = a \tag{A.2}$$

Entonces el método de Euler se define como:

$$y_{n+1} = y_n + hf(x_n, y_n), \tag{A.3}$$

con

$$y_0 = y(x_0) \tag{A.4}$$

La ecuación A.3 es fácil de obtener de diversos modos: geométricamente, aproximando la función mediante tangentes, desarrollando la serie de Taylor usando su primer orden o aproximando la derivada $y'(x)$ por su versión finita $\left(y\left(x_{n+1}\right) - y\left(x_n\right)\right)/h$. Existen teoremas que garantizan la convergencia del método de Euler así como las cotas del error local, es decir, el error que se comete en el cálculo de y_{n+1} a partir de y_n.

En el algoritmo 7 se presenta el pseudocódigo del método de Euler.

Algoritmo 7: Método de Euler.

Para aproximar la solución al problema de valor inicial

$$y' = f(t, y), \quad a \leq t \leq b, \quad y(a) = \alpha, \tag{A.5}$$

en (N+1) números uniformemente espaciados en el intervalo [a,b]:

Input: Extremos a, b; número de puntos N; condición inicial α

Output: aproximación ω a y para los (N+1) valores de t.

$h = (b - a)/N$

$t = a$;

$\omega = \alpha$

OUTPUT(t, ω)

for $i = 1, 2, ..., N$ **do**

  Tome $\omega = \omega + hf(t, \omega)$; (Cálculo de ω_i)

  $t = a + ih$(Cálculo de t_i).

end

A.2. Método de Runge-Kutta

El método de Euler consiste en quedarse con el primer término de la serie de Taylor de la solución. Este método se puede generalizar tomando más términos en la serie de Taylor y así reducir su error. El problema de esto es que requieren derivadas superiores de $f(x, y)$. El método de Runge-Kutta sustituye el cálculo de las derivadas por evaluaciones en puntos intermedios. En el algoritmo 8 se presenta el pseudocódigo

del método de Runge-Kutta de orden 4 que fue el que se utilizó en esta tesis.

Algoritmo 8: Método de Runge-Kutta Orden 4.

Para aproximar la solución al problema de valor inicial

$$y' = f(t, y), \quad a \leq t \leq b, \quad y(a) = \alpha, \tag{A.6}$$

en (N+1) números uniformemente espaciados en el intervalo [a,b]:

Input: Extremos a, b; entro N; condición inicial α

Output: aproximación ω a y para los (N+1) valores de t.

Tome $h = (b - a)/N$

$t = a$;

$\omega = \alpha$

OUTPUT(t, ω)

for $i = 1, 2, ..., N$ **do**

Tome $K_1 = hf(t, \omega)$;

$K_2 = hf(t + h/2, \omega + K_1/2)$;

$K_3 = hf(t + h/2, \omega + K_2/2)$;

$K_4 = hf(t + h, \omega + K_3)$;

$\omega = \omega + (K_1 + 2K_2 + 2K_3 + K_4)/6$;(Cálculo de ω_i)

$t = a + ih$(Calculo de t_i)

OUTPUT(t, ω)

end

Apéndice B

Requerimientos de CUDA

En este apéndice se revisarán los requerimientos necesarios para poder realizar cómputo paralelo en la arquitectura de NVIDIA CUDA

Para poder desarrollar aplicaciones en CUDA se requiere contar con lo siguiente:

- Un procesador gráfico con arquitectura CUDA
- El controlador de desarrollo de NVIDIA
- El conjunto de herramientas de CUDA (*CUDA Toolkit*)
- Un compilador de C estándar

B.1. Procesador gráfico con arquitectura CUDA

Actualmente, es fácil identificar un procesador con arquitectura CUDA debido a que todos los GPUs desarrollados por NVIDIA desde la salida de la GeForce 8800 GTX en el año 2006, utilizan esta arquitectura. Sin embargo, se puede consultar la lista completa de dispositivos que son compatibles con CUDA en [77]. De tal forma, es seguro suponer que todas los GPUs liberados a partir del 2007, con más de 256

<table>
<tr><td></td><td colspan="5">Versión de capacidad de cómputo</td></tr>
<tr><td>Características</td><td>1.0</td><td>1.1</td><td>1.2</td><td>2.0</td><td>2.1</td></tr>
<tr><td>Soporte para funciones atómicas que operan en enteros de 32 bits en la memoria global</td><td>No</td><td colspan="4">Si</td></tr>
<tr><td>Soporte para funciones atómicas que operan en enteros de 32 bits en la memoria compartida</td><td colspan="2">No</td><td colspan="3">Si</td></tr>
<tr><td>Soporte para funciones atómicas que operan en enteros de 64 bits en la memoria global</td><td colspan="2">No</td><td colspan="3">Si</td></tr>
<tr><td>Soporte para números de punto flotante de doble precisión</td><td colspan="3">No</td><td colspan="2">Si</td></tr>
<tr><td>Soporte para funciones atómicas que operan en enteros de 64 bits en la memoria compartida</td><td colspan="3">No</td><td colspan="2">Si</td></tr>
<tr><td>Máximo número de hilos por bloque</td><td colspan="3">512</td><td colspan="2">1024</td></tr>
<tr><td>Cantidad de memoria local por hilo</td><td colspan="3">16 KB</td><td colspan="2">512 KB</td></tr>
</table>

Tabla B.1: Características por versión de capacidad de cómputo.

Mb de memoria, pueden ser utilizados para desarrollar y ejecutar código escrito con CUDA C.

Es conveniente mencionar que aunque todas los GPUs recientes soportan CUDA, con el paso de los años ha habido mejoras en el hardware, y se han agregado capacidades a éste, por lo que los GPUs más recientes soportan más conjuntos de instrucciones que los primeros. Nvidia decidió establecer versiones de acuerdo a las características que soporta cada uno de sus dispositivos y se refiere a éstas como capacidades de cómputo. Hasta el momento, las versiones de capacidad de cómputo que han salido son: 1.0, 1.1, 1.2, 1.3, 2.0 y 2.1. Estas versiones son incrementales, es decir, cada nueva versión soporta a las que le preceden. Por ejemplo, la versión 1.2 soporta todas las características disponibles de las versiones 1.0 y 1.1. La guía de programación de CUDA [78] contiene una lista con todas los GPUs disponibles hasta el momento con su correspondiente versión de capacidad de cómputo. La tabla B.1 muestra algunas de las características que presenta cada versión capacidad de cómputo. La lista completa de características por versión puede consultarse en [78].

B.2. El controlador de desarrollo de Nvidia

Nvidia proporciona un controlador o *driver* de desarrollo mediante el cual son soportadas todas los GPUs que estén basados en la arquitectura CUDA; el driver puede obtenerse del sitio de CUDA [77]

B.3. El conjunto de herramientas de CUDA (*CUDA Toolkit*)

Una vez que se tiene el hardware (GPU) y el driver de Nvidia, es posible ejecutar código compilado de CUDA C. Esto es, se pueden descargar y ejecutar aplicaciones que exploten la capacidad de paralelismo de la arquitectura CUDA. Sin embargo para poder desarrollar software en CUDA C es necesario software adicional, el cual puede obtenerse de manera gratuita en la página de CUDA [77].

Dentro de las herramientas que se proporcionan por parte de Nvidia encontramos:

- Compilador(nvcc): Debido a que una parte del código que se desarrolla se ejecuta en el GPU, Nvidia proporciona un compilador que genera código ejecutable para el GPU. Este compilador se llama nvcc y está basado en gcc de GNU.
- CUDA-GDB: Cuda-GDB es una herramienta de depuración basada en el GDB de GNU
- Visual Profiler: Es una interfaz gráfica para medir el rendimiento de una aplicación y encontrar posibles optimizaciones para aprovechar al máximo las capacidades del GPU.
- Bibliotecas: CUDA proporciona una serie de bibliotecas que contienen funciones optimizadas para realizar tareas comunes. Entre las bibliotecas podemos

encontrar:

- CUBLAS. Es una implementación en GPU de BLAS (Basic Linear Algebra Subprograms), que es una API para realizar operaciones de álgebra lineal tales como multiplicación de matrices y vectores.
- CUFFT. Es la biblioteca de CUDA para la transformada rápida de Fourier. Esta biblioteca cuenta con implementaciones de varios algoritmos para realizar la transformada rápida de Fourier aprovechando el paralelismo de los GPUs. La CUFFT puede ser ampliamente utilizada en aplicaciones de física o tratamiento de señales.
- CUSPARSE. La biblioteca CUSPARSE contiene un conjunto de subrutinas de álgebra lineal utilizadas para la manipulación de matrices dispersas diseñadas para ser llamadas desde C o C++.
- CURAND. Es la biblioteca de CUDA para el uso y generación de números aleatorios. Posee diferentes algoritmos para la generación de números aleatorios dependiendo de las propiedades estadísticas que se quiere que tengan éstos.
- Thrust. Ésta es una de las bibliotecas más recientes introducidas en el CUDA Toolkit y esta basada plantillas basada en la STL de C++.

B.4. Un compilador de C estándar

Como se mencionó anteriormente, Nvidia proporciona un compilador (nvcc) para generar código ejecutable en el GPU. Sin embargo se requiere un compilador que genere la parte que será ejecutada en el CPU. Los compiladores más soportados por Nvidia son gcc de GNU y el compilador de visual C++ de Microsoft. Sin embargo

también pueden ser utilizados otros compiladores como el icc de Intel. No obstante, es importante indicar que el rendimiento no ha sido optimizado para estos compiladores.

Apéndice C

Publicaciones Realizadas

Con el trabajo presentado en esta tesis se realizó la siguiente publicación:

- Presenté el articulo titulado "A GPU-Based Implementation of Differential Evolution for Solving the Gene Regulatory Network Model Inference Problem" en el "4th Workshop on Parallel Architectures and Bioinspired Algorithms" parte del congreso "The Twentieth International Conference on Parallel Architectures and Compilation Techniques (PACT 2011)" realizado en Galveston, Texas, USA (octubre 2011), el articulo será publicado bajo el número ISBN 978-84-695-0254-9.

Bibliografía

[1] European Bioinformatics Institute (EBI). What is bioinformatics? http://www.ebi.ac.uk/2can/bioinformatics/bioinf_what_1.html. Consultado el: 01/11/2011.

[2] S. Kikuchi, D. Tominaga, M. Arita, and M. Tomita. Pathway finding from given time-courses using genetic algorithm. *Genome Informatics Series*, pages 304–305, 2001.

[3] NIH Working Definition of Bioinformatics and Computational Biology, July 2000.

[4] J. Xiong. *Essential bioinformatics*. Cambridge University Press, 2006.

[5] National Institutes of Health. Genbank. http://www.ncbi.nlm.nih.gov/sites/entrez?db=nucleotide, 2011.

[6] Center for Information Biology Japan. The dna data bank of japan. http://www.ddbj.nig.ac.jp/, 2011.

[7] European Molecular Biology Laboratory. Embl nucleotide sequence db. http://www.ebi.ac.uk/embl/index.html, 2011.

[8] European Molecular Biology Laboratory-EBI. Uniprot. `http://www.ebi.ac.uk/uniprot/`, 2011.

[9] Flybase Consortium. Flybase. `http://flybase.org/`, 2011.

[10] European Bioinformatics Institute and Sanger et al. Wormbase. `http://http://www.wormbase.org/`, 2011.

[11] Sean E. Where did the blosum62 alignment score matrix come from? *Nature Biotechnology*, 22:1035–1036, 2004.

[12] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453, March 1970.

[13] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, March 1981.

[14] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, October 1990.

[15] D. J. Lipman and W. R. Pearson. Rapid and sensitive protein similarity searches. *Science*, 227(4693):1435–1441, March 1985.

[16] M. Borodovsky and D. Mcininch. GeneMark: Parallel Gene Recognition for both DNA Strands. *J. Comput. Biochem.*, 17:123–133, 1993.

[17] A. L. Delcher, K. A. Bratke, E. C. Powers, and S. L. Salzberg. Identifying bacterial genes and endosymbiont DNA with Glimmer. *Bioinformatics*, 23(6):673–679, March 2007.

[18] Inc Softberry. Fgenesb: Bacterial operon and gene prediction. `http://linux1.softberry.com/berry.phtml?topic=fgenesb&group=programs&subgroup=gfindb`, 2011.

[19] Oak Ridge National Laboratory. Grailexp: Grail experimental gene discovery suite. `http://compbio.ornl.gov/Grail-1.3/`, 2011.

[20] C. Burge. Prediction of complete gene structures in human genomic DNA. *Journal of Molecular Biology*, 268(1):78–94, April 1997.

[21] T. H. Jukes and C. R. Cantor. *Evolution of Protein Molecules*. Academy Press, 1969.

[22] M. Kimura. A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. *Journal of Molecular Evolution*, 16(2):111–120, December 1980.

[23] M. O. Dayhoff, R. M. Schwartz, and B. C. Orcutt. A model of evolutionary change in proteins. *Atlas of protein sequence and structure*, 5(suppl 3):345–351, 1978.

[24] G. H. Gonnet, M. A. Cohen, and S. A. Benner. Exhaustive matching of the entire protein sequence database. *Science (New York, N.Y.)*, 256(5062):1443–1445, June 1992.

[25] N. C. Jones and P. A. Pevzner. *An Introduction to Bioinformatics Algorithms (Computational Molecular Biology)*. The MIT Press, August 2004.

[26] F. Crick. Central Dogma of Molecular Biology. *Nature*, 227(5258):561–563, August 1970.

[27] Stuart Kauffman. Homeostasis and Differentiation in Random Genetic Control Networks. *Nature*, 224(5215):177–178, October 1969.

[28] M. Ridley. *Evolution*. Oxford readers. Oxford University Press, 1997.

[29] A. Hoffman. *Arguments on evolution : a paleontologist's perspective*. Oxford University Press, 1989.

[30] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. SpringerVerlag, 2003.

[31] G. Rudolph. Convergence of non-elitist strategies. In *Proceedings of the First IEEE International Conference on Evolutionary Computation*, pages 63–66. IEEE Press, 1994.

[32] C. Lin. An evolutionary algorithm with non-random initial population for path planning of manipulators. In *Proceedings of the 22nd International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems: Next-Generation Applied Intelligence*, IEA/AIE '09, pages 193–201, Berlin, Heidelberg, 2009. Springer-Verlag.

[33] K. A. De Jong. *Evolutionary Computation*. The MIT Press, 1st edition edition, 2002.

[34] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, August 2000.

[35] R. Storn and K. Price. Differential Evolution- A Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces. Technical report, 1995.

[36] R. Storn and K. Price. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *J. of Global Optimization*, 11:341–359, December 1997.

[37] K. Price, R. Storn, and J. A. Lampinen. *Differential Evolution: A Practical Approach to Global Optimization*. Springer, 1st edition edition, 2005.

[38] V. Feoktistov. *Differential Evolution*. Springer, 1st edition edition, 2006.

[39] G. E. Moore. Cramming More Components onto Integrated Circuits. *Electronics*, 38(8):114–117, April 1965.

[40] D. B. Kirk and Wen mei. *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2010.

[41] M. P. Frank. Approaching the physical limits of computing. In *Proceedings of the 35th International Symposium on Multiple-Valued Logic*, pages 168–185, Washington, DC, USA, 2005. IEEE Computer Society.

[42] M. J. Flynn. Some Computer Organizations and Their Effectiveness. *IEEE Trans. Comput.*, C-21:948+, 1972.

[43] T. G. Mattson, B. A. Sanders, and B. L. Massingill. *Patterns for Parallel Programming*. Addison-Wesley Professional, 1 edition, September 2004.

[44] Barbara Chapman, Gabriele Jost, and Ruud van van der Pas. *Using OpenMP: Portable Shared Memory Parallel Programming (Scientific and Engineering Computation)*. The MIT Press, October 2007.

[45] D.E. Culler, J.P. Singh, and A. Gupta. *Parallel computer architecture: a hardware/software approach*. The Morgan Kaufmann Series in Computer Architecture and Design. Morgan Kaufmann Publishers, 1999.

[46] J. Sanders and E. Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley Professional, 1 edition, July 2010.

[47] A. Ladikos, S. Benhimane, and N. Navab. Efficient visual hull computation for real-time 3D reconstruction using CUDA. In *Proceedings of the 2008 Conference on Computer Vision and Pattern Recognition Workshops*, volume 0, pages 1–8, Los Alamitos, CA, USA, 2008. IEEE Computer Society.

[48] Performance evaluation of H.264/AVC decoding and visualization using the GPU. volume 6696, pages 669606+. SPIE, 2007.

[49] S. A Manavski. Cuda compatible GPU as an efficient hardware accelerator for AES cryptography. *2007 IEEE International Conference on Signal Processing and Communications*, 9 Suppl 2(November):65–68, 2007.

[50] S. S. Stone, J. P. Haldar, S. C. Tsao, Hwu, B. P. Sutton, and Z. P. Liang. Accelerating advanced MRI reconstructions on GPUs. *J. Parallel Distrib. Comput.*, 68(10):1307–1318, October 2008.

[51] M. Garland, S. Le Grand, J. Nickolls, J. Anderson, J. Hardwick, S. Morton, E. Phillips, Y. Zhang, and V. Volkov. Parallel Computing Experiences with CUDA. *IEEE Micro*, 28:13–27, July 2008.

[52] S. Manavski and G. Valle. CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment. *BMC Bioinformatics*, 9(Suppl 2):S10–9, March 2008.

[53] TOP500 Supercomputer List June 2011. `http://www.top500.org`. Consultado el: 01/11/2011.

[54] VanLuong, Nouredine M., and E. Talbi. Gpu-based island model for evolutionary algorithms. In *GECCO 10: Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 1089–1096, New York, NY, USA, 2010. ACM.

[55] Deepak S. and Pierre C. Gpgpu-compatible archive based stochastic ranking evolutionary algorithm (g-asrea) for multi-objective optimization. In *PPSN (2)*, pages 111–120, 2010.

[56] L. de P. Veronese and R. A. Krohling. Differential evolution algorithm on the gpu with c-cuda. In *IEEE Congress on Evolutionary Computation*, page 1, 2010.

[57] D. H. Irvine and M. A. Sevageau. Efficient solution of nonlinear ordinary differential equations expressed in s-system canonical form. *SIAM J. Numer. Anal.*, 27(3):704–735, 1990.

[58] C. Spieth, F. Streichert, N. Speer, and A. Zell. Optimizing topology and parameters of gene regulatory network models from time-series experiments. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2004)*, volume 3102 (Part I) of *LNCS*, pages 461–470, 2004.

[59] S. Kimura, M. Hatakeyama, and A. Konagaya. Inference of s-system models of genetic networks using a genetic local search. In *The 2003 Congress on Evolutionary Computation, 2003. CEC '03.*, page 631, 2003.

[60] P. Koduru, Zhanshan D., S. Das, S.M. Welch, J.L. Roe, and E. Charbit. A multiobjective evolutionary-simplex hybrid approach for the optimization of differential equation models of gene networks. *IEEE Transactions on Evolutionary Computation*, 12(5):572, 2008.

[61] C. Spieth, F. Streichert, N. Speer, and A. Zell. Multi-objective model optimization for inferring gene regulatory networks. In C. A. Coello Coello, A. Hernandez Aguirre, and E. Zitzler, editors, *Evolutionary Multi-Criterion Optimization*, volume 3410 of *Lecture Notes in Computer Science*, page 607. Springer Berlin / Heidelberg, 2005.

[62] N. Noman and H. Iba. Inferring gene regulatory networks using differential evolution with local search heuristics. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 4:634–647, 2007.

[63] Praveen K., Sanjoy D., Stephen W., and Judith L. R. Fuzzy dominance based multi-objective ga-simplex hybrid algorithms applied to gene network models. In *GECCO (1)*, pages 356–367, 2004.

[64] C. Spieth, F. Streichert, N. Speer, and A. Zell. A memetic inference method for gene regulatory networks based on s-systems. In *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753)*, page 152, 2004.

[65] N Noman and H. Iba. Inference of gene regulatory networks using s-system and differential evolution. In *GECCO 05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 439–446, New York, NY, USA, 2005. ACM.

[66] C. Auliac, V. Frouin, X. Gidrol, and F. d'Alche Buc. Evolutionary approaches for the reverse-engineering of gene regulatory networks: A study on a biologically realistic dataset. *BMC Bioinformatics*, 9(1):91, 2008.

[67] Tominaga D., Okamoto M., Maki Y., Watanabe S., and Eguchi Y. Nonlinear numerical optimization technique based on a genetic algorithm for inverse problems: Towards the inference of genetic networks. In *Proceedings of the German Conference on Bioinformatics (GCB99)*, pages 101–111, 1999.

[68] V. Zumer J. Brest, B. Boskovic. An improved self-adaptive differential evolution algorithm in single objective constrained real-parameter optimization. In *Pro-*

ceedings of the 2010 Congress on Evolutionary Computation CEC2010, pages 1073–1080, 2010.

[69] J. Handl, D. B. Kell, and Knowles J. D. Multiobjective optimization in bioinformatics and computational biology. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 4(2):279–292, 2007.

[70] Praveen Koduru, Sanjoy Das, Stephen Welch, Judith L. Roe, and Zenaida P. Lopez-Dee. A co-evolutionary hybrid algorithm for multi-objective optimization of gene regulatory network models. In *GECCO 05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 393–399, New York, NY, USA, 2005. ACM.

[71] Roland Schwarz, Patrick Musch, Axel von Kamp, Bernd Engels, Heiner Schirmer, Stefan Schuster, and Thomas Dandekar. Yana - a software tool for analyzing flux modes, gene-expression and enzyme activities. *BMC Bioinformatics*, 6:135, 2005.

[72] E. Keedwell and A. Narayanan. Discovering gene networks with a neural-genetic hybrid. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 2:231–242, July 2005.

[73] C. Spieth, F. Streichert, J. Supper, N. Speer, and A. Zell. Feedback memetic algorithms for modeling gene regulatory networks. In *2005 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology*, page 1, 2005.

[74] S. Kikuchi. Dynamic modeling of genetic networks using genetic algorithm and s-system. *Bioinformatics*, 19(5):643, 2003.

[75] Alina Sîrbu, Heather J Ruskin, and Martin Crane. Comparison of evolutionary algorithms in gene regulatory network model inference. *BMC Bioinformatics*, 11:59, 2010.

[76] R. L. Burden and D. J. Faires. *Numerical Analysis*. Brooks Cole, 7 edition, December 2000.

[77] NVIDIA Corporation. Nvidia cuda website. `http://www.nvidia.com/cuda`. Consultado el: 01/11/2011.

[78] NVIDIA Corporation. *NVIDIA CUDA C Programming Guide*. 2011.