UNIVERSIDAD AUTÓNOMA DE CHIHUAHUA

FACULTAD DE INGENIERÍA

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

MAESTRÍA EN INGENIERÍA EN COMPUTACIÓN

ALGORITMO MEMÉTICO BASADO EN GPU CON DIVERSIDAD Y ESTRUCTURA PARA LA INFERENCIA DE MODELOS DE REDES REGULADORAS DE GENES

TESIS

PARA OBTENER EL GRADO DE MAESTRO EN INGENIERÍA EN COMPUTACIÓN

PRESENTA

ANGEL OMAR FLORES OLIVAS

DIRECTOR DE TESIS

DR. LUIS CARLOS GONZALEZ GURROLA

CHIHUAHUA, CHIHUAHUA. Enero 2017

UNIVERSIDAD AUTÓNOMA DE CHIHUAHUA

FACULTAD DE INGENIERIA

MAESTRÍA EN INGENIERÍA EN COMPUTACIÓN

ALGORITMO MEMÉTICO BASADO EN GPU CON DIVERSIDAD Y ESTRUCTURA PARA LA INFERENCIA DE MODELOS DE REDES REGULADORAS DE GENES

TESIS

PARA OBTENER EL GRADO DE MAESTRO EN INGENIERÍA EN COMPUTACIÓN

APROBADO:

PhD. Luis Carlos González Gurrola, director

PhD. Fernando Martínez Reyes, sinodal

PhD. Alain Manzo Martínez, sinodal

ENERO 2017

CHIHUAHUA, CHIH.

17 de enero de 2017

I.S.C. ÁNGEL OMAR FLORES OLIVAS

Presente

En atención a su solicitud relativa al trabajo de tesis para obtener el grado de Maestro en Ingeniería, nos es grato transcribirle el tema aprobado por esta Dirección, propuesto y dirigido por el director **Dr. Luis Carlos González Gurrola** para que lo desarrolle como tesis, con el título: **"ALGORITMO MEMETICO BASADO EN GPU CON DIVERSIDAD Y ESTRUCTURA PARA LA INFERENCIA DE MODELOS DE REDES REGULADORAS DE GENES".**

ÍNDICE

Facultad de Ingeniería
Circuito No.1, Campus Universitario 2
Chihuahua, Chih. C.P. 31125
Tel. (614) 442-95-00 www.fing.uach.mx

Capítulo 5. Computación evolutiva
- 5.1 Representación
- 5.2 Función de evaluación
- 5.3 Población
- 5.4 Diversidad
- 5.5 Mecanismo de selección de padres
- 5.6 Operadores de variación
- 5.7 Mecanismo de selección de sobrevivientes
- 5.8 Inicialización
- 5.9 Condición de paro
- 5.10 Algoritmos meméticos

Capítulo 6. GPGPU-Computo paralelo
- 6.1 Modelos de programación paralela
- 6.2 Arquitectura GPU
- 6.3 Hardware GPU
- 6.4 Librerías para cómputo paralelo en GPU
- 6.5 CUDA

Capítulo 7. Metodología
- 7.1 Pre procesamiento
- 7.2 Diseño de funciones
- 7.3 Aceleración por hilos y STREAMS
- 7.4 Control de información
- 7.5 Búsqueda local
- 7.6 Diversidad

Capítulo 8. Resultados
- 8.1 Tiempos de ejecución
- 8.2 Estructura
- 8.3 Calidad de los resultados

Capítulo 9. Conclusiones
Referencias

Solicitamos a Usted tomar nota de que el título del trabajo se imprima en lugar visible de los ejemplares de las tesis.

A T E N T A M E N T E
"naturam subiecit aliis"

EL DIRECTOR

M.I. JAVIER GONZÁLEZ CANTÚ

FACULTAD DE INGENIERÍA U.A.CH. DIRECCIÓN

EL SECRETARIO DE INVESTIGACIÓN Y POSGRADO

DR. FERNANDO RAFAEL ASTORGA BUSTILLOS

Facultad de Ingeniería
Circuito No.1, Campus Universitario 2
Chihuahua, Chih. C.P. 31125
Tel. (614) 442-95-00 www.fing.uach.mx

Agradecimientos

A mis padres Adrián y Licha que hicieron todo en la vida para que yo pudiera lograr mis sueños, por motivarme y darme la mano cuando sentía que el camino se terminaba, a ustedes por siempre mi corazón y mi agradecimiento.

Al Dr. Luis Carlos González Gurrola por toda su ayuda, su apoyo, su motivación, por todas las oportunidades y sobre todo, por todo el conocimiento adquirido.

Al Dr. Eduardo Rodríguez Tello por su oportunidad de colaboración, su gran ayuda y motivación y por su recibimiento durante mi estancia de investigación en el CINVESTAV.

A Rosario por siempre estar a mi lado apoyándome en todo, por su comprensión, paciencia y amor, dándome ánimos de fuerza y valor para seguir adelante.

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

A mis maestros que influyeron con sus lecciones y experiencias en formarme como una persona de bien y preparada para los retos que pone la vida.

A esas personas importantes en mi vida, que siempre estuvieron listas para brindarme toda su ayuda, esperando algún día regresarles un poco de todo lo inmenso que me han otorgado.

Resumen

La inferencia en redes reguladoras de genes consiste en el análisis de los niveles de expresión de los genes que componen una red para determinar los efectos de las interacciones de los mismos. Este análisis es posible gracias a propiedades específicas del comportamiento de una red de este tipo.

El conocimiento de estas relaciones es importante y tiene muchas utilidades, desde predecir el comportamiento a un medicamento o tratamiento específico o crear medicina personalizada, hasta el diseño inteligente de organismos, como en el caso del diseño de bacterias cuyos procesos ayudan en la descomposición de materiales.

En la actualidad, la búsqueda de inferencia en modelos de redes reguladoras de genes es un proceso complicado que requiere instrumentación muy avanzada, procesos complejos y costosos, sin embargo, existen diversos enfoques que permiten obtener una aproximación utilizando algoritmos computacionales.

Aunque en el ámbito computacional existen varios enfoques para tratar el problema, uno de los más utilizados por la calidad de las soluciones que genera es la utilización de evolución diferencial, en ocasiones combinada con un búsqueda local. El problema de inferencia es modelado a través de ecuaciones diferenciales conocidas como sistemas tipo S.

En este documento se presenta un algoritmo que reúne un conjunto de los enfoques más utilizados en el estado del arte para agilizar y optimizar las funciones de un algoritmo memético para la inferencia de modelos en redes reguladoras de genes, algunas de estas funciones son la aplicación de diversidad, estructura, búsqueda local, aceleración, diseño y optimización para ejecución en GPU.

Abstract

Gene regulatory networks inference is defined as the analysis of expression levels of the genes on a network to determine the effects of interactions among them. This analysis is possible due to the specific properties of this network type. The knowledge of this kind of relationships has significant importance and uses, varying from behavioral prediction of a medical treatment in a patient or a personalized medicine, to the smart design of organisms, such as the design of bacterium for material decomposition process.

Nowadays, finding Gene regulatory networks is a complicated process that requires advanced instrumentation, as well as complicated and expensive procedures. However, some approaches are used to obtain an approximation. Although in Computer science exist several way to tackle this problem, Memetic algorithms using differential evolution combined with a PSO algorithm as local search, are one of the promising techniques to offer quality solutions. Using this approach, the inference problem is modeled through differential equations known as S systems.

In this thesis, an algorithm that integrates a group of the main approaches, from the state of the art, is used to speed up and optimize the functions of a memetic algorithm for gene regulatory networks inference.

Contenido

Contenido

Contenido

Índice de figuras

Índice de tablas

Introducción

Las Redes Reguladoras de Genes (GRN) son representaciones que determinan a través de sus niveles de expresión, la influencia que tiene un gen con cada uno de los genes que comprenden una red (incluso con si mismo). Entender este tipo de relaciones es de suma importancia para el estudio de los sistemas biológicos. Un ejemplo de aplicación es poder predecir la reacción de un organismo con un medicamento, también es ampliamente utilizado en el desarrollo de sustancias y organismos sintéticos[Xiong, 2006].

Estas relaciones se obtienen empleando instrumentación muy avanzada y compleja en un proceso tedioso y costoso, por lo que se han propuesto una variedad muy diversa de enfoques para describir y simular este tipo de redes [Schlitt and Brazma, 2007], uno de los más prácticos e importantes es usar Evolución diferencial (DE) para crear un modelo por medio de ecuaciones diferenciales ordinarias de tipo S (S-systems) [Noman and Iba, 2005].

De esta manera es posible obtener un conjunto de datos mediante la experimentación para luego aplicar ingeniería inversa para generar un modelo computacional que describa el mecanismo completamente.

1.1. Problema de investigación

Los modelos de inferencia en redes reguladoras de genes permiten conocer cómo interactúan los genes y cómo se relacionan a diferentes niveles de expresión, esto es de gran importancia ya que permite predecir el comportamiento de los mismos, por ejemplo, al crear nuevos medicamentos y analizar la manera en que afectan a diferentes organismos. Aunque este tipo de modelos tiene una gran cantidad de fundamentación teórica, la obtención de modelos de inferencia en redes reguladoras de genes es un proceso tardado y complicado que incluye el uso de instrumentación muy avanzada y precisa, por lo que una opción para obtenerlas es utilizando algoritmos computacionales. Estos últimos, aunque hacen más rápido el proceso de obtención, requieren de una gran cantidad de recursos de procesamiento y memoria.

1.2. Pregunta de Investigación

¿De que manera se pueden acelerar los procesos requeridos para obtener resultados más rápidos y precisos en simulación de modelos de redes reguladoras de genes?

1.3. Objetivo

Utilizar el modelo de paralelización y potencia de cómputo de las tarjetas gráficas y adaptar a su funcionamiento los procesos de evolución diferencial mas utilizados en el estado del arte para en combinación con búsqueda local, optimizar los parámetros para la inferencia de modelos de redes reguladoras de genes

1.4. Hipótesis

Al optimizar la ejecución en GPU de un algoritmo memético que trabaje en conjunto con las técnicas mas utilizadas para la inferencia de modelos de redes reguladoras de genes, es posible igualar y superar los resultados de los algoritmos mas destacados por la calidad de sus soluciones, aprovechando los recursos computacionales de manera eficiente y con una configuración mas simple que se traduzca en ahorro de tiempo de ejecución.

1.5. Contribuciones

Aunque el uso de GPU para la aceleración de algoritmos complejos se esta extendiendo de manera sustancial, es limitado el enfoque a utilizarlo para problemas de Bio-informática, mucho más hablando del problema de inferencia de modelos de redes reguladoras de genes.

Aunque éste algoritmo fue específicamente diseñado para tratar el problema de inferencia, se pretende que sea de ayuda para trabajos futuros que requieran el uso de evolución diferencial, PSO o cualquiera de los algoritmos utilizados que ya se encuentran funcionando de manera paralela en GPU.

Tomando en cuanta todos los algoritmos y configuraciones que se incluyen, se pretende que existe una flexibilidad para poder elegir una cantidad mayor de variaciones que se adapten a la instancia que se pretende modelar, por ejemplo, al modificar la cantidad de diversidad o los parámetros de la búsqueda local para analizar las variaciones y diferencias de un mismo conjunto

de datos para elegir la que presente mejor comportamiento. Esta característica no se encontró en ninguno de los trabajos del estado del arte consultados para la realización de este documento.

Estado del arte

En este capítulo se tratará de manera breve las aplicaciones actuales y los conceptos que se utilizarán en la elaboración de este proyecto de investigación. Los conceptos se agrupan de la siguiente manera:

2.1. Algoritmos meméticos

En el documento "*On Memetic Differential Evolution Frameworks: A Study of Advantages and Limitations in Hybridization*", explican brevemente la diferencia entre un algoritmo evolutivo y uno de evolución diferencial y proponen una hibridación con algoritmos de búsqueda local. Analizan tres algoritmos meméticos que implementan búsqueda local coordinada [Neri, 2008].

El primero de ellos Memetic Differential Evolution (MDE), utiliza dos algoritmos de búsqueda local llamados Hooke-Jeeves Algorithm (HJA) y the Stochastic Local Searcher (SLS), según la diversidad de su población, activa la búsqueda local y una selección de ambos métodos es aplicado según su valor a un individuo aleatorio o al mejor individuo. Según su lógica, a valores pequeños, se trata de detectar mejores genes, en contraste a mayores valores se busca una mayor diversidad.

El segundo algoritmo emplea Simulated Annealing (SA) para explorar de manera diferente el espacio de soluciones, Utiliza la búsqueda SLS para crear perturbaciones. La aplicación de la búsqueda local es muy parecida a la del algoritmo anterior. El tercero, crea una población inicial con una distribución uniforme, calcula la calidad de la población y selección al mejor individuo y le aplica Particle Swarm Optimization (PSO). Emplea los algoritmos de Nelder Mead Algorithm (NMA) y Rosenbrock Algorithm (RA).

"*Performance Comparison of Local Search Operators in Differential Evolution for Constrained Numerical Optimization Problems*" justifica el uso de Evolutionary Argorithm (EA) y Swarm Intelligence Algorithms (SIA) para problemas de optimización, además indica que la combinación de métodos de búsqueda global y local puede traducirse en algoritmos meméticos. Realizan una clasificación según el operador de búsqueda [F. van den Bergh, 2012].

"*Modified Differential Evolution with Local Search Algorithm for Real World Optimization*", deta-

lla que la integración de algoritmos meméticos tienen mejores resultados en comparación con los algoritmos de evolución diferencial debido a la integración de métodos de búsqueda local.

2.2. Evolución diferencial sobre GPU

En "*NVIDIA's Next Generation CUDA Compute Architecture: Kepler*", se ofrece la documentación de la arquitectura Kepler de las tarjetas gráficas NVIDIA. A través de la lectura de este archivo se exponen las herramientas y las instrucciones específicas que utiliza CUDA a nivel Hardware, lo más destacable es las diferentes jerarquías y tipos de memoria. Esto es de suma importancia ya que ayuda a decidir cual implementación podría tener un mejor rendimiento, por ejemplo, al poder hacer uso de memorias de registro para acceso con mayor rapidez, con su respectiva desventaja de limitación de espacio [NVIDIA, 2012].

Así mismo, expone algunas de las diferencias con arquitecturas anteriores acentuando las mejoras y nuevas instrucciones que se pueden utilizar, por ejemplo, Streams, que podría ser de gran ayuda para acelerar la ejecución del algoritmo.

Básicamente un Stream permite llamar funciones para ejecutar en paralelo en la GPU, a comparación con las Mayas, una Maya ejecuta la misma instrucción para todos los diferentes datos de un bloque, en contraste, un Stream puede ejecutar varias funciones que utilicen Mayas al mismo tiempo en un Stream independiente.

En el trabajo de Fabrizio Borelli, titulado "*Accelerating Regulatory Networks Inference through GPU/CUDA Programing*"[Fabrizio F. Borelli, 2012], propone el utilizar la paralelización de GPU como una solución de bajo costo para paralelizar algoritmos de búsqueda exhaustiva aplicándolo al problema de redes reguladoras de genes. Este documento básicamente propone la utilización de recursos de GPU y con el apoyo de diagramas y pruebas de tiempo evidencía que es una buena opción para acelerar este tipo de procesos que requiere de un enorme costo computacional. Aunque incluye mucha información de interés sobre la implementación en CUDA, no ofrece detalles complejos del algoritmo de búsqueda exhaustiva que utiliza.

En el documento "*Gene regulatory networks inference using a multi-GPU exhaustive search algorithm, Fabrizio Borelli*"[Fabrizio F Borelli, 2012] retoma el algoritmo de búsqueda exhaustiva para el problema de inferencia en redes reguladoras de genes y explica de una manera más detallada los procesos y funciones que utilizó para abordar el algoritmo desde la perspectiva de GPU. Utiliza múltiples tarjetas gráficas y reporta una gran aceleración obtenida con respecto a la

implementación en CPU con diferentes tamaños de población.

Luis Ramírez Chaves en su trabajo titulado "*A GPU-Based Implementation of Differential Evolution for Solving the Gene regulatory Network Model Inference Problem*", detalla la implementación paralela de un algoritmo de evolución diferencial con CUDA aplicado al problema de redes reguladoras de genes. Al igual que los trabajos anteriores detalla haber obtenido una gran aceleración con respecto a la implementación paralela en CPU. En este documento se detallan también algunas cuestiones muy interesantes, como la formula de fitness y métodos de evaluación en paralelo [Luis E. Ramírez-Chavez, 2011].

En estos trabajos se puede notar que la utilización de GPU para la aceleración del algoritmo de redes reguladoras de genes ha dado buenos resultados tanto en reducción del tiempo de ejecución como en la calidad de la solución. Aunque existen trabajos que tratan el problema de aceleración al problema de inferencia de redes reguladoras de genes desde la perspectiva de GPU, en ellos se utilizan diferentes metodologías como la búsqueda exhaustiva y la evolución diferencial sin las mismas aplicaciones que se integran en este trabajo, es decir, la aplicación de búsqueda local, diversidad y estructura (skeletalizing). La inclusión de estas características permite probar más opciones y configuraciones para hacer el algoritmo más robusto y potente para encontrar buenas soluciones.

Metodología

A partir del libro "*A Developer's Guide to Parallel Computing with GPUs*"[Cook, 2013] se detectaron las funciones y metodologías de utilidad para la implementación. De ahí se sustrajeron algunas ideas importantes como el tipo de estructura de datos, los métodos de paralelización con bloques de uno hasta tres dimensiones y el uso eficiente de los tipos de memoria. Antes de comenzar a detallar estas funciones, es importante conocer algunas herramientas y métodos utilizadas.

3.1. Pre procesamiento

El procesamiento comprende la lectura de los archivos de entrada y su distribución a las localidades de memoria y las tarjetas gráficas que los procesarán.

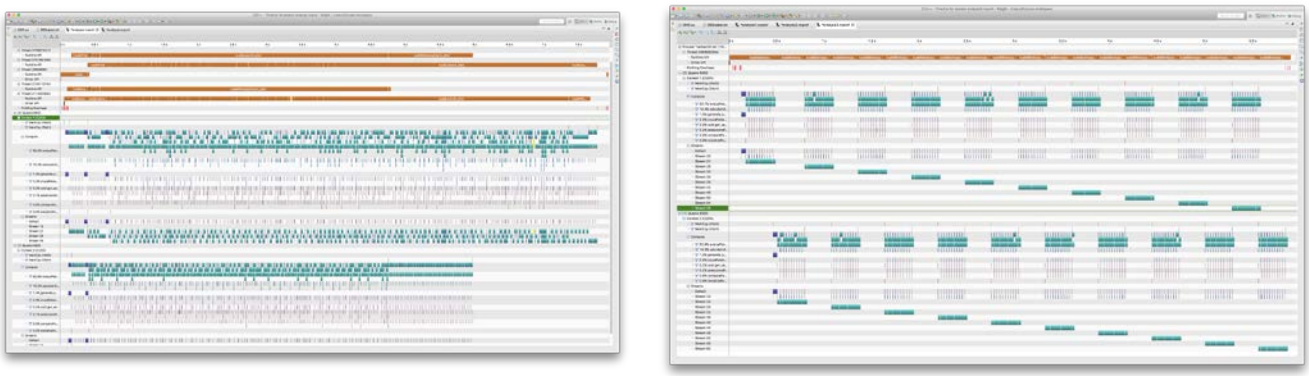
Se adecuó el uso de mas de una tarjeta gráfica, con la idea principal de distribuir los procesos de la ED según el número de tarjetas para acelerar la ejecución de una instancia, sin embargo, las pruebas realizadas daban a conocer que uno de los cuellos de botella fueron las copias de los datos entre la memoria del CPU y la memoria de GPU.

La evolución diferencial al ser cada generación independiente, se incluye una copia de los datos procesados para dividirse en la tarjeta gráfica, esto incluye una copia de memoria de CPU (Host) a GPU (Device) y viceversa en cada modificación en las generaciones, una al comenzar y otra al terminar.

Al distribuir los procesos aumentó el tiempo de ejecución incluso manejando solo una instancia debido a la necesidad de aumentar el número de copias a memoria, aunque también lo hizo la cantidad de procesos concurrentes, no logró mejorar el tiempo.

Debido a esto y consciente de la utilización de que más de una instancia puede ejecutarse de manera más eficiente y puede ser útil para combinación, se determinó que el número de instancias de Evolución Diferencial se dividiera sobre el número de tarjetas para aprovechar el Hardware disponible.

De este modo, al iniciar la ejecución solo son necesarias dos copias de memoria, una al iniciar con los datos de entrada y otra al final con los resultados. En la Figura 3.1 se puede observar la



(a) Streams (b) Sincrona

Figura 3.1: A la izquierda se muestra una salida de NVPROFF utilizando STREAMS dividiendo una instancia en dos tarjetas gráficas a la derecha se aprecia la salida del analizáis con STREAMS de dos instancias en una tarjeta gráfica cada una.

distribución de las copias en memoria en color naranja en ambas variaciones.

Para iniciar los GPU, se reserva memoria para todas las variables y se crea un generador de números aleatorios y la lectura de archivos almacena la información en la memoria previamente reservada.

3.2. Diseño de funciones

En la Figura 3.2 se puede observar una vista general del flujo de los métodos de evolución diferencial.

Es importante mencionar que en cada modificación e inclusión de funciones se revisó que la integridad de los datos se mantuviera, esto para detectar sobre escritura y errores de memoria desde en el momento que se presentan.

3.2.1. Iniciar Población

Para esta función se reciben como parámetros del tamaño del individuo, el tamaño de los órdenes cinéticos y el tamaño de las constantes de ritmo además de un arreglo que corresponde a números aleatorios.

En todas las funciones que requieren números aleatorios se recibe este arreglo, uno de los consejos que menciona la documentación es que es mas eficiente calcular una cantidad determinada de números aleatorios para consumirlos durante la ejecución, que estar generando alea-

3. METODOLOGÍA

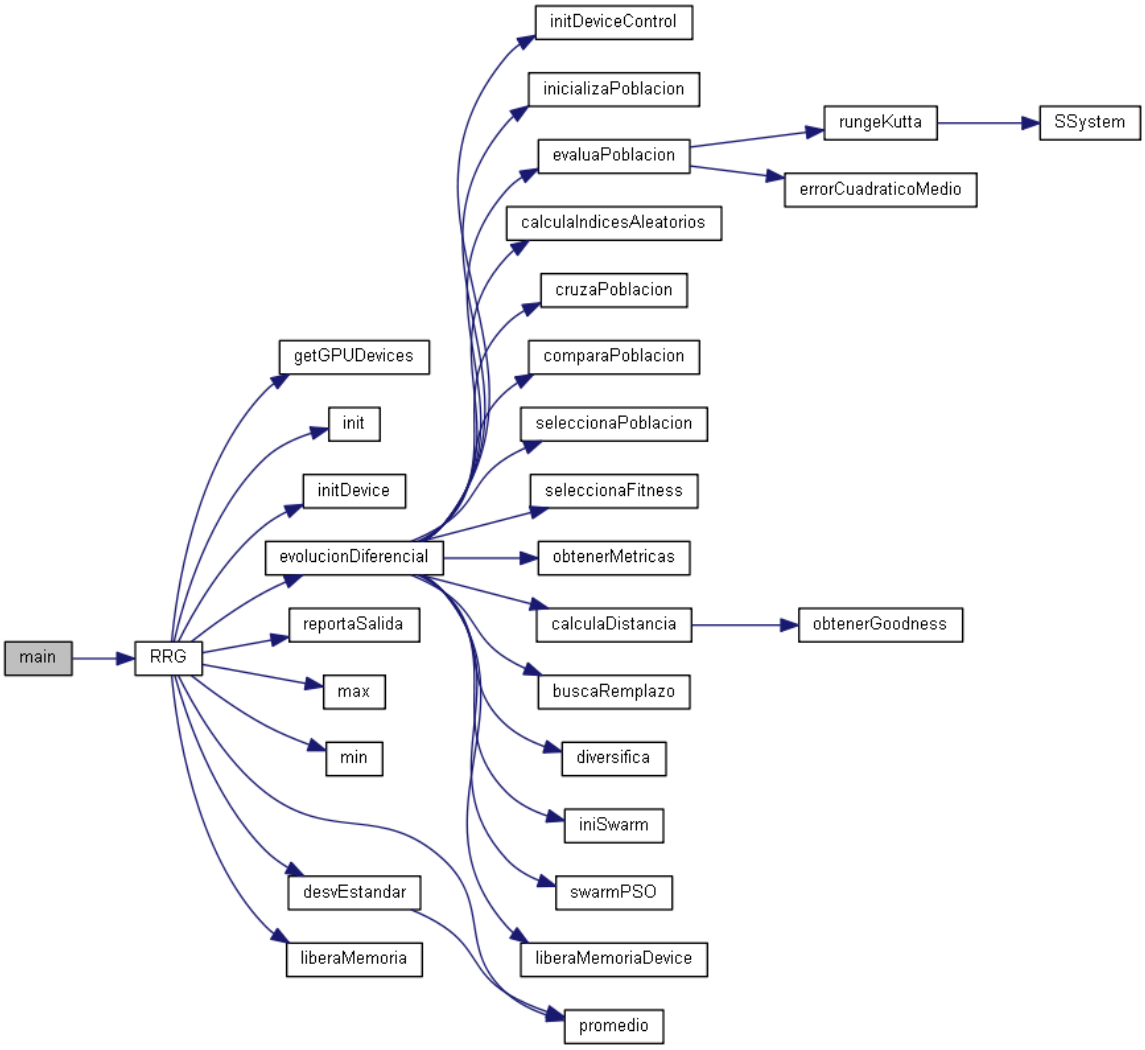


Figura 3.2: Distribución de los procesos de evolución diferencial utilizados.

torios cada vez que se necesiten, esto siempre y cuando no sea dentro de una funciona de un Kernel, en cuyo caso es mas eficiente hacerlo desde el mismo.

Los valores son escritos en un apuntador a la población, al ser la primera función que modificará información este se encuentra "vacío".

Utilizando el modelo de paralelización de CUDA, se inicia cada individuo de la población en un ciclo, el número de parámetros a calcular del mismo corresponden al tamaño del individuo, por lo que se pueden iniciar dentro de otro ciclo.

Dicho de otra manera, lo que se calcula realmente es una cantidad de parámetros aleatorios con límites establecidos correspondientes al tamaño de la población multiplicada por el tamaño de los individuos.

Cada uno de estos parámetros puede ser calculado y evaluado conforme a sus límites en un hilo independiente de la tarjeta gráfica, lo que reduce de manera sustancial la cantidad de tiempo requerido para completar la operación.

Datos: población
Resultado: población inicial con propiedades aleatorias
asignar variables para Kernel;
reiniciar bestSoFar;
Mientras *tamaño de población* **hacer**
    Mientras *tamaño del individuo* **hacer**
        obtener límites;
        asignar propiedad;
        si *valor de propiedad excede límites* **entonces**
            cambiar valor a cero;
        fin
    fin
fin

Algoritmo 1: Pseudocódigo de la función de iniciar población.

3.2.2. Evaluar Población

Este es el método que consume más cantidad de poder de cómputo, utiliza el método de Runge-Kutta (RK) para resolver una ecuación diferencial de un sistema tipo S que determina la

calidad del individuo, además calcula otros parámetros como el error cuadrático medio (MSE) y la desviación media de la raíz cuadrada (RMSE).

Se esperaba obtener todos los tiempo a la vez de todos los individuos utilizando la misma metodología que el método de iniciar población sin embargo, para calcular el tiempo $y(t)$, es necesario conocer $y(t-1)$, por lo que la paralelización de este método requería hacerlo paralelizando el método RK.

Datos: población
Resultado: evaluación de la población
asignar variables para Kernel;
asignar variables para control del individuo;
Mientras *tamaño de población* **hacer**
  establecer valores para g, h, α y β;
  establecer valores para el método RK;
  Mientras *tiempos de evaluación* **hacer**
    método RK;
    sincronizar hilos;
    obtener error cuadrático medio;
  fin
fin

Algoritmo 2: Pseudocódigo de la función de evaluar población.

Runge-Kutta

Fue fundamental revisar la literatura de los trabajos existentes que implementaran este tipo de métodos de forma paralela, comenzando con el trabajo "*The Cuda Implementation Of The Method Of Lines For The Curvature Dependent Flows*" [Žabka, 2011] utilizan el método de Runge-Kutta de segundo y cuarto orden. Este trabajo ofrecía exclusivamente un enfoque a la paralelización del método de Runge-Kutta en CUDA, desgraciadamente el enlace al que hacia referencia al código de la implementación no estaba disponible, por lo que se procedió a contactar al autor del documento, para solicitarle la implementación y comenzar a hacer pruebas. Otro inconveniente presentado en este documento, fue que las referencias se encontraban en idiomas diferentes al español e ingles.

En el trabajo titulado "*Acceleration of Runge-Kutta Integrators*". Éste trabajo, aunque hace una muy buena comparación, no se enfoca detalladamente en la implementación del método en paralelo, sin embargo, fue de gran ayuda para encontrar referencias a trabajos mas completos[Murray, 2012].

A través de la lectura de las referencias del documento, se encontraron varios trabajos de interés, algunos incluso detallan el proceso del Runge-Kutta y otros métodos de implementación en CUDA, no obstante, la gran cantidad de ellos no hace una implementación técnica para paralelizar el método, sino que utilizan un enfoque algebraico, es decir, por medio de reducciones y equivalencias reducen el procedimiento para que sea totalmente independiente de las variables k.

Aunque este método es muy sencillo de implementar y tiene mucha documentación (documentos, libros y resúmenes), no es la opción más viable, ya que en un análisis comparativo que se realiza en una de sus referencias, indica que el error de cálculo es mucho mayor en comparación con métodos secuenciales, esto por la naturaleza matemática que implica su reducción.

Debido a esto, se determinó que el método no sería modificado para darle prioridad a otras áreas de oportunidad.

3.2.3. Índices Aleatorios

Esta función se utiliza para crear índices de referencia que se utilizarán para cruza y mutación.

Datos: índices
Resultado: índices
asignar variables para Kernel;
Mientras *tamaño de población* **hacer**
    llamada a CURAND desde Kernel;
    Mientras *se deban agregar índices* **hacer**
        crear índice aleatorio;
    fin
fin

Algoritmo 3: Pseudocódigo de la función de índices aleatorios.

3.2.4. Cruza de población

Los valores aleatorios determinan la probabilidad de mutación en función de Cr. La mutación esta en función de los índices aleatorios calculados previamente.

Datos: población, índices
Resultado: población
asignar variables para Kernel;
asignar variables de control;
Mientras *tamaño de población* **hacer**
  Mientras *tamaño del individuo* **hacer**
    crear particiones de los individuos según índices aleatorios;
    si *valor aleatorio* < *Cr* **entonces**
      combinar individuos;
    fin
  fin
fin

Algoritmo 4: Pseudocódigo de la función cruza de población.

3.2.5. Comparación de población

Compara las aptitudes de la población original y la población creada. Crea un vector ordenado que indica si el dato de una posición debe de ser remplazado o no, este vector es utilizado para los remplazos tanto en población como en fitness para no tener que realizar la evaluación nuevamente al modificar la población sin haber modificado atributos de sus individuos.

Datos: fitness
Resultado: índices de cambio
asignar variables para Kernel;
Mientras *tamaño de población* **hacer**
  si *individuo nuevo es mejor que el original* **entonces**
    asignar 1 a índice;
  fin
fin

Algoritmo 5: Pseudocódigo de la función comparación de población.

3.2.6. Selección de población

Realiza el cambio en el arreglo de población según el índice de cambio calculado en la función anterior. Si el arreglo indica un uno, se realiza el remplazo del nuevo individuo a la posición correspondiente.

Esta función, esta diseñada de esta manera para que cada uno de los individuos de la población pueda ser comparado en un Hilo del GPU, para que al siguiente paso, sea o no remplazado mediante una comparación simple.

Datos: fitness
Resultado: población
asignar variables para Kernel;
asignar variables de control;
Mientras *tamaño de población* **hacer**
  si *índices de cambio* **entonces**
    remplazar individuo;
  fin
fin

Algoritmo 6: Pseudocódigo de la función selección de población.

3.2.7. Selección de fitness

Al igual que la función anterior, remplaza el valor de una posición según su índice, con la diferencia que el arreglo de entrada corresponde a los valores fitness de la población.

Datos: fitness
Resultado: fitness
asignar variables para Kernel;
asignar variables de control;
Mientras *tamaño de población* **hacer**
  si *índices de cambio* **entonces**
    remplazar fitness;
  fin
fin

Algoritmo 7: Pseudocódigo de la función selección de fitness.

3.2.8. Obtención de métricas

Esta función permite verificar en que cantidad de iteraciones el fitness del mejor individuo (Best So Far) no ha sido mejorado. Esto es útil ya que permite controlar la aplicación de diversidad al conocer la evolución diferencial midiendo la calidad del mejor individuo impidiendo que se estanque por no tener características nuevas que aplicar.

Mientras se realiza la búsqueda también se realiza el cálculo y asignación de los valores de RMS y RMSE.

Datos: fitness
Resultado: BSF, MSE, RMSE
asignar variables de control;
Mientras *tamaño de población* **hacer**
    sincronizar hilos;
    si *mejor valor encontrado* **entonces**
        remplazar BSF;
    fin
fin
si *Se mejoro el BSF* **entonces**
    remplazar BSF, MSE y RMSE;
fin
else
    aumentar número de veces sin mejoras
fin

Algoritmo 8: Pseudocódigo de la función obtener métricas.

3.2.9. Distancia de un individuo con la población

Se calcula la distancia euclidiana de un individuo con cada uno de los otro individuos de la población, esto permite detectar el parecido del mismo con otros para priorizar los que contienen características mas comunes entre si para ser remplazados al aplicar diversidad.

```
//calculo de distancia euclidiana
__global__void calculaDistancia(float *poblacion, int numeroGenes,int indSize,float
```

```
    *aptitud,float *distancia,float *sum, float MValor, float *bestSoFar){
  int i = threadIdx.x + blockIdx.x *  blockDim.x;
   ...

  while(i <  POPSIZE){
    //asignacion de un individuo en un conjunto de hilos
    individuoi   =   poblacion+i*indSize;

    while(j <  POPSIZE){
      //asignacion de un individuo en un conjunto diferente de hilos
      individuoj  =  poblacion+j*indSize;

        //comparacion paralela de hilos con caracteristicas de ambos individuos
        while(k < 2*numeroGenes*numeroGenes + 2*numeroGenes){
        if (i !=  j){
          sum[POPSIZE*i+j]    +=    powf(individuoi[k]-individuoj[k],2);
        }
        else{
          sum[POPSIZE*i+j]  =  nan("");
        }
        k++;

      }

    }
    obtenerGoodness(sum, i, distancia, aptitud, numeroGenes, MValor, bestSoFar);
    i += blockDim.x * gridDim.x;
  }
}
```

3.2.10. Busca remplazo de población

Calcula el valor a partir del cual se remplazaran los individuos para conservar diversidad. Ordena los individuos del menor al mayor según su fitness y elige una posición según porcentaje de remplazo. Cualquier individuo ubicado en una posición siguiente a la elegida será modificado.

Datos: fitness

Resultado: población

asignar variables de control;

calcular porcentaje de cambio;

Mientras *tamaño de población* **hacer**

  Mientras *existan repetidos* **hacer**

    eliminarlos;

  fin

fin

asignar posición de cambio;

Algoritmo 9: Pseudocódigo de la función de remplazo.

3.3. PSO

Se propuso la implementación de un algoritmo PSO como búsqueda local:

```
//Funcion de busqueda local PSO
while( j < indSize-solucionSize){

    /*Calcular vMatriz*/
   vMatrix[j+i*indSize] = WPSO * vMatrix[j+i*indSize] + C1 * (randomValues[k+i*indSize] *
        pbest[j+i*indSize] - swarm[j+i*indSize]) + C2 * (randomValues[(k+1)+i*indSize]  *
        gbest[j]   -   swarm[j+i*indSize]);

   swarm[j+i*indSize]  +=  vMatrix[j+i*indSize];

    /*Verificar si se encuentra dentro de los   limites*/
   if (swarm[j+i*indSize] > upperBound || swarm[j+i*indSize] < lowerBound ){
      swarm[j+i*indSize] = lowerBound + randomValues[k+i*indSize]*(upperBound-lowerBound);
   }

    /*Desplazamiento*/
   k  += 2;
}
```

3.4. Diversidad

Se crea diversidad en la población modificando los valores de g, h, alfa y beta dentro de sus límites de manera aleatoria.

Datos: población, valores aleatorios
Resultado: población
asignar variables para Kernel;
asignar variables de control;
Mientras *tamaño de población* **hacer**
  si *Debe aplicarse diversidad* **entonces**
    Mientras *tamaño del individuo* **hacer**
      obtener limites;
      modificar valores;
    fin
  fin
fin
asignar posición de cambio;

Algoritmo 10: Pseudocódigo de la función de diversidad de población.

Se creó un Kernel para calcular la distancia mínima de un individuo con la población, esto se hace calculando la distancia euclidiana de un individuo con todos los demás de la población, el valor mínimo corresponde a la similitud del mismo con la población en general.

Una vez que se tiene la distancia mínima, se calcula el valor de Goodness del individuo utilizando la formula:

$$g = f + e^{max * be/minimo} \quad (3.1)$$

Se ordenan los individuos del menor al mayor Goodness y se remplaza un porcentaje de individuos definido previamente. Este método de diversidad se aplicará cada vez que el fitness del mejor individuo (bestSoFar) no se mejore después de N generaciones.

```
//Funcion de comparacion de la poblacion
while(i < POPSIZE){

  /*Seleccion de individuo 1*/
```

```
individuoi   =   poblacion+i*indSize;

while(j <  POPSIZE){
        /*Seleccion de individuo 2*/
    individuoj = poblacion+j*indSize;
    sum[POPSIZE*i+j] = 0;

      /*Recorre el tamano del individuo*/
    while(k  <  2*numeroGenes*numeroGenes  +  2*numeroGenes){

        /*Evita la comparacion con el mismo*/
      if  (i  != j){
        sum[POPSIZE*i+j]     +=     powf(individuoi[k]-individuoj[k],2);
      }
      else{
        sum[POPSIZE*i+j]  =  nan("");
      }
      k++;

    }
    j  +=  blockDim.y  * gridDim.y;
}
 /*Obtencion de los resultados de la   medicion*/
obtenerGoodness(sum, i, distancia, aptitud, numeroGenes, MValor, bestSoFar);
i += blockDim.x * gridDim.x;
}
```

El nivel de interacción con genes particulares es muy escaso, por lo que se procedió a la aplicación de la función de cálculo de fitness para penalizar esta baja interacción.

Esta inclusión permite dejar en cero el valor de algunos de los parámetros de *g* y *h* donde no existe interacción con los genes, de igual manera propicia evitar caer en óptimos locales al monitorear la cantidad de veces que estos órdenes cinéticos son 0.

A parir de la definición de este umbral, fue posible obtener una estructura de la red mas dispersa.

3.4.1. Distribución múlti-GPU

La información debe ser distribuida en ambas tarjetas gráficas, además los módulos para obtener números aleatorios también deben de agregarse e iniciarse en cada una, por lo que todas las variables que se copian a GPU son arreglos de tres dimensiones, la primera correspondería a la tarjeta en la cual se utilizará, la segunda a la corrida actual y la ultima corresponde a una variable de control, se realizaron todos los cambios para asignar memoria y se agregó una función para situarla en las tarjetas gráficas.

Los módulos que permiten el control de la información, los flujos en las diversas tarjetas gráficas y algunos otros parámetros importantes se describen a continuación:

```
//Inicializa los arreglos y crea los Streams y el generador de numeros aleatorios en el
    Device correspondiente

void initDeviceControl(int deviceId, int  control){
   /*Asignacion de memoria en CPU de las variables de control*/
   poblacion_D[deviceId][control]  =  (float*)
       malloc(sizeof(float)*individuoSize_H*POPSIZE);
   ---

   /*Seleccion de GPU que se utilizara*/
   CUDA_CALL( cudaSetDevice(deviceId) );

    /*Asignacion de memoria en el GPU de las variables de control*/
   CUDA_CALL(cudaMalloc(&poblacion_D[deviceId][control],
       sizeof(float)*individuoSize_H*POPSIZE));
   ---

    /*Apuntadores a variables compartidas entre CPU y  GPU*/
   CUDA_CALL( cudaHostAlloc((void**) &numBSF[deviceId][control], sizeof(int),
       cudaHostAllocMapped) );
   CUDA_CALL( cudaHostGetDevicePointer(&numBSF_D[deviceId][control],
       numBSF[deviceId][control], 0) );
   ---

    /*Semilla aleatoria para CURAND*/
```

```
    srand(time(NULL));

    /*Iniciador de CURAND*/
    CURAND_CALL(curandCreateGenerator(&randomGenerator[deviceId][control],
      CURAND_RNG_PSEUDO_DEFAULT));
    CURAND_CALL(curandSetPseudoRandomGeneratorSeed(randomGenerator[deviceId][control],
        rand()));
}
```

Es importante mencionar que CUDA requiere asignación de memoria en CPU que después será asignada en GPU, es por esto que existe una asignación con *malloc* y después con *cudamalloc*.

Para las variables de control de diversidad, fitness y error cuadrático medio, se utilizan variables compartidas en memoria utilizando un apuntador, de esta manera es posible evitar copias directas en memoria en cada generación. La limitante del uso de esta es el tamaño de la misma, sin embargo es ideal para variables de control de este tipo.

Hasta este punto ambas tarjetas se distribuyen el trabajo, pero lo hacen de manera secuencial, esto es, que aunque trabajan con el modelo de programación paralela de CUDA, cuyas funciones se realizan en paralelo, es posible ejecutar mas de estas funciones al mismo tiempo por medio de Streams para que las funciones se ejecuten de manera asíncrona en cada tarjeta.

3.5. Aceleración por Hilos y Streams

Básicamente un Stream permite llamar funciones de CUDA para ejecutar en paralelo en la GPU.

Para poner en contexto, un Grid ejecuta la misma instrucción para todos los diferentes datos de un bloque, en contraste, un Stream puede ejecutar varias funciones que utilicen Grids al mismo tiempo en un Stream independiente.

3.5.1. Técnicas

Existen diferentes técnicas para trabajar con Streams, la opción mas viable que trabaja de manera mas eficiente con la implementación es la metodología de una función por Stream. Con el uso de la herramienta Visual Profiler, se realizó un análisis de las diferentes formas de implemen-

tación tomando como la más eficiente la evaluación de individuos por cada corrida del algoritmo, esto sería valido debido a que cada corrida es independiente, a diferencia de las generaciones.

La implementación, aparte del control de Streams y del flujo de los datos, requirió agregar dos parámetros más al Kernel. El primero correspondiente a la copia de memoria para lanzamiento, el cual no sería relevante para esta implementación.

El segundo parámetro indica en que Stream será lanzado el Kernel, por lo que la variable de control lo determinaría como se describe a continuación:

```
//Asignacion de funcion para llamado en Stream
calculaIndicesAleatorios<<<1,bloque, 0,
    streamCalc[device][control]>>>(randomIndices_D[device][control],rand());
```

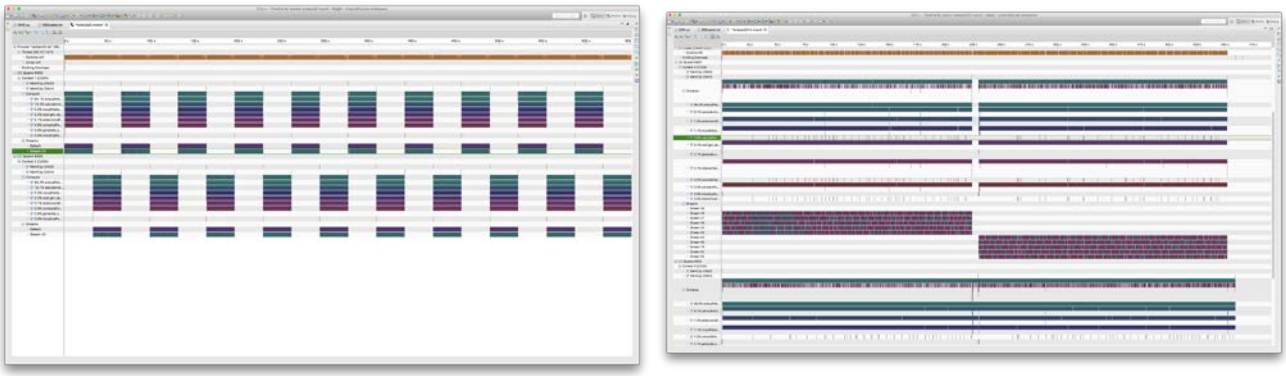
Tras la nueva implementación de Streams, el algoritmo mejoró su eficiencia en cuanto al manejo de memoria y otras oportunidades que mencionaba el análisis guiado, pero no lo hizo con los tiempos de ejecución.

Esto se debió a que aunque se estaba compilando con el indicador *"—default-stream per-thread"* y las funciones se estaban llamaban correctamente, el CPU estaba llamando a las funciones secuencialmente, por lo que se implementó el uso de Threads para llamar las funciones de manera asíncrona.

3.5.2. OpenMP

OpenMP permite el manejo sencillo de Threads, es compatible con CUDA y existe una gran cantidad de documentación por lo que es una excelente opción para la implementación de funciones mediante hilos. CUDA tiene soporte nativo para OMP y para funcionar requiere un compilador GCC. Al compilar requiere las banderas "-Xcompiler -fopenmp" para incluir las dependencias necesarias.

Al ejecutar y analizar con VP, se detecto que la librería no estaba funcionando correctamente ya que no estaba creando los Threads, se hicieron algunos cambios incluso forzándolo a crear la mayor cantidad de hilos posibles sin respuesta positiva, al analizar mas detenidamente en varias fuentes de información, se encontraron algunos inconvenientes al utilizarlo en conjunto con CUDA.



(a) Streams (b) Sincrona

Figura 3.3: A la izquierda se muestra una salida de NVPROFF utilizando STREAMS de manera asíncrona (Sin Hilos) a la derecha se aprecia la salida del analizáis con STREAMS controlados por hilos desde el CPU.

3.5.3. POSIX

Con las primeras pruebas se logro que se realizaran llamadas a funciones en diferentes hilos y Streams en paralelo, por lo que se comenzó con la implementación de la librería. Con ésto se logró el uso de hilos para lanzar todas las corridas al mismo tiempo, teóricamente utilizando Streams es posible ejecutarlas todas al mismo tiempo y distribuyendo el trabajo en las tarjetas gráficas.

3.6. Control de información

Las funciones se ejecutan Streams controlados por hilos de manera concurrente, pero sin ningún tipo de control en el termino de ejecución, lo que causaba perdida de información al no depender de las tareas secuenciales para lanzar los siguientes procesos, Sin embargo, esto fue solucionado mediante la sincronización de la información y de los procesos concurrentes del GPU, el cambio de comportamiento del algoritmo se muestra mediante el análisis de NVProff en la Figura 3.3, a la izquierda las funciones sin sincronizar y a la derecha ya sincronizadas.

Se delegó el control completo de los Streams a los hilos del CPU, por lo que la inclusión de parámetros a los Kernel ya no sería necesaria, el control esta diseñado de la siguiente manera:

```
//Control de numero de instancias de ED
for( int i = 0 ; i < CORRIDAS ; i++){
```

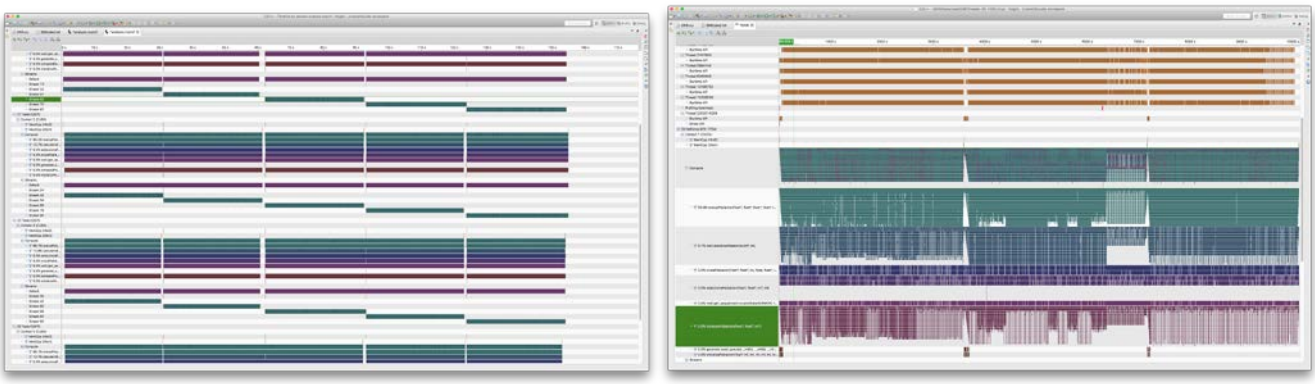
```
    /*Inicio de valores de los parametros de control*/
    args[k][i].serie  =  k;
    args[k][i].corrida = i;
    args[k][i].device = actualDevice;
    args[k][i].control  =  threadControl;

    /*Lanzamiento  de  Hilo  de  control*/
    if (pthread_create(&threads[i], &attr, evolucionDiferencial, (void *)&args[k][i] ))
         exit(-1);

     /*Control de GPU*/
    actualDevice++;
    if (actualDevice >= numGpus){
        actualDevice = 0;
        threadControl++;
        hilosControl++;
    }

    /*Control  de  Hilos  y  sincronizacion*/
    if (hilosControl == STREAMS || i == CORRIDAS-1){
        for( int j = i - corridasControl; j <= i ; j++){
            //printf("\t\tSincronizando %d, de %d\n", j, i);
            pthread_join(threads[j], &status[j]);
        }
        hilosControl  =  0;
        corridasControl  =  0;
    }
    corridasControl++;
}
```

Como resultado de esta implementación, todas las corridas del algoritmo son llamadas en diferentes Hilos al mismo tiempo y controlan el flujo de datos de los Streams en los GPUs, esto permite que se lance un Thread y un Stream por corrida, por lo que todas estas se llaman simultáneamente, por lo cual, se calculan los datos necesarios mientras las funciones terminan de ejecutar una instancia, siempre y cuando no sean dependientes como en el caso de crear núme-



(a) Streams (b) Sincrona

Figura 3.4: En la izquierda se puede observar la distribución del trabajo en las tarjetas Testa C2070, a la derecha un ejemplo de ejecución asíncrona con perdida de información

ros aleatorios. En la Figura 3.4 se puede observar un análisis de todas las funciones del algoritmo funcionando de manera síncrona a la izquierda y de manera asíncrona derecha.

Esta implementación supuso un enorme ahorro de tiempo de ejecución, sin embargo el flujo de la información tenía pérdidas en el caso de instancias grandes de más de 10 genes.

3.6.1. Números Aleatorios CURAND

Con el análisis de la memoria se detectó que al momento de generar números aleatorios se estaba excediendo la memoria de la tarjeta gráfica al tener una cantidad muy grande de datos almacenados en espera para hacer los cálculos, para solucionar esto se limitó la cantidad de procesos paralelos al número de dispositivos disponibles y se reorganizo la función de obtención de números aleatorios.

```
//Control de obtencion de numeros aleatorios con CURAND
if (randControl == MAXRAND-5){
    CURAND_CALL( curandGenerateUniform(randomGenerator[device][control],
     randomFloatValues_D[device][control],
          (individuoSize_H-solucionSize_H)*POPSIZE*MAXRAND) );
    randControl  =  -4;
}
```

Como detalles adicionales para este código, la verificación de la memoria no presento errores de fugas o desbordamiento.

Resultados

En este capitulo se describen los objetivos y los resultados obtenidos con la implementación del algoritmo.

4.1. Tiempos de ejecución

En la Tabla 4.1 se muestra una comparativa de los tiempos de ejecución en 2 variantes en GPU y una en CPU. Las características técnicas se muestran en la Tabla 4.2

Población	Fitness D	Fitness P	Fitness CPU	Tiempo D (Seg)	Tiempo P (Seg)	Tiempo CPU (Seg)	Aceleración D	Aceleración P
32	5.9822	3.1709	4.3915	0.3694	0.2592	0.68	x1.84	x2.62
64	1.6810	1.7183	5.1716	0.7236	0.4971	1.36	x1.87	x2.73
128	1.6929	1.9038	5.2320	1.7329	0.9964	2.66	x1.53	x2.66
256	0.9020	0.3117	5.1751	3.959	2.0216	5.33	x1.34	x2.63
512	0.4753	0.2322	5.1555	8.6625	4.0921	10.71	x1.23	x2.61
1024	0.366	0.1771	4.4566	17.1332	7.5872	21.56	x1.25	x2.84
2048	0.1741	0.1381	4.1586	46.7062	15.7489	42.72	x0.91	x2.71

Tabla 4.1: Valores obtenidos al ejecutar la instancia Tominaga 2 con el equipo de pruebas (P) y con el equipo de desarrollo (D)

	Desarrollo	Pruebas
Modelo de GPU	GeForce GTX 775M	Tesla C2070
Número de GPUs	1	4
Salida al mercado	11/2013	05/2010
Núcleos CUDA	1344	448
CUDA Capability	3.0	2.0
GPU Clock	797 MHz	1.15 GHz
Memoria global	2048 MB GDDR5	5375 MB GDDR5

Tabla 4.2: Características técnicas del equipo de Desarrollo y del equipo de Pruebas, este último consiste en un clúster con tarjetas gráficas específicamente diseñadas para cómputo paralelo.

Como se puede observar, los tiempos de ejecución se lograron reducir con respecto a la imple-

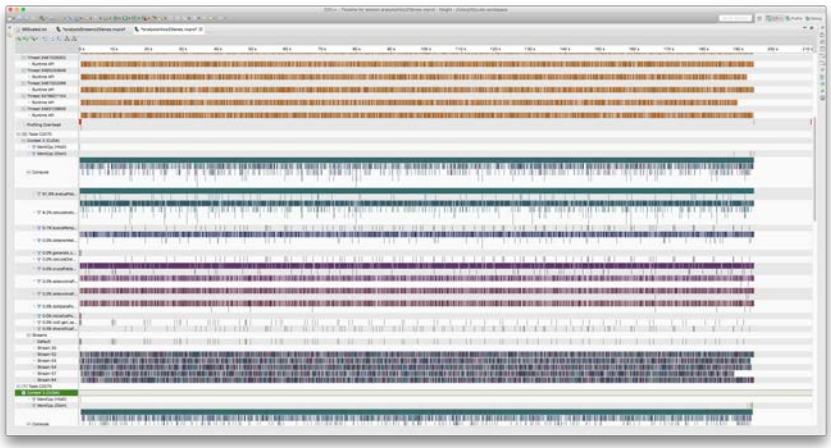


Figura 4.1: Análisis de las funciones ejecutadas en GPU

mentación en CPU en todos los casos con excepción del último, esto debido a que las limitaciones de memoria del GPU de la máquina de desarrollo comienzan a hacerse presentes.

EL tamaño del problema crece según la instancia y el número de individuos, por lo que la memoria del Hardware no basta para almacenar todas las variables involucradas, es en este punto donde se pierde eficiencia.

En el caso de la máquina de pruebas, se alcanzó una gran aceleración con respecto a la velocidad requerida en CPU.

El fitness alcanzado fue menor en la implementación en GPU ya que este algoritmo utiliza las variantes de diversidad y búsqueda local que se describirán más adelante.

En la Figura 4.1 se puede observar como se distribuyen los procesos en las tarjetas gráficas, se puede notar como el proceso de evaluación de la población es el proceso que consume más tiempo de cómputo, esto se debe a que la evaluación se realiza con el método de Runge-Kutta que no es paralelizable. Sin embargo es fácil notar como los demás procesos se ejecutan eficientemente de manera paralela.

4.2. Estructura

Al generar la estructura de la red, el algoritmo tiene algunas variaciones sin embargo si fue posible replicar la estructura original. Las diferentes estructuras encontradas se muestran en la Figura 4.2.

Al hacer el análisis el algoritmo recorre determinada cantidad de genes, en el caso de la

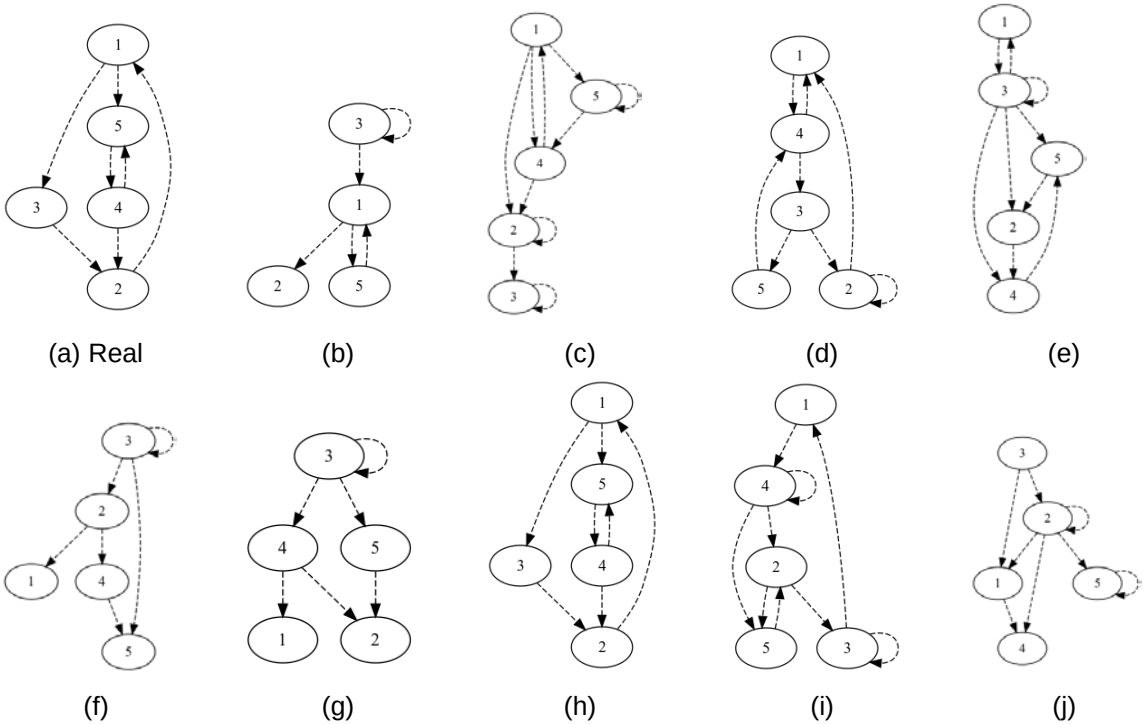


Figura 4.2: Estructura de la red de 5 genes de la instancia Tominaga 5

instancia la profundidad fue definida de acuerdo a la cantidad de genes, es decir que se analizaron cada uno de los genes de la misma.

Aunque esto aumenta el tiempo de ejecución, permite generar la estructura de la red dispersa y mejorar el fitness pero solo cuando encontrar la estructura real.

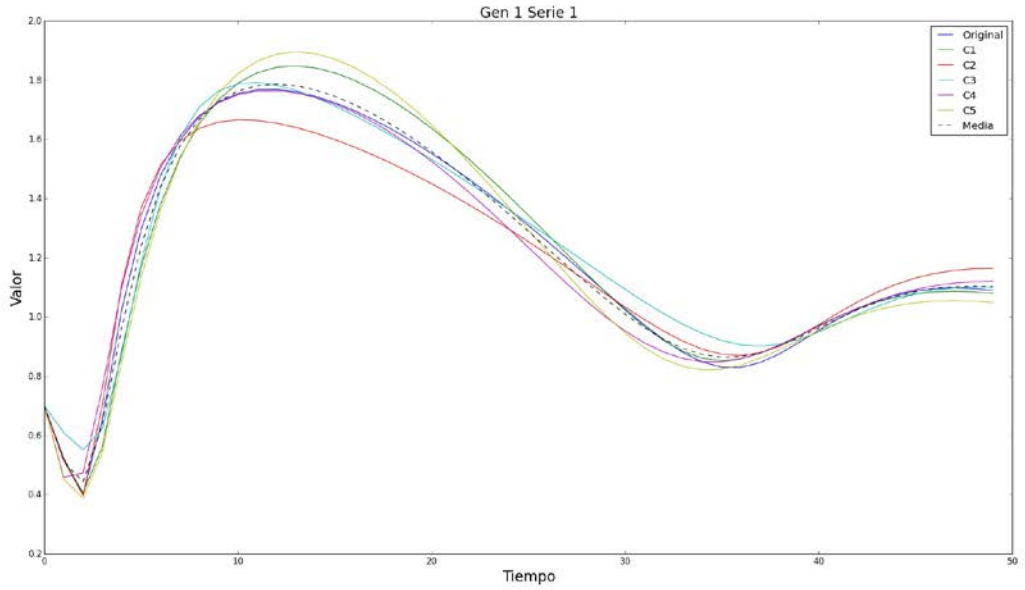
4.3. Aproximación con datos reales

Se realizaron pruebas con la instancia Tominaga [Daisuke TOMINAGA, 1999] de dos genes, ejecutando 20 ejecuciones con 1000 generaciones y 128 individuos. Esta instancia esta compuesta por una serie de 50 tiempos. Se muestran las cinco ejecuciones con menor Fitness en la Figura 4.3.

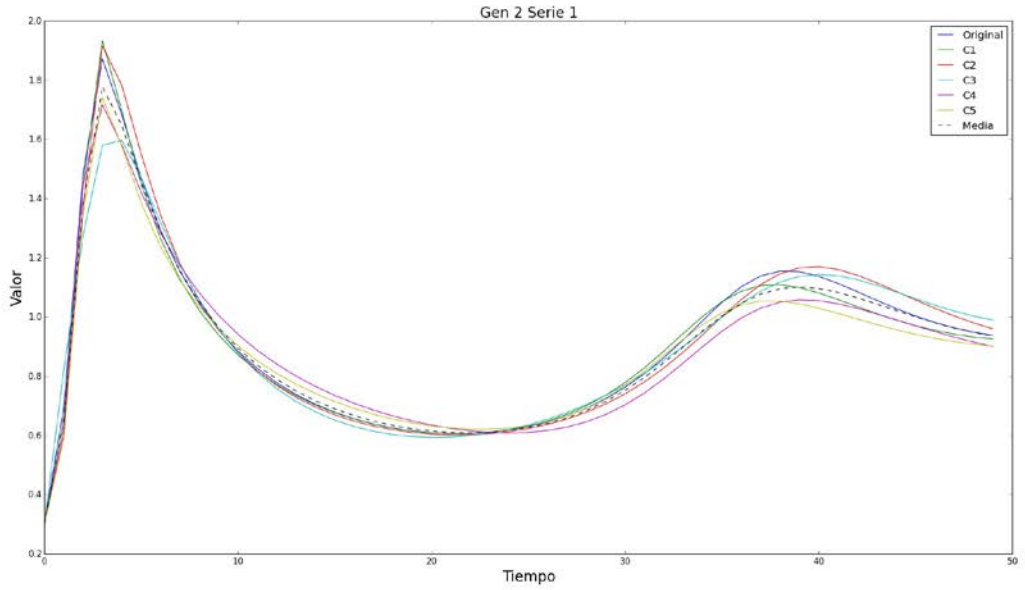
En las Figuras 4.4, 4.5 y 4.6 se muestran los resultados de los cinco mejores resultados para la instancia Tominaga de 5 genes en 3 series diferentes, cada una con 20 tiempos.

Los parámetros de la ejecución corresponden a los mismos para la instancia de 2 genes.

Para la instancia de 2 genes, se puede observar que las salidas tienden a replicar la repre-

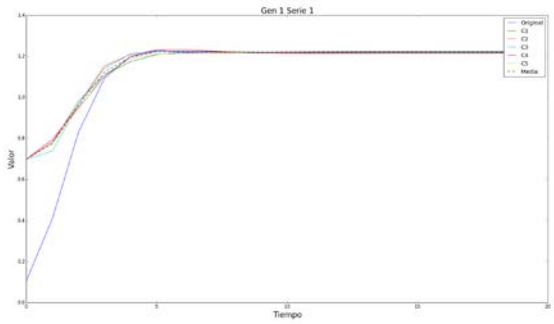


(a) Streams

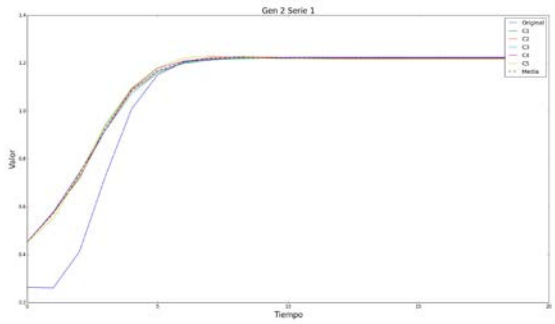


(b) Sincrona

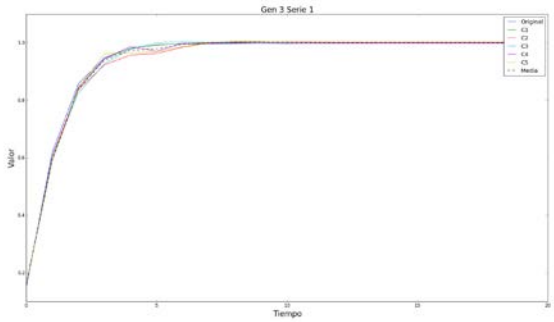
Figura 4.3: Niveles de expresión de la instancia Tominaga 2 genes



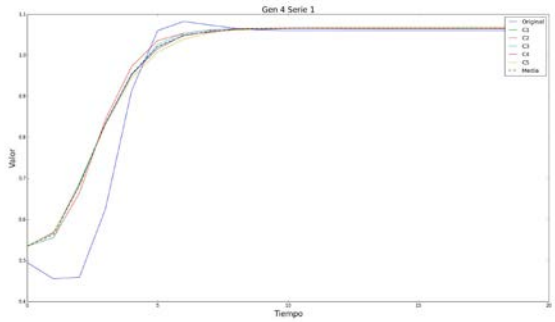
(a) Gen 1



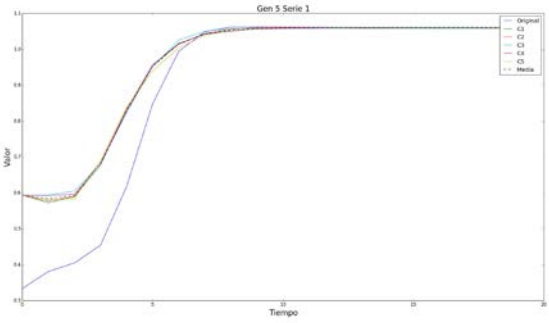
(b) Gen 2



(c) Gen 3

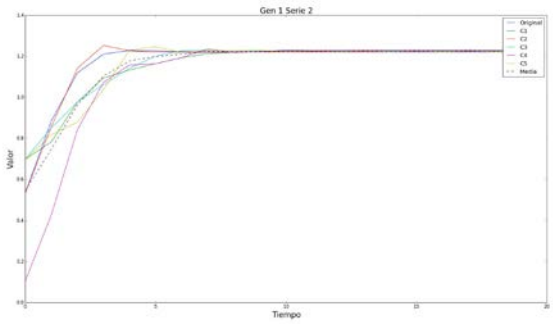


(d) Gen 4

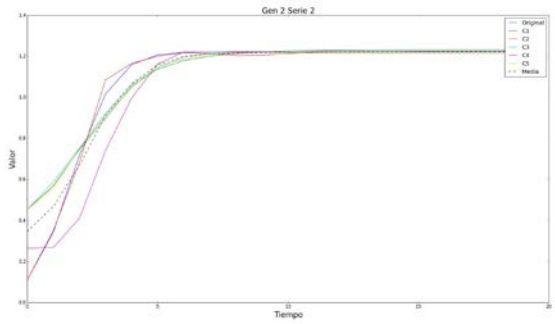


(e) Gen 5

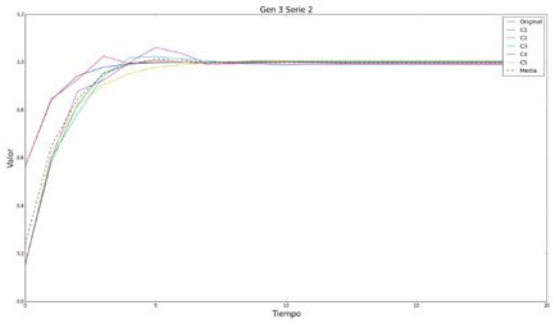
Figura 4.4: Niveles de expresión de la instancia Tominaga 5 genes serie 1



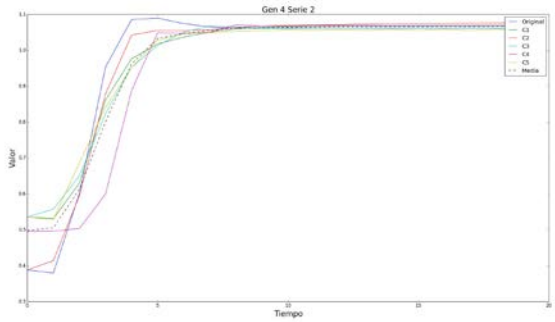
(a) Gen 1



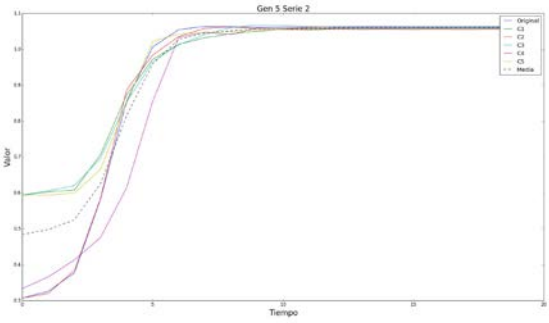
(b) Gen 2



(c) Gen 3

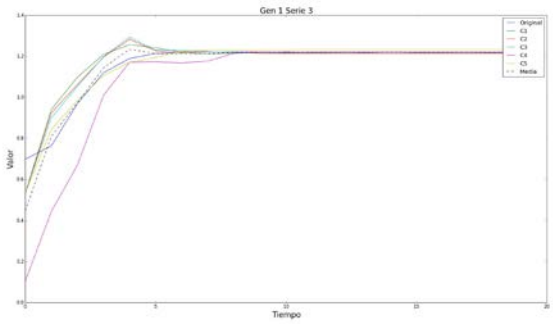


(d) Gen 4

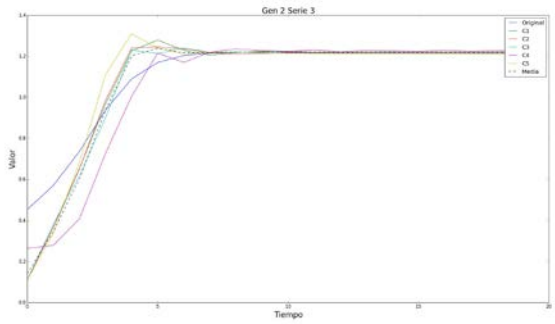


(e) Gen 5

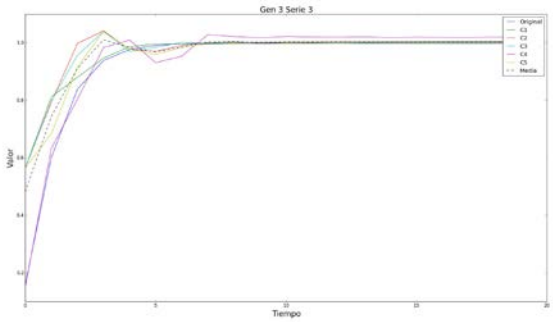
Figura 4.5: Niveles de expresión de la instancia Tominaga 5 genes serie 2



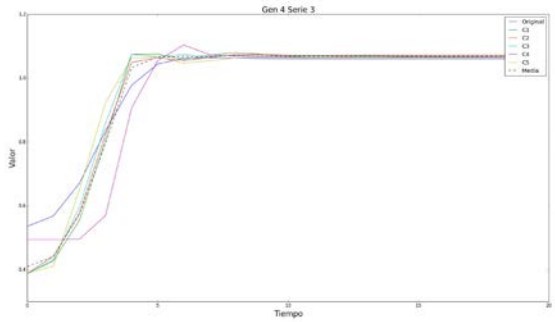
(a) Gen 1



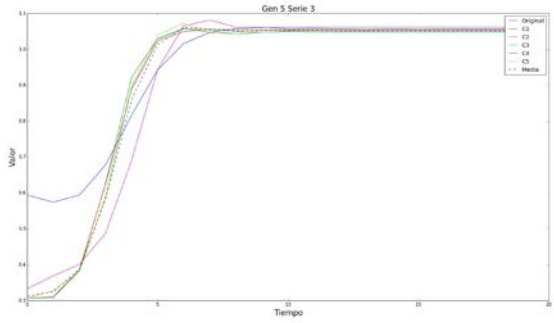
(b) Gen 2



(c) Gen 3



(d) Gen 4



(e) Gen 5

Figura 4.6: Niveles de expresión de la instancia Tominaga 5 genes serie 3

sentación en la gráfica. Aunque existen algunas variaciones no parecen muy relevantes ya que en general fue posible obtener una buena aproximación a la salida real.

En el caso de la instancia de cinco genes, se analizaron 3 series diferentes que incluye la instancia, obteniendo resultados muy parecidos aunque con un poco más de dispersión. La instancia de la serie 2 es la que tuvo un mejor comportamiento al reproducirla.

Esto es debido a que la instancia tiene parámetros conservadores, la inclusión de más generaciones o más individuos podrían obtener mejores resultados.

4.4. Comparativa de Evolución Diferencial y búsqueda local

La Figura 4.7 muestra una gráfica de convergencia comparando cuatro variaciones del algoritmo. Se muestra el mejor resultado de 20 ejecuciones:

- DE/PSO corresponde a un proceso de evolución diferencial completo, seguido de la aplicación de búsqueda local. Ésto sucede en cada generación.
- DE+PSO es la ejecución de evolución diferencial en una determinada cantidad de generaciones, a la cual se le aplica búsqueda local por separado una vez terminado el proceso de evolución diferencial.
- DE es solo la ejecución de evolución diferencial.
- PSO es solo la aplicación de búsqueda local (PSO).

Se puede observar que el mejor comportamiento corresponde a la variante de DE/PSO con el cual se pudo obtener el mejor fitness final, seguido por la variante DE+PSO.

Aunque en este ejemplo en particular ambos resultados son muy parecidos, en gran mayoría de las pruebas realizadas la variante DE/PSO fue la que obtuvo el mejor resultado, es por esto que se determino que sería la variación con mejor comportamiento.

La variante ED tubo un resultado muy bueno a comparación con el método PSO, aunque este último no es comúnmente utilizado para este problema en particular.

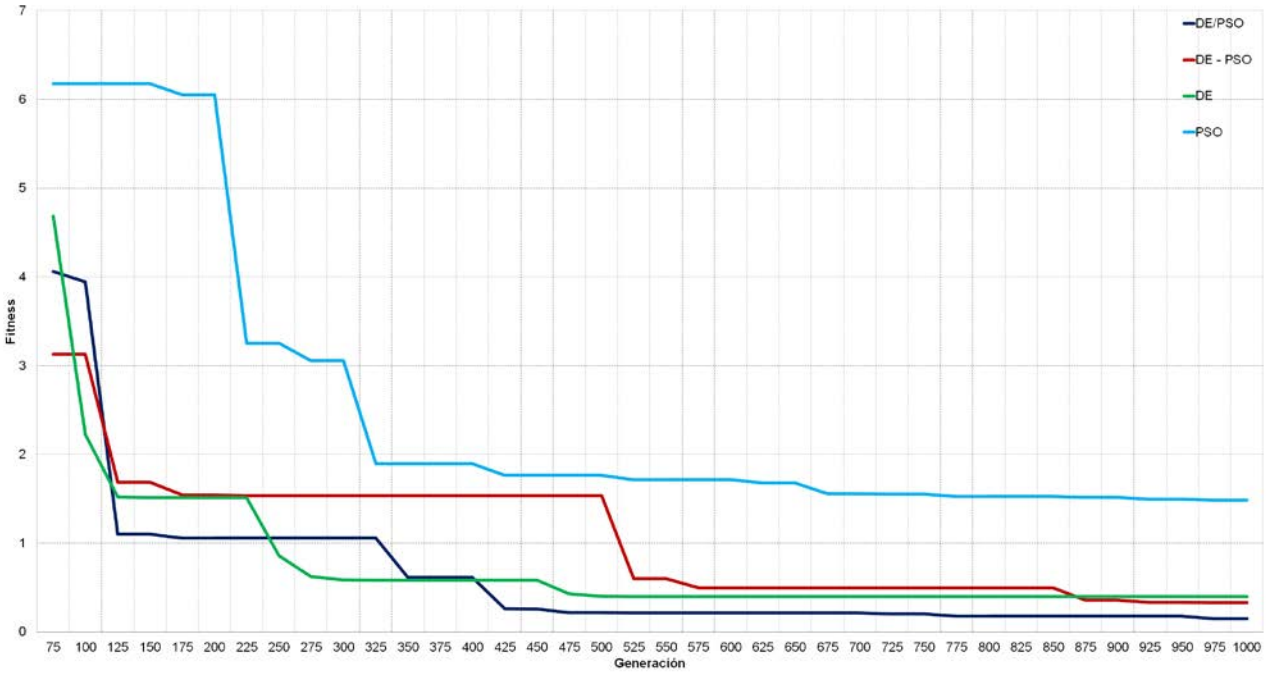


Figura 4.7: Comparación de convergencia con variaciones en la aplicaron de búsqueda local

4.5. Calidad de los Resultados

Para medir la calidad de los resultados, se realizó una comparación con el documento de Sirbu, [Alina Sîrbu, 2010] el cual corresponde a una comparativa de una instancia de cinco genes con diferentes porcentajes de ruido, específicamente 1 %, 2 %, 5 % y 10 %.

Esta comparativa esta realizada con el Framework EVA2, el cual es una recopilación de diferentes algoritmos genéticos para resolver todo tipo de problemas, en este caso específicamente la inferencia en redes reguladoras de genes, para las comparativas se realizo una aproximación con lo que reporta Sirbu y corresponde a cinco algoritmos diferentes que se muestran en la representación.

Los algoritmos son los siguientes:

- GA+ES. Algoritmo genético con estrategias de evolución. El algoritmo genético encuentra las relaciones y las estrategias de evolución, los niveles de expresión
- GA+ANN. Algoritmo genético con red neuronal artificial. Intercambia el uso de estrategias evolutivas de GA+ES por una red neuronal artificial.
- DE+AIC. Evolución diferencial con el criterio de información de Akaike (Akaike's Information

Criterion). Penaliza los niveles de expresión para forzar la dispersión de la red.

- GLSDC. Corrección diferencial de mínimos cuadrados de Gauss. Divide el problema en dos instancias, una de búsqueda local y otra de convergencia.
- PEACE1. Predictor por algoritmo evolutivos y ecuaciones canónicas 1. Basa su funcionamiento en la propiedad de dispersión de las redes reguladoras de genes.

Los resultados serán comparados en dos puntos:

- Ruido: Agrega perturbaciones a la serie real para verificar que tan buena es la aproximación a los datos reales.
- Escalabilidad: Mide como cambia la calidad de las soluciones que aproxima el algoritmo si aumenta la cantidad de genes que tiene que aproximar.

La Figura 4.8 muestra un diagrama de cajas con la aproximación que presenta Sirbu y los resultados obtenidos por el algoritmo.

Se puede observar que el error cuadrático medio es más reducido a comparación con los algoritmos utilizados por Sirbu, este incluso se mantiene al aumentar ruido, lo que podría indicar que no es tan susceptible al ruido como los otros algoritmo de la comparación. Otro punto importante es que el algoritmo muestra menos dispersión.

En la Figura 4.9 se muestra la comparativa de escalabilidad, se utilizaron 4 instancias con 10, 20 30 y 50 genes.

El objetivo es que una mayor cantidad de genes no tenga una implicación negativa importante para el algoritmo.

Se puede observar que para una cantidad de 10 genes, el algoritmo sigue teniendo un buen comportamiento, mientras que a más cantidad de genes, comienza a presentar más dispersión y aunque en todas las ocasiones se logró obtener un error cuadrático medio, en los casos mayores a cinco genes, tiene muchas más variaciones a comparación con los otros algoritmos. En el caso de la instancia de 50 genes, el algoritmo muestra que no puede manejar de manera efectiva una instancia de ese tamaño, al menos con la configuración asignada para las ejecuciones.

Es importante mencionar que estas instancias fueron realizadas con 128 individuos y 1000 generaciones, es decir con parámetros muy conservadores, por lo que la ejecución de instancias con parámetros más robustos tiene la capacidad de mejorar los resultados.

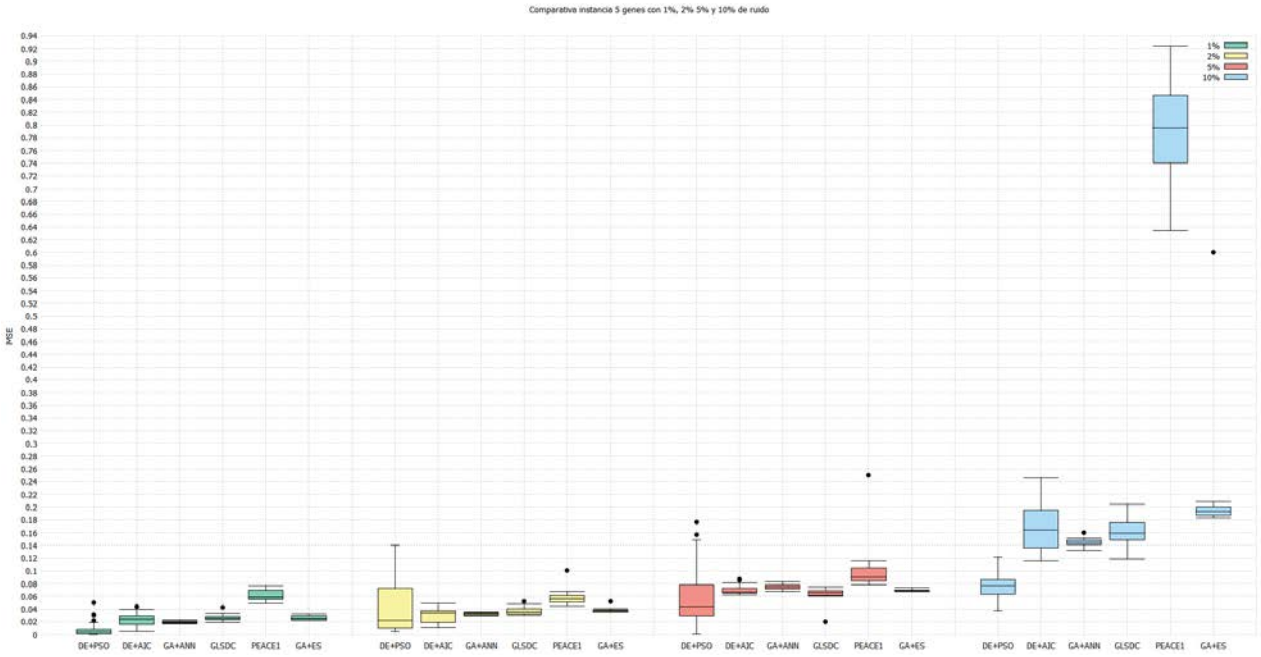


Figura 4.8: Framework EVA2 con los algoritmos DE+AIC, GA+ANN, GLSDC, PEACE1 y GA+ES en 4 instancias con 1 %, 2 %, 5 % y 10 % de ruido. [Alina Sîrbu, 2010]

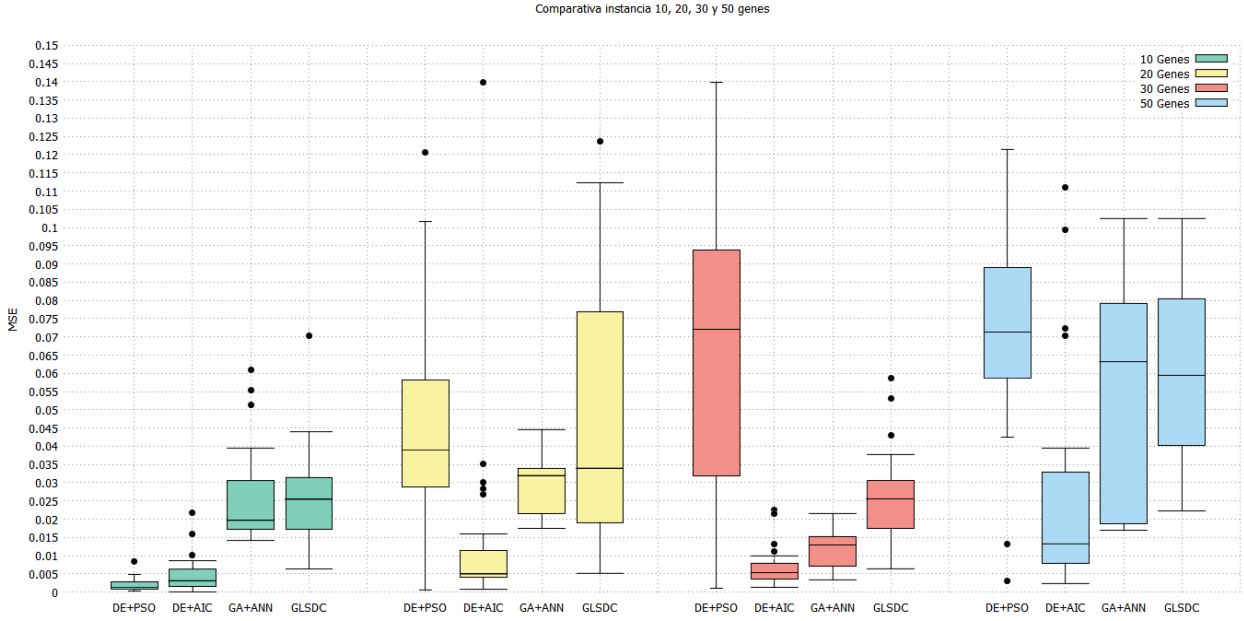


Figura 4.9: Framework EVA2 con los algoritmos DE+AIC, GA+ANN y GLSDC en 4 instancias con 10, 20 30 y 50 genes. [Alina Sîrbu, 2010]

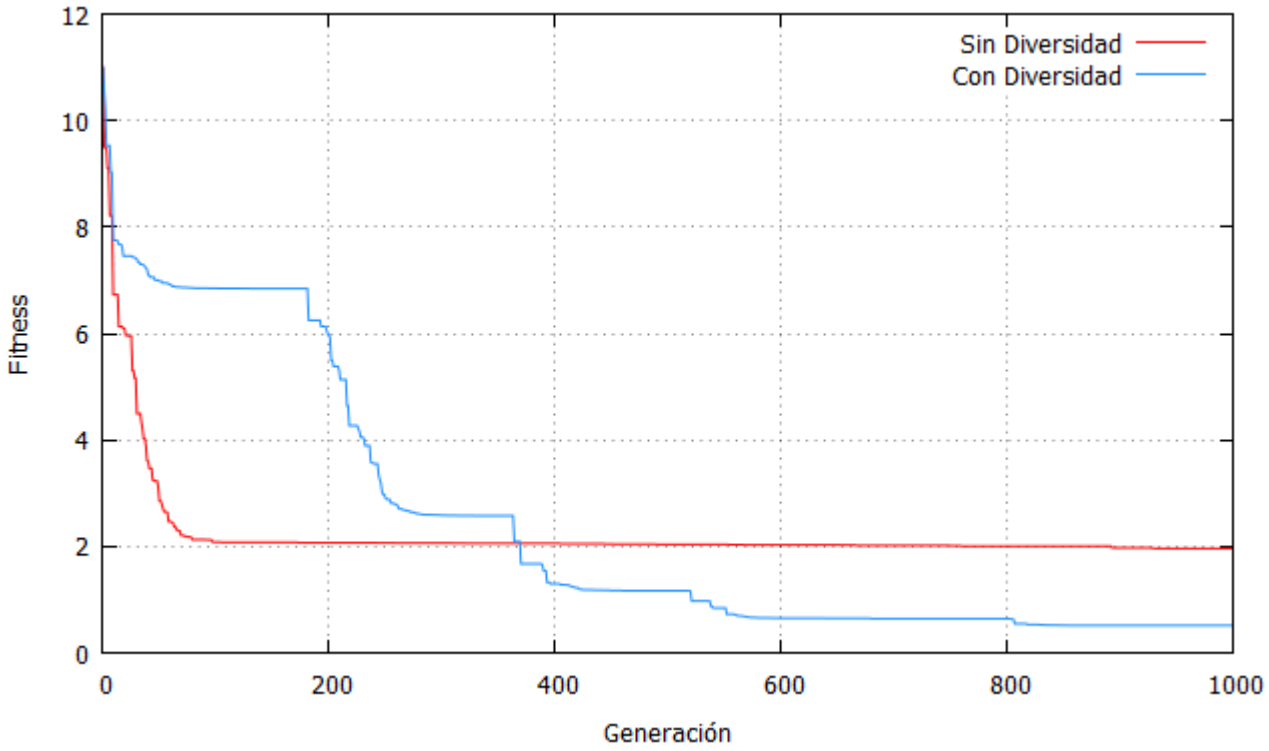


Figura 4.10: Gráfica de convergencia del algoritmo con y sin aplicación de diversidad

4.6. Diversidad

En la Figura 4.10 se muestra la comparativa de convergencia con el uso y sin el uso de diversidad, estas representaciones corresponden a la mejor corrida de 20 ejecuciones.

Se puede observar que al momento de aplicar diversidad se puede lograr una mejor convergencia.

En la gráfica se muestra en color rojo, que corresponde a la ejecución sin diversidad, como el fitness bajo a un buen nivel pero se quedó estancado y no mejoró con el tiempo. En contraste, en la representación azul que corresponde a la aplicación de diversidad, se muestra como tras cada estancamiento, baja el fitness de manea súbita cuando se realiza la aplicación de diversidad. ó

Conclusiones

Las conclusiones finales se muestran a continuación.

En análisis de NVPROF se puede observar cómo se distribuyen las funciones a través de los diferentes Kernel, esto es importante ya que indica que porcentaje de cómputo está siendo utilizado por determinada función.

Las operaciones en paralelo tardan una cantidad mucho mayor en comparación con las que contienen procesos secuenciales, tal es el caso de la comparativa de la función de evaluación de la población que ocupa la mayor parte del proceso de cómputo al no ser una función totalmente paralelizable.

En contraste, la función de comparación, que de ejecutarse secuencialmente sería la de más costo computacional, se logró paralelizar a tal grado que la comparativa se hace por medio de una cantidad masiva de hilos, lo que perite reducir el tiempo de cómputo a una cantidad muy baja.

Estas comparativas indican que la implementación en el GPU se está realizando las instrucciones de manera paralela, además, que las estrategias utilizadas permitieron que el valor del fitness y las mediciones de tiempo fueron mejoradas.

Los métodos utilizados con el modelo de programación paralela permitieron que se obtuvieran resultados muy parecidos a las mediciones reales, en las gráficas se puede observar como aunque tienen variaciones estas son mínimas, además muestra buenos resultados para ser una instancia pequeña (128 individuos y 1000 generaciones).

La variación mostrada en las representaciones puede dar una buena aproximación de la eficiencia real del algoritmo, además los valores del fitness y MSE se mantienen constantes en instancias de hasta 10 genes, sin embargo, en instancias más grandes tiene oportunidad de mejora utilizando parámetros más completos, como una cantidad mayor de individuos o más generaciones.

Al generar la estructura de la red también se obtuvieron discrepancias, sin embargo fue posible obtener la estructura real. Otro punto de mejora será utilizar parámetros más acordes a las instancias para reducir el número de variaciones.

Aunque los procesos de la evoluciona diferencial se adaptaron al modelo de programación

paralela individualmente, la manera de medir su eficiencia tiene que evaluarse con todos ellos trabajando en conjunto.

Los resultados obtenidos permiten determinar que cada uno de los procesos de la evolución diferencial, como todos trabajando en conjunto se ejecutan de manera eficiente en paralelo gracias al modelo de programación.

La sola implementación de búsqueda local no sería suficiente si esta no implicara un impacto sustancial en la eficacia del algoritmo, por lo que las comparativas de calidad en las soluciones permitirían medir que tan buena es la implementación.

Al principio se agregó una búsqueda local muy básica, que no tuvo este impacto esperado, aunque mejoraba ligeramente la solución, tenía mucho margen de mejora, además el estado del arte proponía métodos basados en heurísticas mucho más funcionales y precisos con mejores probabilidades de dar buenos resultados.

Sin embargo, la implementación del algoritmo PSO como búsqueda local implicó una gran mejora al comportamiento de la aplicación, con diversas pruebas se obtuvo la variación que tuvo mejor resultado para ser implementada.

La búsqueda local permitió mejorar los resultados ya que el menor fitness fue mejorado al agregarlo a la implementación.

Las comparativas con el trabajo de Sirbu [Alina Sîrbu, 2010] indican que el desempeño del algoritmo es tan bueno como cualquiera de los métodos utilizados del framework para la comparativa, con margen de mejora al incluir parámetros más robustos al utilizar instancias con más cantidad de genes.

5.1. Discusión

El uso de GPU puede ayudar a acelerar los procesos utilizados en la evolución diferencial gracias a su modelo de programación, lo cual hace especialmente útil este tipo de hardware para resolver procesos iterativos que requiere un gran poder de cómputo. Gracias a esto es posible reducir el tiempo requerido para obtener soluciones de buena calidad, tan buena como los obtenidos con métodos más comunes.

Una de las características más favorables es que al ser un lenguaje unificado, un algoritmo escrito para CUDA puede correr tanto en una tarjeta gráfica sencilla como en una especialmente diseñada para cómputo paralelo, aprovechando al máximo las capacidades del *Device* donde se

esté ejecutando sin realizar modificaciones importantes.

La mayor limitación del uso de GPU es la memoria de la tarjeta gráfica. En las pruebas realizadas una instancia de 10 genes y 512 individuos puede ser ejecutada sin problemas en el equipo de desarrollo, pero la misma instancia utilizando 1024 individuos comienza a dar problemas de memoria en la tarjeta, sin embargo en el equipo de pruebas, se pueden correr instancias de 50 genes y 2048 individuos sin problema.

5.2. Trabajo futuro

Extender las opciones de variantes de evolución diferencial, como *DE/best/2/bin*, *DE/rand/1/exp* etc, con la posibilidad de elegir la variante como un parámetro más del algoritmo.

Para la implementación de búsqueda local sería interesante poder combinar instancias independientes en cualquier momento de la evolución diferencial para modificar todas las poblaciones y compartiendo características totalmente ajenas a las mismas.

Optimizar los parámetros en instancias de más de 20 genes, aunque es posible analizar este tamaño de instancias, es un proceso muy lento debido a las capacidades técnicas de los equipos de prueba.

Buscar un remplazo para el método numérico de Runge-Kutta como método de evaluación de la ecuación diferencial que calcula los niveles de expresión. Aunque es un método sumamente funcional, en este método existe una dependencia del tiempo actual para el cálculo del siguiente tiempo por lo que no se pueden evaluar todos los tiempos a la vez como se pretendía.

No se pudieron aprovechar las nuevas herramientas que ofrece la arquitectura mas reciente de CUDA debido a las limitaciones del Hardware con el que se realizaron las pruebas, éstos equipos contaban con una arquitectura de GPU que no soportaba las funciones incluidas en la ultima versión que podrían tener una gran utilidad para el problema. Tal es el caso de poder ejecutar Kernel sobre Kernel. Esta función podría agilizar la ejecución de los métodos de evaluación y otras áreas de oportunidad como el cálculo de números aleatorios.

Anexos

6.1. Bioinformática

Todos los seres vivos están compuestos por cuatro elementos básicos, Adenina, Citosina, Guanina y Tiamina, estos son llamados nucleótidos y componen el ADN. Las secuencias de ADN y ARN codifican información para producir las proteínas y moléculas de un organismo. Esta información se encuentra codificada en cadenas de ADN denominadas cromosomas. Los cromosomas están compuestos por moléculas llamadas genes, los cuales contienen la información necesaria para construir una molécula de ARN o proteína. La Figura 6.1 muestra una representación común de cadenas de ADN y ARN

La bioinformática es un área de investigación interdisciplinaria que engloba las ciencias de la computación y las ciencias biológicas, se puede definir como la unión de la tecnología computacional para guardar, distribuir, manipular y recuperar información asociada a las macromoléculas de ADN, ARN y proteínas. El análisis de datos genómicos depende de tareas repetitivas y de alta complejidad matemática, por lo que el uso de computadoras es ideal para este tipo de análisis.

A diferencia de la biología computacional, la bioinformática se enfoca en el análisis de secuencia, estructura y funcionamiento de los genes, genomas y sus derivados. Su objetivo es entender el funcionamiento de las células a nivel molecular analizando secuencias y estructuras para generar

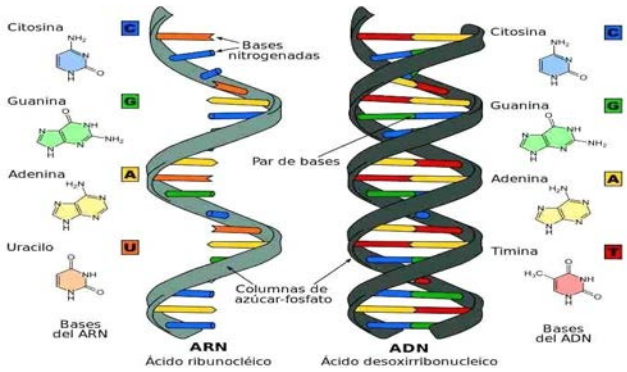


Figura 6.1: Representación de las cadenas de ARN y ADN, además de los nucleídos que componen cada una de las cadenas.

una perspectiva global de la célula.

La razón de que sea posible el entendimiento de estas secuencias de datos es gracias al dogma de la biología molecular para el flujo de información genética, el cual dicta que el ADN es transcrito en ARN, el cual se transforma en proteínas, las funciones celulares serán determinadas según estas secuencias.

La bioinformática tiene un gran impacto en varias áreas de biotecnología y ciencias biomédicas. Los enfoques basados en bioinformática permiten reducir el tiempo y costo necesario para el desarrollo de medicamentos además de reducción de efectos secundarios al evitar el uso de prueba y error, incluso permite el desarrollo de medicación personalizada, mencionando otra utilidad, el análisis forense de ADN es bien aceptado al presentarlo como evidencia criminal o análisis de identidad. Otra de las aplicaciones es para el análisis de genoma para detección temprana de mutaciones peligrosas y tratamiento efectivo de enfermedades. También tiene utilidad en la agricultura para creación de nuevas variedades de cultivos y diseño inteligente de los mismos para hacerlos más resistentes a plagas y condiciones ambientales.

La experimentación y la bioinformática son actividades independientes pero complementarias, ya que calidad de los resultados depende en gran medida de la calidad de los datos de análisis obtenidos con la experimentación, además de la calidad y complejidad del algoritmo utilizado. Es importante señalar que las predicciones de la bioinformática no remplaza totalmente a la experimentación tradicional. [Xiong, 2006]

6.1.1. Áreas y campos de bioinformática

Existen dos campos básicos para la bioinformática:

- El desarrollo de herramientas computacionales y bases de datos
- Aplicación de herramientas computacionales para el mejor entendimiento de organizamos vivos.

Estos campos son complementarios y engloban los tres aspectos de análisis de bioinformática.

- Análisis de secuencia
    - Alineación de secuencias
    - Búsqueda en base de datos

- Descubrimiento de patrones y motivos
- Búsqueda de genes y promotores
- Reconstrucción de relaciones de evolución
- Ensamblaje y comparación de genomas

- Análisis de estructura
  - Análisis de estructura de proteínas y ácidos nucleicos
  - Comparación de estructura de proteínas
  - Clasificación de estructura de proteínas
  - Predicción de estructura de proteínas

- Análisis funcional
  - Perfil de expresión de genes
  - Predicción de interacción de proteínas
  - Predicción de localización de proteínas subcelulares
  - Reconstrucción de camino metabólico
  - Simulación

A continuación se abordaran de manera breve las áreas más importantes en el ámbito de bioinformática.

6.1.2. Bases de datos biológicas

El denominado descubrimiento de conocimiento se define como la identificación de conexiones entre piezas dispersas de información, para este tipo de análisis que tienen la necesidad de manejar enormes cantidades de datos, se requiere una estructuración y organización eficientes para acceder a los valores almacenados. En el ámbito computacional los sistemas de manejo de bases de datos son ideales para este tipo de tareas, ya que proveen una conjunción de Hardware y Software para las tareas descritas anteriormente, es extendido su uso por medio de bases de datos de tipo relacional y orientado a objetos entre otros tipos.

En la Figura 6.2 se muestra a manera de diagrama las ramas de las bases de datos biológicas.

Según su contenido, se pueden dividir en tres categorías principales [Xiong, 2006]:

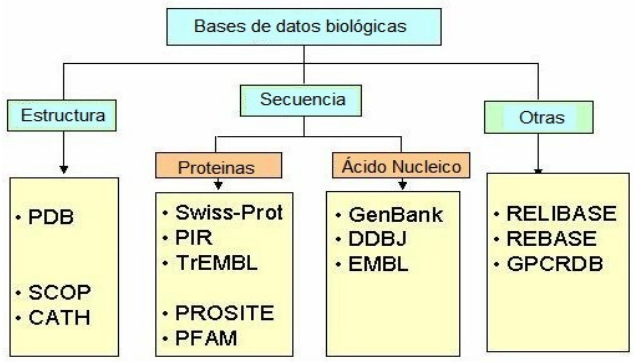


Figura 6.2: Ejemplo de bases de datos biológicas organizadas según el tipo básico al que pertenecen.

- Bases de datos primarias.

  Contienen datos biológicos de estructuras como datos en bruto o secuencias completas compartidas por la comunidad científica. Un ejemplo de este tipo de bases de datos es un banco de genes (GenBank) o el banco de datos de proteínas (PDB)).

- Bases de datos secundarias.

  Albergan información procesada procedente de las bases de datos primarias, ya sea manualmente o de forma computacional.

- Bases de datos Especializadas.

  Contienen información específica y particular de interés de investigación, un ejemplo es las bases de datos de VIH, o datos de estudio de un organismo en particular.

6.1.3. Alineación de secuencias

Las proteínas y el ADN son productos de la evolución. La construcción de bloques de macromoléculas biológicas, nucleótidos base y aminoácidos forman secuencias lineales que determinan la estructura primaria de las moléculas. Estas gradualmente acumulan mutaciones y trazos de la evolución, por lo que la comparación de porciones de estas secuencias permite la identificación de ancestros comunes ya que algunos residuos se mantienen por la selección natural. Por esta razón estas moléculas son consideradas fósiles moleculares y codifican millones de años de evolución.

En la Figura 6.3 se puede observar un ejemplo de esta relación, donde diferentes variaciones de la proteína coincide al alinear la secuencia que la compone.

Identificar las relaciones evolutivas entre secuencias es de ayuda para conocer la función de secuencias no conocidas. Cuando la alineación de secuencia tiene similitudes relevantes con un

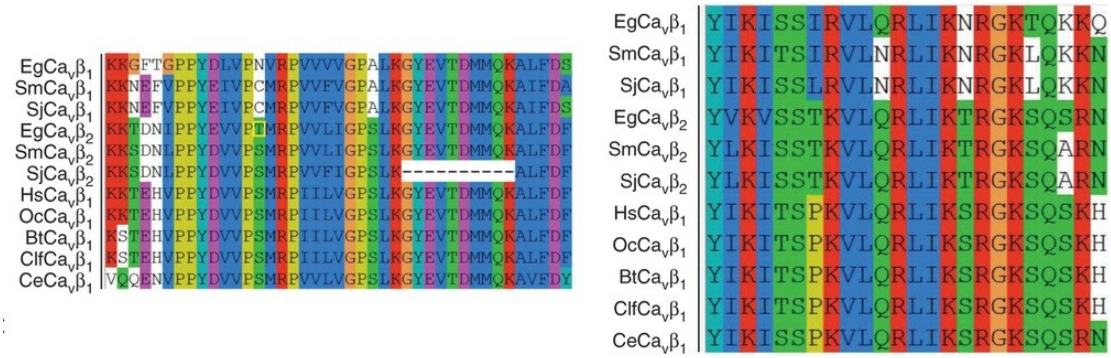


Figura 6.3: Porción del análisis de alineación de secuencias un diferentes organizamos del ácido ribosomal proteína P0.

grupo de secuencias, esta es considerada de la misma familia.

La alineación de secuencias provee la inferencia entre dos secuencias comparadas, la similitud de las mismas determina si tienen un origen evolutivo común, por lo que puede ser utilizada para la predicción básica de la estructura y función de secuencias desconocidas

6.1.4. Predicción de genes y promotores

La predicción de genes es un requisito para el análisis detallado del funcionamiento de genes y genomas, incluye detectar la ubicación del marco abierto de lectura (ORFs) y delineación de las estructuras de Intrones y Exones. Debido a la cantidad de información presente en las secuencias de genoma es imprescindible el uso de potencia computacional para una predicción precisa de la estructura genética.

Esta predicción representa uno de los problemas con más dificultad en el campo de reconocimiento de patrones, debido a que las regiones codificadas no tienden a presentarse en regiones similares, además es muy complicada la detección de regiones potencialmente útiles con características favorables para el análisis, sin embargo, los algoritmos utilizados en este campo suelen dar resultados satisfactorios. Se pueden clasificar en dos categorías mayores, se presentan a continuación.

- Basados en ab-initio.

  Predice los genes basándose en una sola secuencia. Su efectividad es gracias que se basa en dos cualidades características asociadas a los genes.

  - La existencia de señales en los genes. Esto incluye condiciones de inicio y termino,

señales de empalme de Introns, factor de transcripción a sitios de unión, sitios de unión ribosomal y sitios de poliadenilación (poli-A).

- Contenido de los genes. Es la descripción estadística de las regiones codificadas, se puede observar la composición de nucleótidos y estadísticas y patrones de otras áreas significativas.

- Basados en enfoque en homología.

  Se basa en uniones significativas de la secuencia de análisis para, en comparación con secuencias similares o conocidas, realizar las predicciones. Por ejemplo si al traducir una secuencia de ADN esta es similar a una proteína conocida (o de la misma familia), esta es una evidencia fuerte de que esa región codifica la proteína.

6.1.5. Filogenética molecular

En el contexto biológico, la evolución se puede definir como el desarrollo de organismos biológicos a partir de otras formas prexistentes o a partir de la selección natural y otras modificaciones y mutaciones.

La filogenética es el estudio del historial evolutivo de los organizamos vivos usando diagramas de tipo árbol para representar la genealogía de esos organizamos, este tipo de representación se puede observar en la Figura 6.4.

El análisis de secuencias biológicas se fundamenta en los principios de evolución, similitudes y diferencias entre secuencias biológicas con relación entre sí.

Por ejemplo al analizar restos fósiles, estos contienen información de la morfología de los ancestros de algunas de las especies actuales. En contraste, para el análisis de microrganismos es posible obtener una perspectiva del historial evolutivo del mismo mediante el análisis de las secuencias de proteínas de ADN debido a que el material genético con el tiempo acumula las mutaciones causantes de los cambios. Al comparar este análisis del *fósil molecular* con respecto a organizamos relacionados, se puede obtener una perspectiva del historial evolutivo.

6.1.6. Bioinformática estructural

Las proteínas realizan las funciones biológicas y químicas esenciales en la célula, son indispensables para las funciones de regulación, transporte, enzimáticas y estructurales. Su funcionamiento esta determinado por la estructura de la misma.

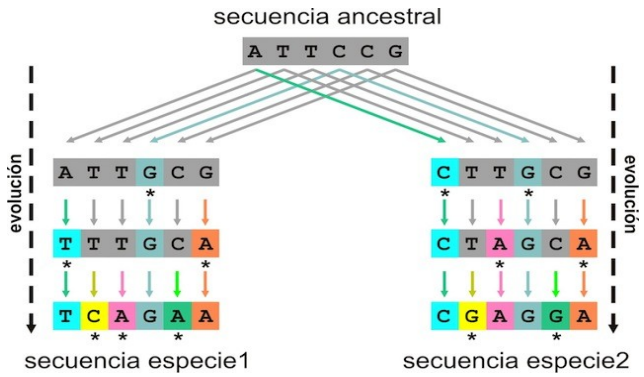


Figura 6.4: Ejemplo de obtención mediante Filogenia de la secuencia ancestral común de dos especias relacionadas.

Por ejemplo, existen 20 aminoácidos de origen natural, estas moléculas contienen un grupo amino libre (NH_2) y un grupo carboxil libre (COOH). Cada uno de estos grupos están unidos a un carbono central (Cα) y al lado de una cadena grupo (R).

Esta es la única característica que los hace diferenciables, las propiedades específicas del aminoácido son determinadas gracias a los reactivos químicos del grupo (R).

Las estructuras de las proteínas se pueden clasificar en 4 niveles:

- Estructuras Primarias.

  Estructura de la secuencia de residuos de aminoácidos

- Estructuras Secundarias.

  Corresponde al plegamiento regular entre residuos de aminoácidos cercanos.

- Estructuras Terciarias.

  Conformación tridimensional de la cadena de poli-péptidos.

- Estructuras Cuaternarias.

  Arreglo complejo de múltiples cadenas poli-peptídicas.

6.1.7. Redes Reguladoras de Genes

La mayor meta de la revolución genómica es el entendimiento de las causas genéticas detrás de las características fenotípicas de los organismos. Las tecnologías y métodos actuales para el análisis de redes de genes hacen posible esta meta al identificar interacciones entre genes en organismos vivos.

Los modelos de redes más simples son sistemas complejos que involucran muchos parámetros, afortunadamente, existe una gran variedad de enfoques para ser modeladas y simuladas. Cada uno de estos busca adaptar los parámetros necesarios a los datos, esto no es una tarea trivial y es conocido como inferencia en la red, identificación de la red, o ingeniería inversa.

6.1.8. Red de genes

Una red de genes generalmente es definido como un número de procesos secuenciales conocidos comúnmente como transcripción y traducción, que controlan los niveles de expresión de los genes y como último resultado la cantidad específica de una proteína objetivo.

Un sistema de regulación de genes consiste en genes, elementos CIS y reguladores. Los reguladores a menudo son proteínas llamadas factores de transcripción, pero en ocasiones pequeñas moléculas como el ARN y metabolitos pueden participar en la regulación. Las interacciones y vinculaciones de reguladores a elementos CIS en las regiones CIS de los genes, controlan el nivel de expresión durante la transcripción. Las regiones CIS sirven para agregar las señales de entrada, mediante los reguladores. En la Figura 6.5 se muestra una representaciones de los niveles de inferencia y las regiones CIS y cómo interactúan con las proteínas.

Dependiendo del grado de abstracción y disponibilidad de los datos empíricos, existen diferentes niveles al modelar redes reguladoras de genes, esto dependerá del conocimiento biológico y los datos disponibles, además de la meta del experimento que puede ser la prueba de una hipótesis simple o un modelo complejo de una red cuantitativa.

6.1.9. Propiedades biológicas

Son un conjunto de factores y reglas de diseño que meramente son empíricos, pero gracias al conocimiento actual acerca de las redes reguladoras de genes, son de utilidad para modelar de manera correcta una red de este tipo.

Topología

La topología de una red define las conexiones entre los nodos, y pueden ser el punto de partida para el modelado. Uno de las reglas más poderosas e importantes es que las redes son muy dispersas, es decir que tiene una cantidad de conexiones por nodo menor que la cantidad total de nodos.

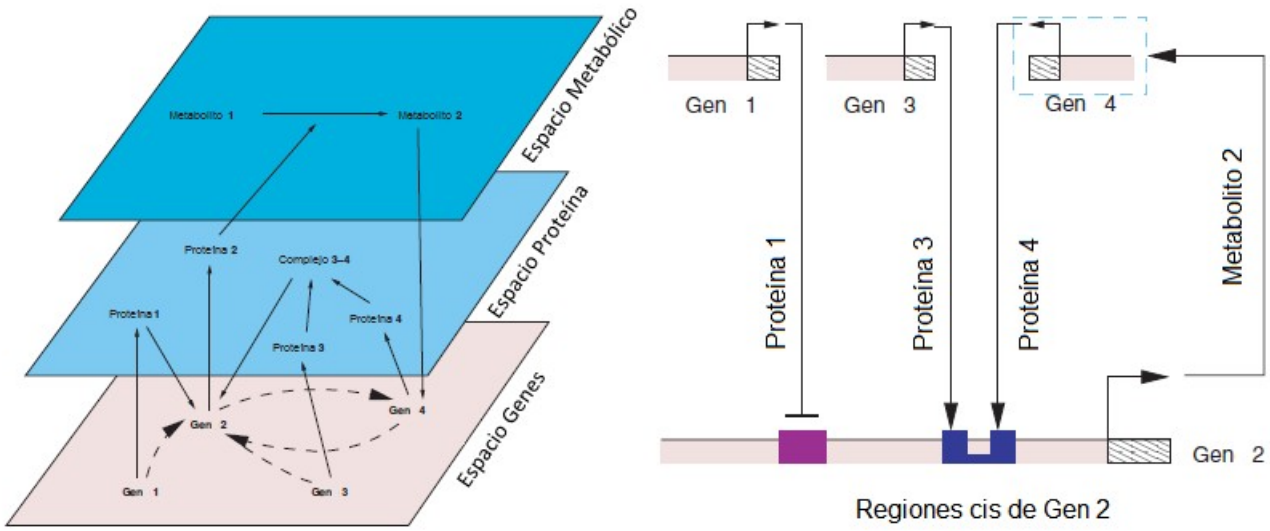


Figura 6.5: A la izquierda, diferentes niveles de inferencia en una red reguladora de genes, a la derecha, representación de las regiones CIS.

Control transcripcional

Las regiones CIS sirven como agregadores de los efectos de los factores de transcripción relacionales en la regulación de los genes. A través de proteínas preparan sitios de las regiones CIS para unir grupos de TFs que tienen propiedades específicas de regulación con el propósito de indicar donde, cuando y en que porcentaje se expresará el gen.

Robustez

Las redes de genes son muy robustas a la variación de valores en sus parámetros y esta es un fuerte indicador de que solo topologías específicas pueden garantizar este comportamiento. La insensibilidad a variaciones en concentraciones moleculares es particularmente importante durante el desarrollo del organismo, una gran cantidad de cambios suceden al mismo tiempo bajo diferentes condiciones, sin embargo, cada vez el resultado al terminar de desarrollarse es el mismo.

Ruido

El ruido es una parte integral de una red de genes, de este emergen propiedades de reacciones bioquímicas que son estocásticos por naturaleza. Cada pequeña variación en la concentración

molecular durante el proceso de transaccional, puede ser adjunto a la red. Las redes controlan el ruido mediante retroalimentación, aunque en algunos casos el ruido mejora algunas características funcionales de las redes y por lo tanto, pueden tener un rol relevante en la evolución de la misma.

6.1.10. Utilidad

Una red de genes, a cualquier nivel de modelación, es un modelo para la comprensión de la funcionalidad cooperativa de los genes.

De manera individual, las redes son representaciones sucintas del conocimiento del sistema estudiado. Por lo tanto pueden utilizarse para clasificación basado en la localización de los genes más influyentes y sobre los que ejercen esa influencia, o en otro caso, para entender cascadas de redes de regulación mediante un análisis a mayor escala.

En términos cuantitativos, se pueden utilizar para obtener varios escenarios y predecir el comportamiento futuro del sistema. Conocer las interacciones entre los componentes puede ayudar a identificar moléculas específicas para medicamentos o entender el comportamiento de la red para generar curas potenciales a enfermedades e incluso, medicación personalizada. Los diferentes efectos en diferentes organizamos pueden ser atribuidos a diferencias en este tipo de redes.

6.1.11. Análisis y Datos

La cantidad de ARN mensajero producido durante la transcripción es una medida que representa lo activo o funcional que es un gen. Chips de genes o micro arreglos son tecnologías de monitoreo de expresiones de genes de larga escala que se utilizan para detectar diferencias en el nivel del ARNm en miles de genes al mismo tiempo, lo que permite acelerar de manera sustancial los estudios funcionales a nivel genoma. Los micro arreglos son usados para identificar las diferencias entre dos experimentos en los niveles de expresión de los genes y su similitud en múltiples experimentos.

Los micro arreglos se pueden clasificar en dos grupos:

- Experimentos sobre tiempo.

  Observan los cambios de la expresión de los genes en el tiempo y se utiliza para entender las variaciones de los procesos de la célula con el tiempo.

- Experimentos de perturbación.

Identifican los efectos de un cambio o tratamiento en la célula, son utilizados para observar los efectos que se producen y de esta manera entender las causas genéticas.

6.1.12. Propiedades de formalismo de modelado

El formalismo de modelado se puede elegir dependiendo de varias cuestiones, por ejemplo el tipo y cantidad de datos disponibles, el conocimiento previo de las interacciones de la red, la naturaleza del estudio y los recursos computacionales y experimentales entre otros.

Estos modelos se pueden clasificar en los siguientes grupos:

- Modelos Físicos contra Combinatorios.

  Los modelos complejos de sistemas dinámicos, como las redes de genes, se basan en ecuaciones diferenciales para describir las relaciones cuantitativas entre las variables de estado en el sistema.

  - Modelos Físicos.

    Se usan para correr simulaciones y predecir el comportamiento del sistema.

  - Modelos Combinatorios.

    Comienzan con las características de alto nivel del sistema para definir otras características de interés.

- Modelos Dinámicos contra Estáticos.

  - Modelos Dinámicos.

    Describen los cambios de los niveles de expresión a lo largo del tiempo. Normalmente cada nodo en la red es una función que calcula la salida en base a la agrupación de las entradas. Tienden a ser más completos que los modelos estáticos, en ellos se pueden modelar las interacciones entre las señales de entrada y ofrecen predicciones cuantitativas para las variables observables, sin embargo, requieren de una cantidad mayor de entradas.

  - Modelos Estáticos.

    Son útiles para revelar las interacciones combinatorias de los genes de manera más simplificada a comparación de los modelos dinámicos. Un ejemplo de este tipo de modelos son los basados en Redes Bayesianas y los modelos aditivos lineales.

- Modelos Determinísticos contra Estocásticos.

  - Modelos Determinísticos.

    En este tipo de modelos los estados de la expresión de los genes son dados mediante una fórmula o pertenecen a determinadas clases. Son medidos en tiempos o lugares diferentes manteniendo los otros parámetros con el objetivo de que la medida de los niveles de expresión sea la misma. La precisión de este tipo de modelos esta determinada por la configuración del experimento y la de la tecnología utilizada para las mediciones.

  - Modelos Estocásticos.

    Asume que los valores de expresión de los genes se describen a través de variables aleatorias con probabilidades de distribución. A diferencia de los modelos determinísticos, la aleatoriedad es modelada a través de los procesos intrínsecos observados, lo que indica que en diferentes ocasiones la expresión de los genes puede variar.

6.1.13. Modelos basados en Teoría de grafos

Son utilizados para describir la arquitectura o la topología de una red de genes. Estos modelos describen las relaciones entre los genes y posiblemente su naturaleza, sin embargo no es posible describir en ellos representaciones dinámicas, como lo es su variabilidad en tiempo. En la Figura 6.6 se observa la representación de cinco estructuras diferentes correspondientes a una red de cinco genes.

Este tipo de modelos, las redes de genes se representan con un grafo de estructura $G(V, E)$ donde $V = \{1, 2, ..., n\}$, representa los elementos de la red de genes, como, genes y proteínas y $E = \{(i, j)|i, j \in V\}$ Indica las interacciones entre ellos, por ejemplo, activación, inhibición, causalidad, etc.

G es un grafo simple y sus bordes representan las relaciones con los nodos. Esta representación es de suma utilidad ya que permite resolver algunas incógnitas de la red de genes usando métodos y algoritmos para grafos.

La inferencia en redes de genes bajo modelos de teoría de grafos pueden identificar los bordes y sus parámetros para los datos de expresión analizados, algunos de los conceptos más importantes de enumeran a continuación.

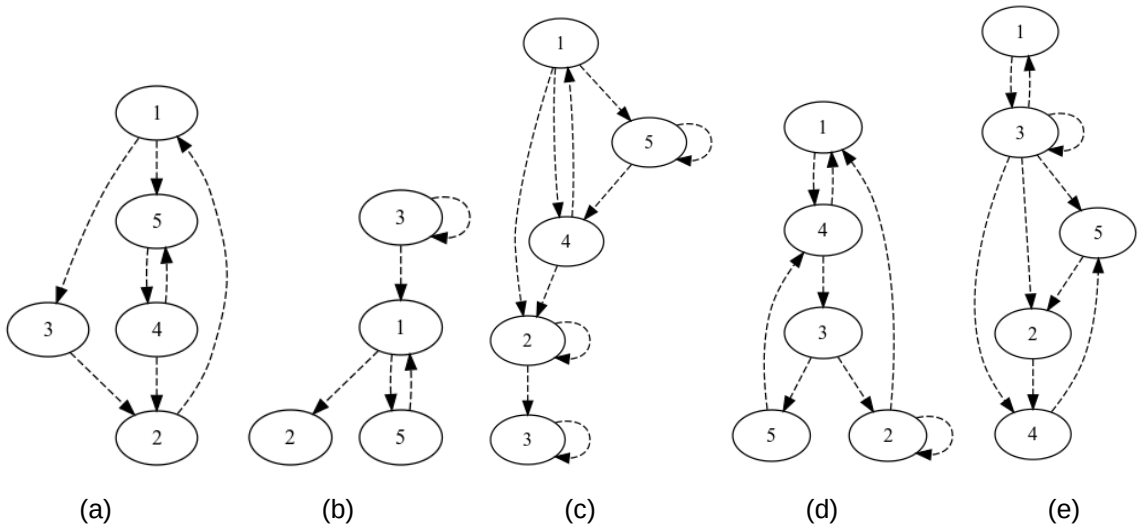


Figura 6.6: Variaciones de la estructura de una red de cinco genes representada por grafos.

Inferencia en co-expresión de redes

Las redes de co-expresión son grafos donde los bordes conectan nodos de co-expresión. Los datos pueden ser conjuntos de datos como mediciones y datos de perturbación. La meta de este enfoque es obtener un grafo donde los nodos son clúster de genes con perfiles de expresión indistinguibles y las conexiones entre los nodos indican similitudes entre ellos.

Interacciones de regulación en datos de series de tiempo

EL objetivo es indicar cuales bordes del grafo son candidatos potenciales para regular las relaciones entre genes. La idea consiste en considerar la expresión de los genes del experimento para durante el transcurso del tiempo co-relacionar los cambios positivos y negativos en los niveles de expresión incorporando consideraciones biológicas.

Redes causales en experimentos de perturbación

La perturbación de un gen en la red causa un efecto en cadena, llegando o no a afectar a todos los genes pertenecientes a la misma. La medición de estas afectaciones a los genes permite deducir las relaciones de los mismos. El problema de inferencia consiste en buscar una red consistente cuyos datos de expresión sean resultado de perturbaciones genéticas.

6.1.14. Redes Bayesianas

Las redes Bayesianas son una clase de modelos probabilistas gráficos, en ellas se combinan las áreas matemáticas de probabilidad y teoría de grafos. Una red Bayesiana consiste en un grafo a cíclico directo $G(V, E)$ donde los nodos $x_i \in X$ son variables aleatorias que representan la expresión de los genes y los bordes indican las dependencias entre los nodos. Las variables aleatorias se generan como distribuciones de condiciones probabilísticas $P(x_i|Pa(x_i))$, donde $Pa(x_i)$ es el conjunto de parámetros por cada nodo.

Cada red Bayesiana únicamente específica una descomposición de la distribución conjunta de todas las variables bajo la distribución condicional de los nodos:

$$P(x_1, x_2, ..., x_n) = \prod_{i=1}^{n} P(x_i|Pa(x_i)) \tag{6.1}$$

Este modelo probabilístico es muy utilizado para modelar relaciones causales debido a que la distribución de la probabilidad de eventos está determinada por la probabilidad de otros eventos.

6.1.15. Redes Booleanas

Son modelos dinámicos de interacciones sincrónicas entre nodos en una red. Son la representación más simple de las propiedades biológicas y sistemáticas de una red de genes real.

Modelo y propiedades

Una red booleana es un grafo directo $G(X, E)$ donde los nodos $x_i \in X$ son de tipo boléanos. Para cada nodo x_i es asociada una función booleana $b_i(x_{i1}, x_{i2}, ..., x_{il})$, $l \leq n, x_{ij} \in X$, donde los argumentos son todos nodos de x_i en G. Para cada tiempo dado los estados de todos los nodos representan el estado de la red con el vector $S(t) = (x_1(t), x_2(t), ..., x_n(t))$. Para las redes de genes, las variables de los nodos corresponden a los niveles de expresión transformados a valores discretos del gen.

Ingeniería Inversa

Su meta es entender cómo se genera la topología y las funciones booleanas de los nodos de los niveles de expresión de los genes observados. La observación parte de medicines a través del tiempo o de experimentos de niveles de expresión. Se miden estos valores en dos experimentos consecutivos que corresponden a dos estados consecutivos de la red.

Dadas las observaciones de los estados de la red booleana, en general algunas redes son encontradas con consistencia de los datos, dando una solución ambigua de la red. Algunas variantes de los problemas de la ingeniería inversa son encontrar una red consistente con los datos, encontrar todas las redes consistentes con los datos y encontrar la mejor red consistente con los datos.

Requerimientos de datos

La cantidad de datos requeridos para determinar una única red es conocida como problema de requerimiento de datos. La cantidad de datos requeridos depende de la dispersión de la topología y el tipo de funciones booleanas. En el peor de los casos este problema de requerimiento de datos se encuentra en el orden de $m = 2^n$ pares de transiciones de datos para inferir en una red densamente conectada con funciones booleanas generales en los nodos.

6.1.16. Linealización y modelos de ED (LAM).

Son el punto de partida para modelar cualitativamente sistemas complejos. Son modelos de formalismos continuos y determinísticos capaces de descubrir fenómenos no lineales y emergentes de sistemas dinámicos complejos.

Se basan en ecuaciones de ritmo, cuantifican la cantidad de cambio en los niveles de expresión de los genes como una función de expresión de otros genes. La forma general de representación para cada uno de n genes es:

$$\frac{dx_i}{dt} = f_i(x_{i1}, x_{i2}, ...x_{il}) \tag{6.2}$$

donde cada x_j es una función continua que representa la expresión del gen$j$.

Cada $fi(.)$ cuantifica el efecto combinado de sus argumentos o reguladores en x_i y esto subscribe los efectos bioquímicos de las interacciones y degradaciones moleculares.

Modelos de aditivos linealizados

Una forma simple de la función $f_i(.)$ donde puede tomar funciones lineales aditivas es:

$$\frac{dx_i(t)}{dt} = ext_i(t) + w_{i1}x_1 + ... + w_{in}x_n(t) \tag{6.3}$$

donde $ext_i(t)$ indica una posible influencia en el gen i (como una perturbación) que es directamente observable.

El activo w_{ij} relaciona el efecto de regulación del gen j a otro gen i y corresponde a la fuerza de este efecto. En la representación por grafos de la red de genes, los genes representan los nodos y los bordes para cada $w_{ij} \neq 0$ nodos padre, son los reguladores. La representación biológica para estos valores esta dada por la condición de si $w_{ij} > 0$, el gen j induce la expresión del gen i y en caso contrario, $w_{ij} < 0$ reprime la transcripción del gen i. En el caso de que $w_{ij} = 0$, se asume que no existe relación entre el gen j y el gen i.

Modelos Basados en Ecuaciones Diferenciales

Un enfoque común es el uso de ecuaciones diferenciales, en este caso particular son de especial utilidad los sistemas tipo S, la ecuación utilizada para determinar los niveles de inferencia se muestra en la siguiente ecuación:

$$\frac{dX_i(t)}{d(t)} = \alpha_i \left(\prod_{j=1}^{N} X_j^{g_{ij}(t)} \right) - \beta_i \left(\prod_{j=1}^{N} X_j^{h_{ij}(t)} \right) \tag{6.4}$$

Donde:

N es el número de genes.

g_{ij} y h_{ij} son los ordenes cinéticos de síntesis y degradación.

α y β son las constates de ritmo.

$X_j(t)$ representa el nivel de expresión del gen j en el tiempo t.

Donde el número de parámetros que deben ser calculados esta determinado por $2N(N+1)$.

Para determinar la aptitud del individuo se utiliza la siguiente expresión:

$$f = \sum_{i=1}^{N} \sum_{t=1}^{T} \left\{ \left(\frac{X_{i,cal,t} - X_{i,exp,t}}{X_{i,exp,t}} \right)^2 \right\} + \lambda \left\{ \sum_{ij}^{N} |g_{ij}| + \sum_{ij}^{N} |h_{ij}| \right\} \tag{6.5}$$

Donde λ controla una penalización en función de que tan alejados de cero se encuentran los ordenes cinéticos, esto es importante ya que una de las propiedades de este tipo de redes es que son muy dispersas, esta parte de la ecuación permite priorizar los resultados que cumplen con esta propiedad.

	Grafos	Bayes	Boléanos	LAM
Nodos	Genes	Variables Aleatorias	Expresión Discretizada	Expresión continua
Bordes	Causalidad	Dependencias condicionales	Argumentos en funciones booleanas	Argumentos en funciones de regulación
Parámetros Adicionales	Activación e Inhibición	Activación e Inhibición	Funciones booleanas	matriz de pesos
Propiedades	Topología estática	Topología estática, Estocástica	Dinámica, Biológicamente realista	Dinámica, Estable realista
Inferencia	Parsimonia biológica y optimización	Puntuación de Bayes, Optimización	Optimización, Teoría de información	Regresión lineal, Optimización
Requerimiento de datos	$O(logn) - O(n)$	NA	$O(2^k klogn)$	$O(knlogn)$

Tabla 6.1: Tabla con las características más representativas de los diferentes modelos para el problema de inferencia en redes reguladoras de genes.

La tabla 6.1 muestra en forma de resumen los modelos presentados anteriormente.

Runge-Kutta

Es un método de evaluación comúnmente utilizado para resolver ecuaciones diferenciales, se puede describir de la siguiente manera:

$$y(x+h) = y(x) + \frac{1}{6}(F_1 + 2F_2 + 2F_3 + F_4) \tag{6.6}$$

donde:

$$F_1 = hf(x, y) \tag{6.7}$$

$$F_2 = hf\left(x + \frac{h}{2}, y + \frac{F_1}{2}\right) \tag{6.8}$$

$$F_3 = hf\left(x + \frac{h}{2}, y + \frac{F_2}{2}\right) \tag{6.9}$$

$$F_4 = hf(x + h, y + F_3) \tag{6.10}$$

Para calcular el tiempo $y(t)$, es necesario conocer $y(t-1)$.

6.2. Computación Evolutiva

Existen diferentes variantes de este tipo de algoritmo, pero es común la idea y las técnicas:

Dada una población de individuos, el ambiente propicia la selección natural, lo que causa un aumento en la aptitud de los individuos de la población.

Una función mide la calidad de la población, esta métrica debe de ser maximizada o minimizada, se crean soluciones candidatas de manera aleatoria y se aplica la medición. Basado en

el resultado se selecciona una proporción de las soluciones candidatas con la opción de aplicar mutación o recombinación [Eiben and Smith., 2003].

La recombinación es un operador aplicado a dos o más candidatos seleccionados.

La mutación es aplicada a un candidato y da como resultado uno nuevo.

Un conjunto de nuevos candidatos con mejores aptitudes, mutación o recombinación son elegidos para pasar a la siguiente generación. Este proceso es iterativo.

En este proceso existen dos conceptos fundamentales:

- Operadores de variación

  Crean la diversidad necesaria, pueden ser de mutación y de recombinación

- Selección

  Para forzar soluciones con calidad

Los procesos de evolución son estocásticos, durante la selección individuos con mejores cualidades tienen una mayor probabilidad de ser seleccionados, pero es común que sean seleccionados también los de menores cualidades, esto en conjunto con mutación y recombinación ayudan a preservar la diversidad.

En la Figura 6.7 se muestra el funcionamiento general de un algoritmo evolutivo, el ciclo de selección, recombinación y mutación se realiza hasta una condición de paro, esto se detallará más adelante.

Resultado:
INICIALIZAR;
EVALUAR;
Mientras *CONDICION DE PARO* **hacer**
    SELECCIÓN;
    RECOMBINACIÓN;
    MUTACIÓN;
    EVALUACIÓN;
    SELECCIÓN;
fin

Algoritmo 11: Esquema general en pseudocódigo de un algoritmo evolutivo.

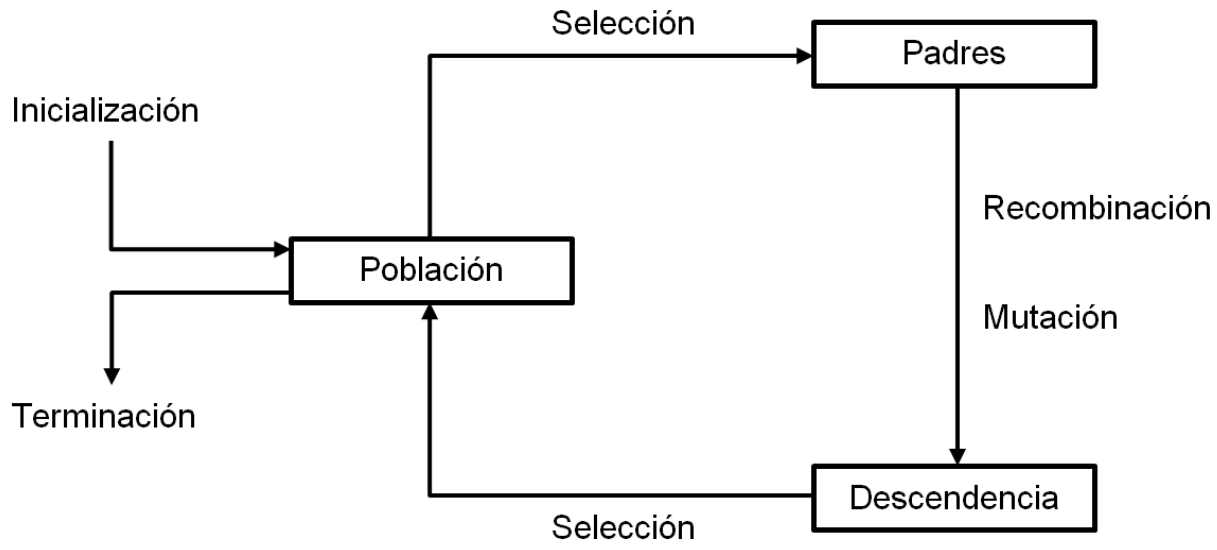


Figura 6.7: Diagrama de flujo general de un algoritmo evolutivo.

Los algoritmos evolutivos tienen un número de componentes, procedimientos y operadores definidos, se detallan a continuación.

6.2.1. Representación.

Objetos forman soluciones posibles, en el contexto del problema de computación evolutiva, estos son denominados fenotipos. El primer paso para este diseño es denominado representación y es un conjunto de genotipos que representan los fenotipos, que pueden ser por ejemplo un conjunto de enteros.

Es importante conocer que en este contexto, el espacio de soluciones de los fenotipos y los genotipos no es el mismo. Una solución de fenotipo es obtenida decodificando al mejor genotipo.

En la terminología común una solución candidato, un fenotipo y un individuo son utilizados para denotar posibles soluciones. Este espacio de soluciones se puede denominar como espacio fenotipo.

6.2.2. Función de evaluación

La función de evaluación es utilizada para representar los requerimientos a adaptar. Representa la tarea a resolver, es una tarea o procedimiento que designa la calidad de una solución del espacio fenotipo. Es comúnmente denominada función de fitness, sin embargo, la computación evolutiva es comúnmente utilizada para problemas de optimización, en cuyo caso, se puede de-

nominar a la función de fitness como función objetivo.

6.2.3. Población

Es la representación de las posibles soluciones. Es un múlti conjunto de genotipos y forma la unidad de la evolución. Los individuos son objetos estáticos que no cambian o se adaptan, es la población la que lo hace.

La población puede ser un conjunto simple de pocos individuos, pero en casos más robustos, puede tener una estructura con las medidas de distancias entre individuos o una relación de vecindario.

6.2.4. Diversidad

Esta es medida con el número de diferentes soluciones presentes. Generalmente se refiere a el número de fitness, fenotipos o genotipos diferentes. En ocasiones también puede contener medidas de entropía.

6.2.5. Mecanismo de selección de padres

Consiste en distinguir individuos basándose en su calidad para elegir los mejores para ser padres de la siguiente generación. Un individuo se considera padre cuando es elegido para crear descendencia.

6.2.6. Operadores de variación

Sirven para crear nuevos individuos a partir de los que ya se encuentran en la población, en el contexto del cómputo evolutivo, sirven para crear nuevas soluciones candidatas del espacio fenotipo.

Estos operadores se pueden dividir en dos categorías:

- Mutación:

  Es una variación unitaria aplicada a un genotipo y genera un hijo o descendencia mutante. Es un operador estocástico que depende de una serie de elecciones aleatorias de características que son modificadas.

- Recombinación:

  La recombinación o cruza, es un operador binario de variación e indica que información de

padres será utilizada. Al igual que la mutación es un operador estocástico donde se hace la elección de las partes de padres a combinar para generar descendencia.

6.2.7. Mecanismo de selección de sobrevivientes.

Al igual que la selección de padres, su tarea es distinguir individuos basándose en su calidad, sin embargo, este se aplica en una etapa diferente del ciclo de evolución, generalmente después de haber creado descendencia.

6.2.8. Inicialización.

La primera población se conforma por individuos generados de manera aleatoria, pero en ocasiones y dependiendo del problema puede ser iniciada por individuos con una alta calidad de fitness.

6.2.9. Condición de paro

Una condición debe ser definida para evitar que el algoritmo nunca pare, algunas de las condiciones más comunes son:

1. Máximo tiempo de CPU
2. Número total de evaluaciones de fitness
3. Determinado tiempo de ejecución

6.2.10. Evolución diferencial

Es un modelo evolutivo que prioriza la mutación aunque también utiliza operadores de cruce y recombinación. Se basa en la evolución de una población generalmente simulada como vectores de valores que representan las soluciones en un espacio de búsqueda.

Entre las principales características que sobreponen al algoritmo de evolución diferencial ante otros algoritmos genéticos, es su rapidez y velocidad de convergencia, es muy eficiente aplicado a problemas reales o de simulación, además, destaca su resistencia a convergencia prematura.

6.2.11. Algoritmos Meméticos

Son métodos basados en población con aplicaciones de búsqueda local, son comúnmente utilizados en problemas de optimización NP para aproximar soluciones. Este tipo de algoritmos es una combinación de conceptos de diferentes heurísticas.

En el contexto de la computación evolutiva una solución es conocida como individuo, sin embargo esto denota un comportamiento pasivo expuesto a modificaciones. En el ámbito de los algoritmos meméticos, es más comúnmente conocido como agentes, ya que al ser su propósito la solución de un problema, pasa a tener un rol activo.

6.2.12. Búsqueda local

Es un proceso iterativo que examina un conjunto de puntos ubicados en el espacio de soluciones y remplaza las soluciones por otras mejores si existen.

El funcionamiento general de este tipo de algoritmos se describe a continuación:

INICIALIZAR;

Mientras *PROFUNDIDAD* **hacer**
  Mientras *CONDICION DE PARO* **hacer**
    GENERAR SIGUIENTE VECINO $j \in n(I)$;
    si $f(j)$ *es mejor que* $f(mejor)$ **entonces**
      REMPLAZAR;
    fin
  fin
fin

Algoritmo 12: Esquema general en pseudocódigo de un algoritmo de búsqueda local.

6.2.13. Algoritmo PSO como búsqueda local

Un algoritmo PSO (Particle swarm optimization) en un algoritmo estocástico basado en población utilizado para solución de problemas de optimización. Los parámetros de control son el coeficiente de aceleración, peso de inercia, velocidad de sujeción y tamaño, se describe con la siguiente formula [F. van den Bergh, 2006].

$$v_{ij}(t+1) = v_{ij}(t) + \phi_{1j}(t)(y_{ij}(t) - x_{ij}(t)) + \phi_{2j}(t)(\hat{y}_{ij}(t) - x_{ij}(t)) \quad (6.11)$$

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1) \quad (6.12)$$

donde:

$\phi_{1j}(t) = c_1 r_{1j}(t)$

$\phi_{2j}(t) = c_2 r_{2j}(t)$

s es el número total de partículas

n es la dimensión del problema

c_1 y c_2 son los coeficientes de aceleración

$r_{1j}(t), r_{2j}(t) \sim U(0,1)$

$x_i(t)$ es la posición de la partícula i en tiempo t

v_i es la velocidad de la partícula i en tiempo t

$y_i(t)$ es la mejor solución de la partícula i en el tiempo t $\hat{y}_i(t)$ es la mejor posición encontrada en el vecindario de la partícula i en el tiempo t

6.3. GPGPU - Cómputo paralelo

La unidad de procesamiento gráfico (GPU), sirve de apoyo a operaciones básicas de un CPU, como renderizar una imagen en memoria y desplegara en la pantalla o realizar cálculos sobre efectos visuales como sombras. Normalmente, los datos dentro de un GPU son representados como un mapa de polígonos en tres dimensiones.

Aunque no fuese diseñado para eso, desde hace tiempo el uso del poder de procesamiento de GPU había sido utilizado por investigadores para cómputo de propósito general, creando iniciativas como BrookGPU, Cg y CTM se ha logrado programar GPU como si de un CPU se tratase, sin embargo, la programación en estos dispositivos era muy complicada y muy específica para cada variante [Cook, 2013].

En años más recientes, el uso de cómputo paralelo se ha propagado ya que el Hardware actual permite una mayor eficiencia al utilizar cómputo paralelo en comparación con métodos secuenciales, apoyado con la creación de librerías que permiten programar las unidades gráficas con mayor facilidad y con métodos ya adaptados de manera eficiente para utilizarse.

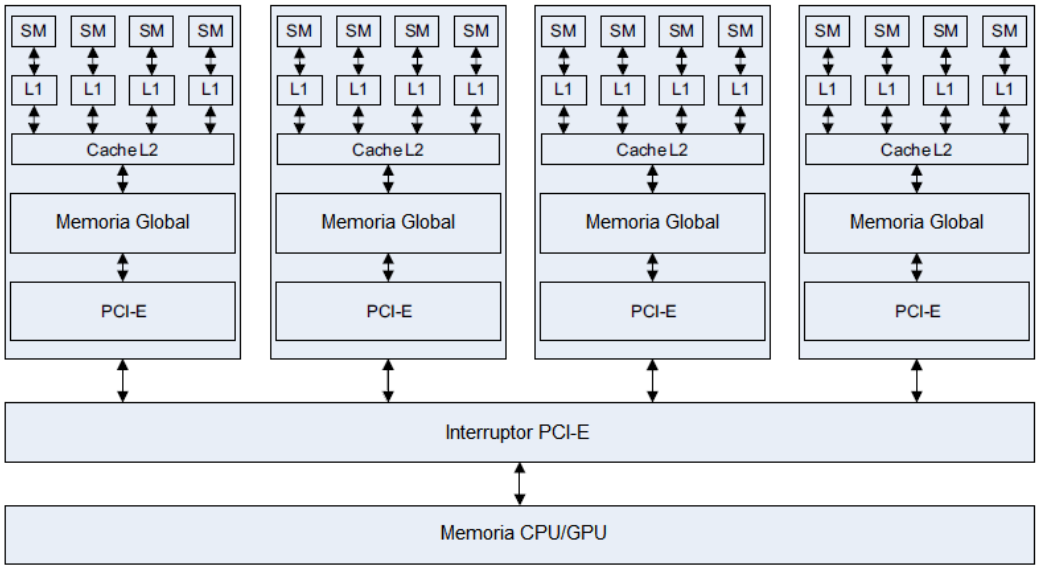


Figura 6.8: Arquitectura genérica de una tarjeta gráfica.

6.3.1. Arquitectura GPU

- Cada tarjeta gráfica consiste en un número de multiprocesadores de transmisión (SMs), son equivalentes a los núcleos de un CPU.
- Estos se encuentran conectados a una memoria compartida (L1 cache).
- Todo está conectado a la memoria de L2 de cache, el cual actúa como un interruptor de los SMs
- Los datos se encuentran en la memoria global de donde se extraen para ser transformados.

La Figura 6.8 representa la arquitectura de una tarjeta gráfica, se puede observar los multiprocesadores comunicándose con la memoria L1, los cuales a su vez se comunican a la memoria Cache L2 en conjunto. Para el manejo de más de un GPU el interruptor PCI-E controla el flujo de información.

6.3.2. Hardware GPU

Cada tarjeta gráfica es un conjunto de SMs, en cada uno de estos existe un conjunto de Procesador de flujo (SP) o núcleos CUDA. Un procesador de flujo ejecuta trabajos en paralelo en conjuntos de 32 unidades, esto elimina los circuitos complejos necesitados en CPU para ejecución serial de alta velocidad para instrucciones con alto nivel de paralelismo.

El rendimiento de un GPU está determinado por el número de SPs presentes, el ancho de banda de la memoria global y la forma de programación paralela utilizada para el algoritmo.

6.3.3. Modelos de programación paralela

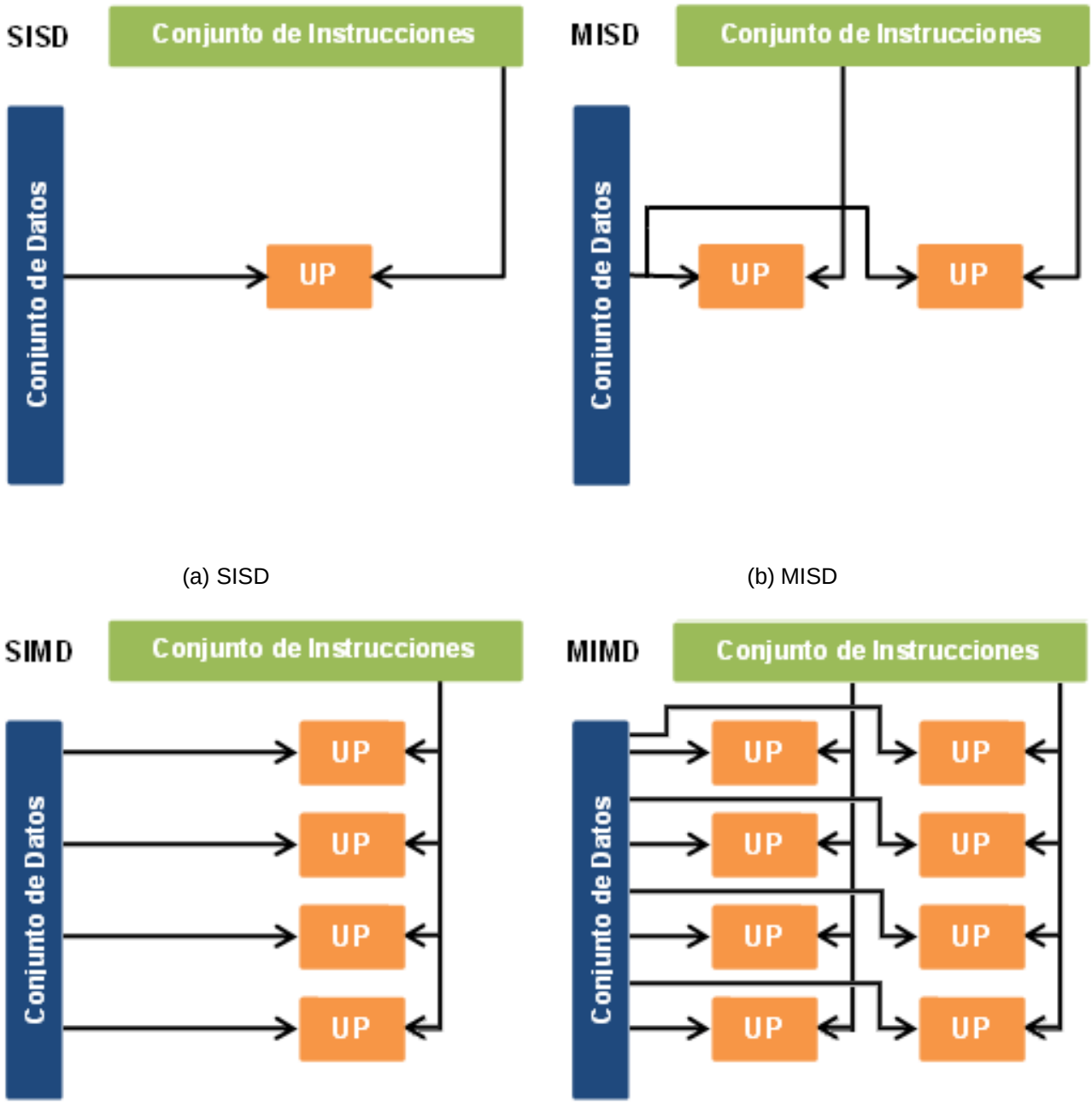
Según la clasificación de taxonomía de Flyn [Flynn, 1972], existen cuatro tipos de arquitecturas paralelas que se diferencian en el número datos sobre los que se aplican y el número de instrucciones que ejecutan:

- SISD (Single Instruction, Single Data)
  En esta clasificación no existe paralelismo, a esta clasificación pertenecen las maquinas secuenciales.
- MISD (Multiple Instruction, Single Data)
  Un conjunto de instrucciones diferentes es aplicado a un solo flujo de datos.
- SIMD (Single Instruction, Multiple Data)
  Una instrucción es ejecutada en un conjunto de datos diferentes de manera simultánea por diferentes procesadores.
- MIMD (Multiple Instruction, Multiple Data)
  Se ejecutan al mismo tiempo diferentes instrucciones sobre diferentes datos. Una aplicación de esta arquitectura es la utilizada en sistemas distribuidos.

En la Figura 6.9 se muestra de manera gráfica el flujo de información en los diferentes modelos de programación.

6.3.4. Librerías para cómputo paralelo en GPU

- OpenCL
  Es un estándar abierto y libre que es soportado por NVIDIA, AMD, Intel etc. Permite su uso en dispositivos de CPU, GPU y otros dispositivos con soporte.
- DirectCompute
  Es una alternativa creada por Microsoft, es propietaria y asociada a sistemas operativos Windows con DirectX API



(a) SISD (b) MISD

(c) SIMD (d) MIMD

Figura 6.9: Representación del flujo de datos y ejecución de instrucciones en las diferentes arquitecturas de clasificación de Fynn.

- MPI

  El paralelismo se da entre un conjunto de nodos que intercambian información a alta velocidad.

- OpenMP

  Es un sistema diseñado para paralelismo entre nodos o sistemas informáticos completos donde el problema se intenta dividir en N partes, de acuerdo al número de núcleos de procesador disponibles.

- Pthreads

  Está diseñado para paralelismo en un único nodo, por lo que divide la tarea en subprocesos, es utilizado comúnmente para aplicaciones múlti-hilos en sistemas Linux.

6.4. CUDA

Compute Unified Device Architecture es un framework creado por NVIDIA para resolver el problema de utilización de unidades de procesamiento gráfico para cómputo general de alto rendimiento, provee herramientas y estructuras que permiten el uso eficiente de la arquitectura de GPU de sus productos.

Es una extensión de lenguaje C, permite que la escritura de código de GPU sea escrito e interpretado en lenguaje C estándar. El CPU (Host) delega tareas múlti hilo en el GPU (Device). En la Figura 6.10 se puede observa una comparación de una función en lenguaje c y la misma implementación con las instrucciones específicas de CUDA.

Esto lo hace una excelente herramienta ya que no es necesario conocer instrucciones complejas para hacer paralelismo, sin embargo, hay que crear modelos de programación que permitan explotar el poder de cómputo paralelo aplicado a las funciones.

Además, existen algunas "*reglas de diseño"* que sugieren utilizar en la documentación de CUDA para hacer más eficiente la implementación de sus funciones paralelas.

A continuación de describen algunas de las herramientas y librerías más importantes de CUDA.

6.4.1. Nsight

CUDA cuenta con la herramienta Profiler que permite un análisis de diversas estadísticas de utilidad, por ejemplo, mide la cantidad de tiempo que tarda en ejecutarse una función en el CPU y

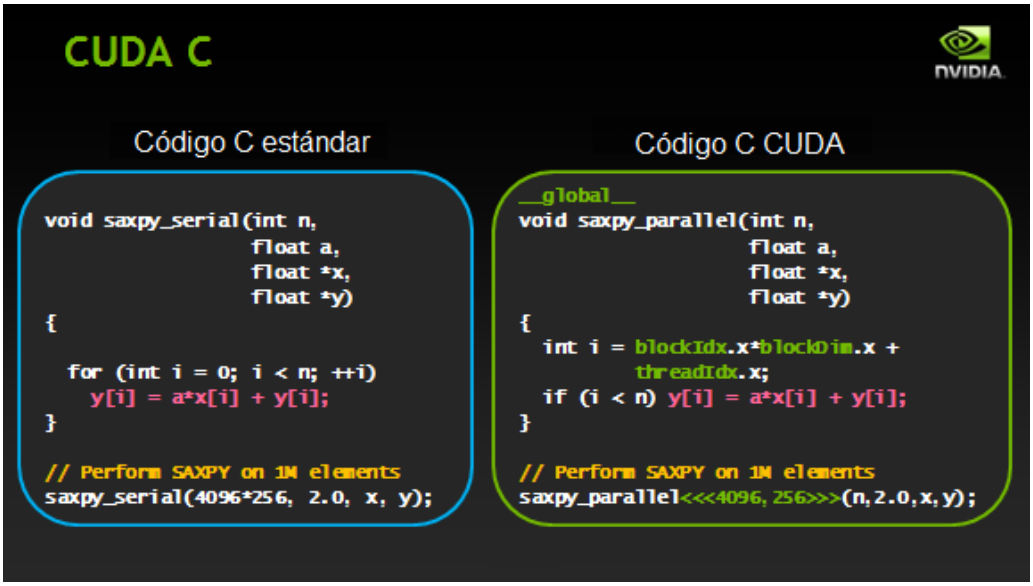


Figura 6.10: Comparación del código estándar utilizado en c y la función paralelizada con CUDA.

GPU, la cantidad de Kernels que son lanzados de forma concurrente, los tiempos de ejecución de cada uno de ellos, entre otras funciones.

6.4.2. Nsight y NV-Profiler

Nsight es un IDE basado en eclipse diseñado para desarrollo en CUDA. Además de las funciones que comúnmente se encuentran en un IDE, Nsight cuenta con la integración de Visual Profiler, el cual muestra a través de una interfaz gráfica la información que recaba con la herramienta Profiler [NVIDIA, 2012]. La Figura 6.11 representa la interfaz que permite la interacción con los diferentes procesos que componen el algoritmo.

De esta manera se puede interactuar con la cantidad de procesos que se ejecutan en paralelo, permite identificar las funciones con colores, muestra los hilos de CPU y GPU y muestra las funciones que se ejecutan por Stream.

Otra de las utilidades de Nsight es que permite ejecutar un análisis guiado donde señala los puntos de mejora al código, uso de memoria etc.

6.4.3. CURAND

Es una librería para generar números aleatorios o pseudoaleatorios utilizando un algoritmo determinista para satisfacer las propiedades de secuencia aleatoria[NVIDIA, 2015].

Está compuesto por dos librerías, una en CPU y otra en GPU. La primera es una librería C

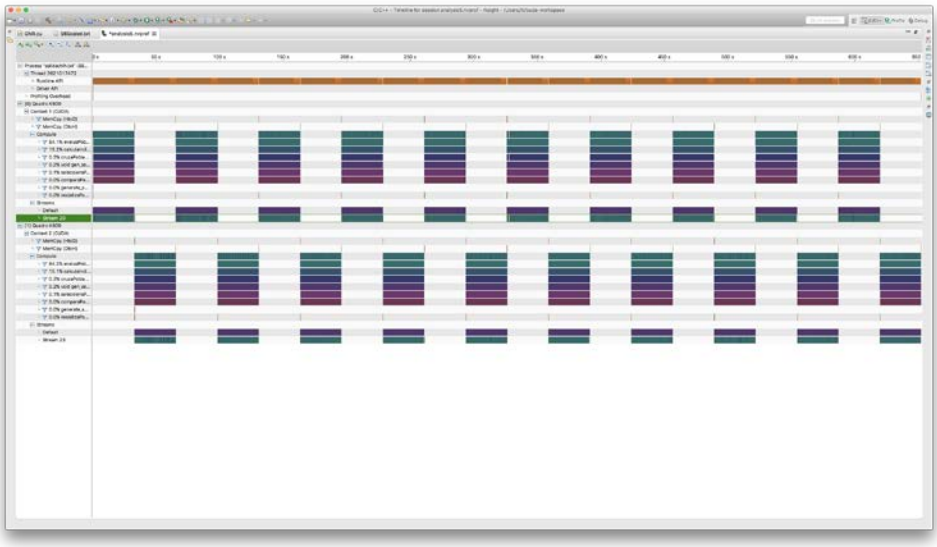


Figura 6.11: Interface de VN-Profiler, se pueden observar en color naranja las interacciones con la memoria y en diferentes colores las funciones que son analizadas.

estándar que utiliza el GPU para la generación de números aleatorios desde el control del CPU, los números generados permanecen en la memoria global de la tarjeta gráfica.

La segunda librería está diseñada para generar números aleatorios en la tarjeta gráfica que puedan ser utilizados en el momento por los Kernel que está ejecutando escribiendo y leyendo desde la memoria global.

Referencias

[Alina Sîrbu, 2010] Alina Sîrbu, Heather J Ruskin, M. C. (2010). Comparison of evolutionary algorithms in gene regulatory network model inference. *BMC Bioinformatics*, pages 11, 59.

[Cook, 2013] Cook, S. (2013). *CUDA Programming A Developer's Guide to Parallel Computing with GPUs.*

[Daisuke TOMINAGA, 1999] Daisuke TOMINAGA, M. O. (1999). Nonlinear numerical optimization technique based on a genetic algorithm for inverse problems: Towards the inference of genetic networks. *DBLP*.

[Eiben and Smith., 2003] Eiben, A. E. and Smith., J. E. (2003). Introduction to evolutionary computing. *SpringerVerlag*.

[F. van den Bergh, 2006] F. van den Bergh, A. E. (2006). A study of particle swarm optimization particle trajectories. *Information Sciences.*, pages 937, 971.

[F. van den Bergh, 2012] F. van den Bergh, A. E. (2012). Performance comparison of local search operators in differential evolution for constrained numerical optimization problems. *Differential Evolution (SDE), 2014 IEEE Symposium on*.

[Fabrizio F. Borelli, 2012] Fabrizio F. Borelli, Raphael Y. Camargo, D. C. M.-J.-B. S. L. C. S. R. (2012). Accelerating gene regulatory networks inference through gpu/cuda programming. *IEEE*.

[Fabrizio F Borelli, 2012] Fabrizio F Borelli, Raphael Y de Camargo, D. C. M. J. L. C. R. (2012). Gene regulatory networks inference using a multi-gpu exhaustive search algorithm. *Second IEEE International Conference on Computational Advances in Bio and Medical Sciences (ICCABS 2012)*, 14(18)(S5).

[Flynn, 1972] Flynn, M. J. (1972). Some computer organizations and their effectiveness. *IEEE Trans. Comput.*

[Luis E. Ramírez-Chavez, 2011] Luis E. Ramírez-Chavez, Carlos A. Coello Coello, E. R.-T. (2011). A gpu-based implementation of differential evolution for solving the gene regulatory network model inference problem.

[Murray, 2012] Murray, L. (2012). Gpu acceleration of runge-kutta integrators. *IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS*, pages 94, 101.

[Neri, 2008] Neri, F. (2008). On memetic differential evolution frameworks: A study of advantages and limitations in hybridization. *IEEE Congress on Evolutionary Computation (CEC 2008)*.

[Noman and Iba, 2005] Noman, N. and Iba, H. (2005). Inference of gene regulatory networks using s-system and differential evolution. *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation (GECCO'2005),*, pages 439, 446.

[NVIDIA, 2012] NVIDIA (2012). *NVIDIA's Next Generation CUDA Compute Architecture: Kepler*.

[NVIDIA, 2015] NVIDIA (2015). *CURAND library Programming Guide*.

[Schlitt and Brazma, 2007] Schlitt, T. and Brazma, A. (2007). Current approaches to gene regulatory network modelling. *BMC Bioinformatics*, 8(6)(S9).

[Xiong, 2006] Xiong, J. (2006). *Essential Bioinformatics.*

[Žabka, 2011] Žabka, T. O. A. S. V. (2011). The cuda implementation of the method of lines for the curvature dependent flows. *Kybernetika*, pages 251, 272.

Curriculum vitae

Formación académica

- 2009 - 2014 Ingeniería en sistemas Computacionales Hardware.
  Facultad de Ingeniería en la Universidad Autónoma de Chihuahua.
- 2015 - 2016 Maestría en Ingeniería en Computación.
  Facultad de Ingeniería en la Universidad Autónoma de Chihuahua.

Experiencia profesional

- 2016 - Actual MEXROLL
  Sistema para departamentos administrativos y de calidad para el manejo de reportes, manejo de productor, inventario, precios y cotizaciones.
- 2014 - Actual Freelance
  Desarrollo de software
- 2008 - 2013 Atención telefónica
  Asesor telefónico de asistencia y seguimiento de solicitudes a clientes de compañía telefónica.

Correo Electronico: afv9988@gmail.com

Domicilio permanente: Montealbán #8127, Infonavit Nacional
Chihuahua, Chihuahua, C.P. 31120

Esta tesis fue mecanografiada por el autor.