Centro de Investigación y de Estudios
Avanzados del Instituto Politécnico Nacional

Unidad Zacatenco

Departamento de Computación

Auto-adaptación de esquemas algorítmicos de Minería de Datos con aplicación en Bioinformática

Presentada por
A. Alejandra Serrano-Rubio

Para obtener el grado de
Doctora en Ciencias en Computación

Director de tesis
Dr. Guillermo B. Morales-Luna

México, Ciudad de México **Octubre, 2021**

Auto-adaptación de esquemas algorítmicos de Minería de Datos con aplicación en Bioinformática

A. Alejandra Serrano-Rubio

Enviada al Departamento de Computación
el 15 de octubre, 2021, para obtener el grado de
Doctora en Ciencias de la Computación

Resumen

Con la secuenciación genómica completa de la planta *Arabidopsis thaliana* y el desciframiento del genoma humano se comenzó a generar una cantidad considerable de información. El problema surgió cuando se necesitó organizar y analizar los cientos y miles de datos que este descubrimiento brindaba. La solución fue hacerlo de manera automática mediante el uso de algoritmos, los cuales son el resultado del trabajo multidisciplinario entre las ciencias biológicas y computacionales. A pesar de que este tipo de algoritmos han demostrado ser robustos en este tipo de análisis, han tenido que ser optimizados a tráves de mecanismos de Cómputo de Alto Rendimiento con el objetivo de reducir el tiempo de ejecución, y asegurar el uso eficiente de los recursos computacionales de una arquitectura específica. Desafortunadamente, la fuerte dependencia que sugiere este tipo de optimización contribuye a que los algoritmos queden obsoletos debido a que la cantidad de información se incrementa y la arquitectura computacional para la que fueron previstos evoluciona.

Se propone la implementación de un marco de trabajo de auto-adaptación que asegura la portabilidad de los algoritmos y garantiza el uso eficiente de los recursos computacionales, independientemente de la arquitectura y el compilador que se esté utilizando. Para lograrlo, el marco de trabajo selecciona el subconjunto de parámetros y banderas del compilador que minimizan el tiempo de ejecución durante el proceso de compilación. De esta forma, el flujo de instrucciones se ajusta de manera automática a los entornos en que se ejecuta, reduciendo el costo del proceso de optimización.

Aunque la principal aplicación del marco de trabajo de auto-adaptación ha sido el análisis diferencial de expresión genética, no es limitativo a este tipo de dominio. Específicamente, la propuesta ha sido validada a través del análisis de microarreglos relacionados con la evolución de hepatocarcinomas. Para realizar este experimento, se propuso un algoritmo distribuido, que refina la selección de centroides en técnicas de agrupamiento particional; la escalabilidad del algoritmo distribuido se mejoró mediante una biblioteca de operaciones elementales basada en `CUDA` y `OpenMP`. Los resultados muestran que esta propuesta es una herramienta eficiente, capaz de reducir el tiempo de ejecución mediante la selección de parámetros y banderas que mejoran el rendimiento en tiempo de compilación y de esta forma, garantizar la portabilidad de algoritmos de forma casi transparente al usuario.

Palabras clave— Cómputo de Alto Rendimiento, Minería de Datos, Bioinformática, auto-adaptación, rendimiento computacional, análisis de expresión genética.

Auto-adaptación de esquemas algorítmicos de Minería de Datos con aplicación en Bioinformática

A. Alejandra Serrano-Rubio

Submitted to the Computer Science Department
on October 15, 2021, for the degree of
Ph.D. in Computer Science

Abstract

With the complete genomic sequencing of the plant *Arabidopsis thaliana* and the deciphering of the human genome, a considerable amount of information began to be generated. The problem arose when it was necessary to organize and analyze the hundreds and thousands of data that this discovery provided. The solution was to do it automatically through the use of algorithms, which are the result of multidisciplinary work between the biological and computational sciences. Although these types of algorithms have proven to be robust in this type of analysis, they have had to be optimized through High Performance Computing mechanisms in order to reduce execution time, and ensure the efficient use of computational resources of a specific architecture. Unfortunately, the strong dependency suggested by this type of optimization contributes to the obsolescence of the algorithms as the amount of information increases and the computational architecture for which they were intended evolves.

The implementation of a self-adaptation framework is proposed that ensures the portability of the algorithms and guarantees the efficient use of computational resources, regardless of the architecture and compiler being used. To achieve this, the framework selects the subset of compiler flags and parameters that minimize execution time during the compilation process. In this way, the flow of instructions adjusts automatically to the environments in which it is executed, reducing the cost of the optimization process.

Although the main application of the self-adaptation framework has been the differential analysis of gene expression, it is not limited to this type of domain. Specifically, the proposal has been validated through the analysis of microarrays related to the evolution of hepatocarcinomas. To carry out this experiment, a distributed algorithm was proposed that refines the selection of centroids in partitioning *clustering* techniques; the scalability of the distributed algorithm was improved by an elementary operations library based on `CUDA` and `OpenMP`. The results show that this proposal is an efficient tool capable of reducing execution time through the selection of parameters and flags that improve performance at compile time and thus guarantee the portability of algorithms in a way that is almost transparent to the user.

Keywords— High-Performance Computing, Data Mining, Bioinformatics, code auto-adaptation, computational performance, gene expression analysis.

Incluso un error puede resultar ser la única cosa necesaria para un logro.

Henry Ford

Agradecimientos

La aventura que comenzó con una motivación personal ha concluido. Al principio tuve miedo pues el camino parecía ser largo y difícil. Eran muchas cosas que desconocía y otras más que sigo aprendiendo. Sin embargo, en todo momento tuve a las personas indicadas que de una u otra forma me apoyaron para culminar este proyecto. Por lo tanto, quiero aprovechar la ocasión y hacer mención de cada una de ellas.

Al Dr. Guillermo B. Morales-Luna.

Quien es la primera persona a la que quiero agradecer por su paciencia y apoyo, por haber sido parte escencial en mi formación al haberme enseñado lo que significa ser investigador, y finalmente, por haber confiado en mi para formar parte de otros proyectos, los cuales me hicieron crecer en todos los aspectos. Es un placer trabajar y aprender de usted. Gracias por creer en mi. Mi más sincera admiración y respeto.

A mis sinodales.

Agradezco a mis sinodales por sus valiosos comentarios y disposición para formar parte del comite evaluador. De forma personal, al Dr. Luis Gerardo de la Fraga por su paciencia e importante aportación a este trabajo. Al Dr. Saúl Villa-Treviño por compartirme sus datos biológicos para probar mis algoritmos, así como sus palabras que siempre me han alentado a culminar este trabajo. Agradezco a la Dra. Ruth Reyes-Cortés por sus consejos y lectura del libro de Bioinformática que redacte en colaboración con mi asesor. Finalmente, agradezco al Dr. Amilcar Meneses-Viveros por ser parte de mi formación académica desde la maestría, por su disposición para participar activamente en la culminación de este proyecto, así como por su labor profesional y su apoyo brindado en todo momento.

Al Departamento de Computación del CINVESTAV.

Especialmente a los doctores Sonia G. Mendoza-Chapa, José F. Rambó-Rodriguez, José Guadalupe Rodríguez-García y a la Dra. Dolores Lara por transmitirme su conocimiento, su apoyo y los consejos brindados. Además, agradezco a Sofy, Felipa, Erika, Arcadio, Dr. Santiado, José Luis y Cedillo, por siempre procurar todos los recursos necesarios para no distraerme de mis actividades y sobre todo por siempre estar dispuestos a ayudarme con una sonrisa en la cara.

A mi familia.

De manera especial agradezco a mi mami por apoyarme desde que tome esta decisión y alentarme a terminar cuando parecía que todo se ponía dificil. Gracias por, literalmente, ser parte de cada momento. Agradezco a mis hermanos, Bichito, Jhony y Ale por acercarme a la ciencia desde pequeña y ayudarme a ser la persona que soy el día de hoy. Agradezco a mi hermana Kitty por todas sus enseñanzas, por escucharme y por alentarme a cumplir mis objetivos personales y profesionales. Simplemente gracias por creer y confiar en mí, por pasar más de una noche en vela conmigo. Agradezco a mis hermanos por haberme dado la oportunidad de ser tía de cuatro maravillosas sobrinas y un capitán tapón, a los cuales les agradezco por sus sabios consejos de terminar mi tarea para poder entrar al cuarto de juguetes. Finalmente, de manera especial agradezco a Julito por animarme a culminar mis metas en cada etapa de mi vida.

A mis amigos.

A mis amigos Pepe, Vero, Gabi, Ceci, Oli y Eduardo que han estado conmigo de forma incondicional y que siempre me han apoyado para culminar esta etapa de mi vida. Gracias por las pláticas eternas, por sus oraciones, por su confianza y por su ayuda. Agradezco sinceramente por siempre preocuparse por mi y por encontrar el momento para pasar tiempo juntos.

Al CINVESTAV.

Por haber sido mi casa por más de cuatro años y de lo cual siempre me sentiré muy orgullosa.

Al CONACYT.

Finalmente, y no por eso menos importante. Agradezco al Consejo Nacional de Ciencia y Tecnología (CONACYT) por todos los apoyos recibidos y en especial, por haberme provisto de los recursos económicos necesarios para poder realizar mis estudios de maestría y doctorado. Sin ellos hubiese sido imposible.

Índice general

Esta página se dejó en blanco intencionalmente.

Índice de figuras

Esta página se dejó en blanco intencionalmente.

Índice de tablas

Capítulo 1
Introducción

El análisis de expresión genética [23] es una de las disciplinas más importantes de la Bioinformática. Su objetivo es dar respuesta a las cuestiones biológicas más relevantes sobre un organismo que ha sido expuesto a cambios ambientales, circunstancias patológicas o condiciones de estrés. Su análisis requiere algoritmos que faciliten la evaluación de datos «ómicos» [1] de alta dimensionalidad.

La generación de datos «transcriptómicos» [2] ha comenzado a crecer de manera exponencial debido al desarrollo de métodos que permiten deducir y cuantificar el «transcriptoma». Estos métodos están basados en hibridación de microarreglos o en tecnologías de nueva generación (NGS, *Next Generation Sequencing*). El uso de técnicas de Minería de Datos (MD) basadas en tareas de clasificación y agrupación ha permitido el análisis de este tipo de datos [21, 37, 38, 175]. Sin embargo, ambas técnicas no funcionan correctamente en conjuntos de datos de alta dimensionalidad, es decir, con problemas donde el volumen del espacio aumenta haciendo que los datos disponibles se vuelvan dispersos [112, 168, 180]. Los dos enfoques más comunes para analizar la gran cantidad de datos transcriptómicos son: (1) Algoritmos basados en agrupamiento y (2) Algoritmos basados en biagrupamiento. El primer enfoque permite realizar agrupaciones de datos correlacionados que describen relaciones funcionales entre distintos genes frente a un conjunto de condiciones experimentales o viceversa. Por otro lado, biagrupamiento permite la formación de agrupaciones de genes que presenten un comportamiento correlacionado frente a un subconjunto de condiciones experimentales. La principal característica que diferencía a ambos enfoques es la manera en que se realizan las agrupaciones, ya que las técnicas de biagrupamiento permiten solapamiento, así como la existencia de genes o condiciones que no se hayan incluido en ninguna agrupación [42, 139]. Tal característica aporta mayor flexibilidad al momento de realizar un análisis. Dado que en ambos enfoques se analiza completamente el espacio de soluciones se dice que son computacionalmente costosos.

En este capítulo se describe la principal motivación de este trabajo, la definición del problema, y la pregunta de investigación. Se presentan los objetivos y las contribuciones obtenidas. Finalmente, se presenta la estructura de este documento.

[1] Conjunto de disciplinas como la Genómica, Transcriptómica, Proteómica, y Metabolómica.
[2] Secuencias de nucleótidos presentes en una célula, tejido u organismo.

1.1. Motivación

La aplicación de Cómputo de Alto Rendimiento (HPC) en algoritmos de agrupamiento y biagrupamiento juega un papel importante en el desempeño que tienen los algoritmos durante el análisis de expresión genética, ya que su aplicación sugiere una alta dependencia entre el algoritmo y la arquitectura. En este sentido, se debe garantizar que estos algoritmos, independientemente de la arquitectura y del compilador, sigan haciendo uso eficiente de los recursos computacionales conforme mejoran sus características de procesamiento. Por ejemplo, la optimización de: la distribución de las unidades de procesamiento del sistema, la memoria, el tiempo de ejecución de un programa en una arquitectura computacional específica y el conjunto de banderas de compilación que optimizan el flujo de instrucciones. Varios estudios han proporcionado un análisis del rendimiento computacional de algoritmos de agrupamiento y biagrupamiento, optimizados con técnicas de HPC [15, 81, 126, 132, 135, 136, 151, 156, 158, 162, 163, 237], lo que indica que la eficiencia de los algoritmos depende de las características específicas de la arquitectura, el compilador que se utilice, así como también, de las particularidades de los datos a analizar.

Dos posibles soluciones para tratar el problema son la aplicación de técnicas de HPC a un algoritmo en una arquitectura específica, y la autoadaptación de código previamente paralelizado que mejor se adapte a una arquitectura [1,11,12,200,229]. La primera posibilidad se refiere a establecer *a priori* una paralelización a la medida de la arquitectura, y utilizarla en todas las arquitecturas compatibles. Lo anterior, tiene el objetivo de alcanzar el mejor rendimiento computacional posible. Por el contrario, el segundo caso incluye mecanismos adaptativos que modifican los valores de los parámetros en función de la arquitectura donde se esté ejecutando el algoritmo en tiempo de ejecución o compilación. En este trabajo es de interés explorar ambas opciones, utilizando las siguientes pautas: (1) Aplicar una metodología de paralelización hecha a la medida, para obtener una mejor comprensión del comportamiento y solidez de los algoritmos de agrupamiento y biagrupamiento en el análisis de expresión genética, y (2) Explorar mecanismos de adaptación durante el análisis de expresión genética que combinen las ventajas proporcionadas por los métodos y técnicas de HPC.

La principal aportación de este trabajo es proporcionar una estrategia novedosa al campo de la Bioinformática, la cual sea capaz de encontrar un conjunto óptimo de banderas de compilación, con las que un algoritmo paralelizado con mecanismos de HPC debe ser ejecutado en una arquitectura específica, tal que se minimice el tiempo de cómputo necesario para obtener una solución aceptable. Este marco de trabajo es válido para técnicas de MD enfocadas en tareas de clasificación y agrupación, especialmente para algoritmos enfocados en tareas de agrupamiento y biagrupamiento, cuyo objetivo es el análisis diferencial de datos ómicos de alta dimensionalidad. Se espera que esta implementación ayude en el análisis de expresión genética al enriquecer la interpretación de los resultados.

1.2. Definición del problema

En las ciencias biológicas, las arquitecturas computacionales son utilizadas principalmente como una herramienta que facilita la ejecución de cálculos complejos para la resolución de cuestiones biológicas que difícilmente podrían ser resueltas sólo con la intervención humana. Esta comunidad demanda algoritmos que aseguren el uso eficiente de los recursos computacionales y, a su vez, sean capaces de dar una respuesta confiable a los problemas que continuamente se plantean.

Afortunadamente se han desarrollado mecanismos basados en HPC que garantizan el uso eficiente de los recursos computacionales. Estos mecanismos adaptan los algoritmos a las características de procesamiento de una arquitectura, optimizando la memoria, la distribución de los procesadores, el tiempo de ejecución y la selección de banderas que optimizan el flujo de instrucciones en tiempo de compilación. De esta manera, es usual disponer de algoritmos que faciliten la investigación científica, al ser ejecutados en arquitecturas de altas prestaciones.

Sin embargo, cualquier tipo de adaptación de un algoritmo a una arquitectura específica se realiza de forma manual. Es decir, un científico especialista en HPC es responsable de adaptar el algoritmo, y definir el conjunto de banderas de compilación que mejoran el rendimiento computacional. Lo anterior se realiza a través del análisis tanto del algoritmo, como de la arquitectura donde este se ejecutará, y específicamente hablando, de la selección de banderas de compilación, generalmente se realizan ejecuciones sucesivas en ambientes altamente controlados, que en la práctica, no necesariamente resultan ser los óptimos. Esta tarea no es siempre sencilla debido a la fuerte dependencia que existe entre el algoritmo, las metodologías de paralelización y las características de la arquitectura definidas por el tipo de procesador, sistema operativo, memoria y disco duro. En este sentido, en el escenario ideal se desea que los algoritmos sigan siendo eficientes con independencia de la arquitectura y del compilador que se utilice, y que además, si la arquitectura o compilador llegase a cambiar, la optimización se realice de forma que interfiera lo menos posible con el trabajo diario del usuario final, es decir, sin que ello suponga un tiempo y un esfuerzo suplementario.

1.3. Hipótesis

La solución de problemas biológicos modernos requiere del uso de métodos computacionales sofisticados, debido a que el volumen de datos ómicos crece exponencialmente en tamaño y complejidad. Además, el mantenimiento de estos algoritmos es imposible de realizar debido a que, su optimización, depende tanto del compilador, como de las diferentes arquitecturas computacionales, las cuales guiadas por la conocida «Ley de Moore» [32] evolucionan constantemente.

Con el objetivo de liberar al usuario del mantenimiento de algoritmos, mejorando la portabilidad en su rendimiento, y asegurando el uso eficiente de los recursos computacionales, se define la hipótesis:

La implementación de un marco de trabajo basado en técnicas de optimización permite minimizar el tiempo de ejecución de un algoritmo mediante la selección de un conjunto de parámetros durante el proceso de compilación, independientemente de la arquitectura y del compilador que se esté utilizando, con el objetivo de asegurar la portabilidad y el uso eficiente de los recursos computacionales.
Lo anterior debe ser válido para técnicas de MD enfocadas en tareas de clasificación y agrupación, especialmente para algoritmos enfocados en tareas de agrupamiento y biagrupamiento cuyo objetivo es el análisis diferencial de datos ómicos de alta dimensionalidad.

De esta manera se evitará rediseñar la estrategia de paralelización en algoritmos de agrupamiento para obtener versiones eficientes en distintas arquitecturas con diferentes compiladores. Esto permitirá a los investigadores de las ciencias biológicas u otras áreas trabajar sólo en su ámbito de investigación, sin preocuparse por el rendimiento de los algoritmos y herramientas que utilizan constantemente. La idea principal de esta metodología, se basa en la optimización de un modelo matemático que minimiza el tiempo de ejecución de un algoritmo, a partir del control del flujo de instrucciones definido por las banderas de compilación. Finalmente, considere que aunque este marco de trabajo se enfoca en algoritmos de MD, la propuesta se plantea de manera general para cualquier tipo de dominio.

1.4. Objetivos

Se describen los objetivos de esta tesis.

1.4.1. Objetivo general

Diseñar e implementar un marco de trabajo basado en técnicas de optimización que permita minimizar el tiempo de ejecución de un algoritmo mediante la selección de un conjunto de parámetros durante el proceso de compilación, con independencia de la arquitectura y del compilador, con el objetivo de asegurar la portabilidad y el uso eficiente de los recursos computacionales.

Lo anterior debe ser válido para técnicas de Minería de Datos enfocadas en tareas de clasificación y agrupación, especialmente para algoritmos de agrupamiento cuyo objetivo es el análisis diferencial de datos ómicos de alta dimensionalidad.

1.4.2. Objetivos particulares

Para alcanzar el objetivo general se plantean los siguientes objetivos particulares:

- Identificar los aspectos relevantes de los algoritmos de Minería de Datos basados en tareas de clasificación y agrupación. Estos algoritmos deben estar enfocados en el análisis de expresión genética de datos de alta dimensionalidad, para definir un algoritmo genérico que proponga una solución a este tipo de problemas.

- Diseñar mecanismos de HPC que mejoren el tiempo computacional de los algoritmos de Minería de Datos basados en tareas de clasificación y agrupación. Estos mecanismos deben orientarse al análisis de expresión genética de datos de alta dimensionalidad, para garantizar el uso eficiente de los recursos computacionales durante su ejecución.

- Proponer un modelo matemático que refleje el tiempo computacional que requiere un algoritmo para obtener una solución aceptable. En este objetivo se debe considerar la diversidad de características físicas que describen a los componentes que conforman tanto a las arquitecturas homogéneas, como a las heterogéneas.

- Implementar un marco de trabajo basado en técnicas de optimización las cuales minimicen el tiempo computacional que requiere un algoritmo para obtener una solución aceptable. Se espera que el marco de trabajo se retroalimente de los resultados obtenidos de experimentos realizados con antelación.

- Validar el marco de trabajo mediante su aplicación en el análisis diferencial de expresión genética.

1.5. Visión general de la metodología propuesta

Como ya se mencionó, el objetivo de este trabajo es la propuesta de un marco de trabajo de autoadaptación capaz de encontrar un conjunto de parámetros en tiempo de compilación, con los que se minimice el tiempo de ejecución de un algoritmo de clasificación y agrupación, tanto en arquitecturas homogéneas, como heterogéneas. Lo anterior con independencia de la arquitectura y del compilador que se esté utilizando. Estas arquitecturas están constituidas por un número arbitrario de unidades de procesamiento con canales de comunicación imperfectos.

La metodología de autoadaptación propuesta se divide en tres fases principales: diseño, instalación y ejecución. En la etapa de diseño se analizan las características de los algoritmos de MD para su posterior implementación, utilizando mecanismos de HPC basados en `OpenMP`, `CUDA` y `MPI`. Enseguida, se realiza el estudio analítico del tiempo de ejecución, considerando los «parámetros objetivo» que contribuirán en la autoadaptación de un algoritmo a una arquitectura, basados en el tamaño de bloque que deberá ser procesado por cada unidad de procesamiento, y en el conjunto de banderas del compilador que mejoran el flujo de instrucciones durante el proceso de compilación. A partir de esta fase, se propone un modelo matemático que refleja el tiempo teórico computacional que requiere un algoritmo para obtener una solución en una arquitectura específica; en la fase de instalación se obtienen las características de la arquitectura paralela. Esta fase se basa en un modelo puramente experimental que debe ser ejecutado cada vez que la arquitectura evolucione o existan cambios sustanciales en el código o paralelización del algoritmo. Los datos obtenidos son almacenados en una base de datos, la cual retroalimenta al marco de trabajo para la futura ejecución de nuevos experimentos; en la última etapa se obtiene información

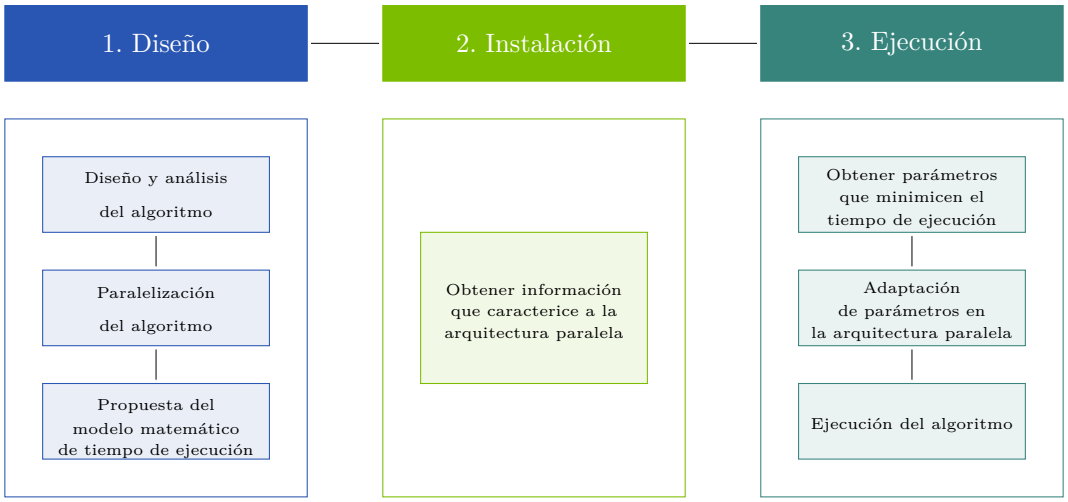


Figura 1-1: Resumen del marco de autoadaptación propuesto. La metodología de autoadaptación propuesta se divide en tres fases principales: diseño, instalación y ejecución. En la etapa de diseño se analizan las características del algoritmo para su posterior implementación. En la fase de instalación se obtienen las características de la arquitectura. La etapa de diseño forma parte del marco de trabajo general y por tanto, debe actualizarse cada vez que se ocurre un cambio en la arquitectura destino; en la última etapa, el valor de los parámetros se adapta a las características de la arquitectura minimizando así, el tiempo de ejecución. La metodología finaliza con la ejecución del algoritmo.

relacionada a la carga de trabajo de la arquitectura paralela y se invoca al modelo matemático para poder ajustar los parámetros objetivo a las características de la arquitectura. La metodología termina con la ejecución del algoritmo que haga uso eficiente de los recursos computacionales (ver figura 1-1).

Se hace énfasis en el modelo experimental constituido por los parámetros que caracterizan a la arquitectura, y que ha sido dividido en dos grupos: (1) submodelo de cómputo que representa el tiempo necesario por una unidad de procesamiento para realizar una operación básica; (2) submodelo de comunicación que agrupa la capacidad de intercomunicación en función de la capacidad física de la red y el tamaño de los mensajes. De esta manera, la primera parte del modelo analítico considera los componentes del modelo experimental que tienen influencia sobre el tiempo de ejecución y los parámetros objetivo de la implementación del algoritmo.

Como ejemplo introductorio, en la figura 1-2 se muestra la optimización del tiempo de ejecución en la autoadaptación del algoritmo de Strassen para multiplicación de matrices [99] al utilizar la metodología propuesta. Los resultados han sido obtenidos mediante la múltiple ejecución del algoritmo para distintos tamaños de problema (2^8, 2^9, 2^{10}, 2^{11}, 2^{12}). Se muestra el tiempo promedio de ejecución del algoritmo secuencial; el tiempo promedio de ejecución obtenido por algoritmo paralelizado «a mano», es decir, hecho a la medida de la arquitectura considerando las características de esta; y el tiempo promedio de ejecución obtenido de manera automática por la metodología propuesta. El tiempo de ejecución se ha normalizado utilizando una escala logarítmica y se ha considerado una arquitectura homogénea.

Como se puede apreciar en este ejemplo, se han conseguido mejoras importantes respecto al caso promedio, así como un acercamiento al tiempo de ejecución óptimo definido por la paralelización realizada a mano. En las siguientes secciones se describe la metodología aplicada a tareas de clasificación y agrupación de MD.

1.6. Contribuciones

Las principales contribuciones de este trabajo doctoral se describen dentro de tres campos de investigación, definidos en orden de importancia, por HPC, MD y la Bioinformática. Las más destacadas se presentan a continuación.

La aportación más importante es la propuesta de un marco de trabajo de autoadaptación que se encarga de encontrar un conjunto de parámetros en tiempo de compilación, definidos por: (1) el tamaño de bloque de datos que debe ser enviado a cada unidad de procesamiento y (2) las banderas del compilador que deben ser utilizadas para controlar el flujo de instrucciones durante el proceso de compilación, con el objetivo de minimizar el tiempo de ejecución de un algoritmo. El marco de trabajo es de propósito general, sin embargo ha demostrado tener un buen desempeño en algoritmos de agrupamiento enfocados en el análisis diferencial de expresión genética. Este marco de trabajo se desarrolló en `python`, lo cual representa una interfaz amigable al usuario que permite la optimización de algoritmos desarrollados en `C`, `C++` y `python`. El marco de trabajo es multiplataforma e integra una base de datos donde almacena los resultados obtenidos de previas optimizaciones. A diferencia de otras propuestas en el estado del arte [7, 39, 124, 155, 165, 220], se toma ventaja de los beneficios que aporta el aprendizaje supervisado, ya que los resultados obtenidos con antelación son utilizados para retroalimentar futuros experimentos y así, obtener resultados más «finos». Se ha implementado un conjunto de algoritmos de optimización que son ejecutados al mismo tiempo con el objetivo de encontrar una solución óptima. Esta implementación se ha realizado debido al hecho de que el dominio del problema puede ser tan grande que la búsqueda exhaustiva no representa una solución viable. De esta forma, el proceso de autoadaptación se realiza en dos momentos: (1) durante la instalación, identificando los componentes básicos de la arquitectura y (2) durante la ejecución, mediante la selección aleatoria de algoritmos de optimización que tratarán de resolver el problema. Cuando el marco de trabajo encuentra la solución, ejecuta el algoritmo utilizando los parámetros óptimos que minimizan el tiempo de cómputo. Los resultados demostraron un rendimiento competitivo con `HPL`, obteniendo una mejora en el rendimiento de 2.3x con respecto a la versión secuencial. Finalmente, el marco de trabajo permite la implementación de nuevas funciones objetivo, por ejemplo la optimización del consumo de energía. La principal ventaja que supone esta propuesta es que en lugar de optimizar un programa directamente a través de mecanismos de HPC, el usuario determina un espacio de búsqueda de posibles parámetros de implementación para encontrar los parámetros objetivo.

Se contribuyó al estado del arte de HPC a través de la propuesta de un modelo matemático que refleja el tiempo de cómputo de un algoritmo iterativo paralelo. Este modelo supone un sistema compuesto por dos subsistemas: (1) subsistema de

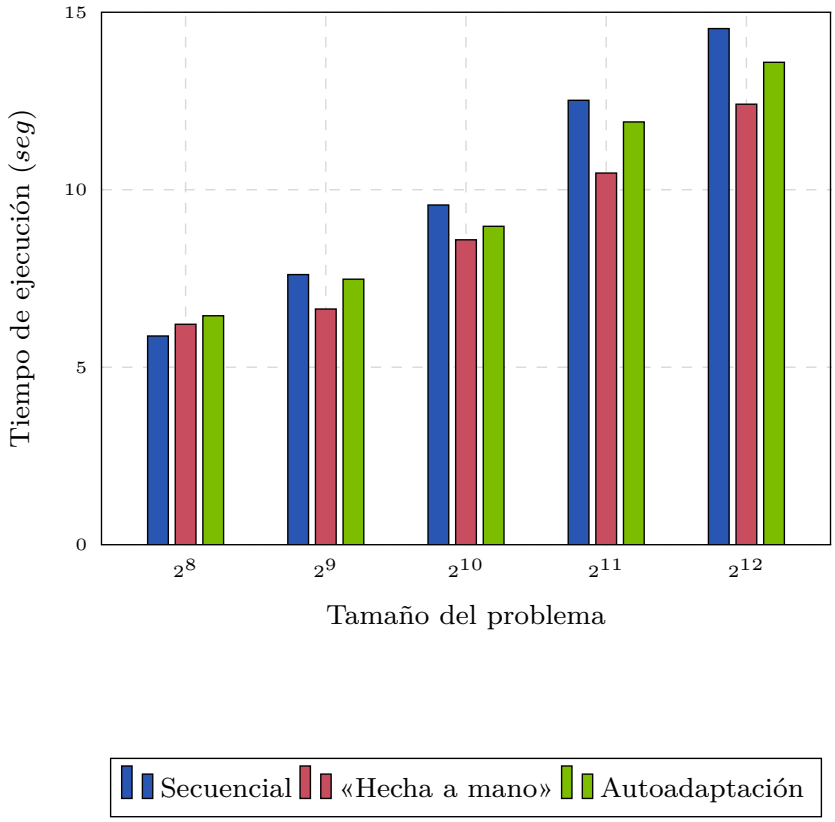


Figura 1-2: Optimización del tiempo de ejecución de la autoadaptación del algoritmo de Strassen para multiplicación de matrices (normalización del tiempo de ejecución utilizando una escala logarítmica). Los resultados han sido obtenidos mediante la múltiple ejecución del algoritmo para distintos tamaños de problema. Cada experimento ha sido ejecutado 32 veces para obtener una significancia estadística. Se muestran los tiempos promedio de ejecución para: el algoritmo secuencial, el algoritmo paralelizado a mano (es decir, hecho a la medida de la arquitectura considerando las características de esta) y el obtenido de manera automática por la metodología propuesta. Los resultados demuestran mejoras importantes respecto al caso promedio, así como un acercamiento al tiempo de ejecución óptimo definida por la paralelización hecha «a mano».

cómputo, el cual considera el costo computacional de cada una de las operaciones aritméticas que utiliza el algoritmo y (2) subsistema de comunicación, el cual supone el tiempo computacional que se requiere para: iniciar la comunicación entre dos unidades de procesamiento, comenzar a enviar un dato (latencia) y enviar un dato básico. El planteamiento del modelo se ha realizado tanto para arquitecturas homogéneas, como para arquitecturas heterogéneas, en las cuales se considera el tiempo de cómputo de la unidad de procesamiento que más tarda en obtener un resultado. En el estado del arte [63, 72, 101, 113, 207, 228, 233] se han propuesto dos formas para auto-adaptar un algoritmo (1) utilizando ejecuciones sucesivas de un algoritmo en cuestión, variando los parámetros y (2) mediante la propuesta de un modelo matemático que refleje el tiempo computacional. Sin embargo, en esta propuesta se emplean ambos enfoques, el primero con los resultados previos almacenados en la base de datos que sirven como datos de entrenamiento para un nuevo experimento y el segundo a través del modelo matemático propuesto. Esto hace que la compilación, sea un proceso transparente al usuario, sin suponer una mayor carga de trabajo al sistema. Finalmente, el usuario debe tener en cuenta, que cada vez que la arquitectura evoluciona, tiene que realizar la actualización de las características de la arquitectura.

La disponibilidad de operaciones de Álgebra Lineal es crucial para la solución eficiente de algunos problemas computacionales. Con el objetivo de maximizar el rendimiento de los algoritmos se contribuyó al estado del arte de HPC y Cómputo Científico con la propuesta de una biblioteca de operaciones elementales optimizadas con mecanismos de HPC basada en `CUDA` y `OpenMP`. Las operaciones elementales están caracterizadas por las operaciones descritas por la biblioteca BLAS y algunos otros algoritmos básicos como el producto de Strassen. La biblioteca es de propósito general, sin embargo ha demostrado tener un buen desempeño en algoritmos de agrupamiento enfocados en el análisis diferencial de expresión genética. La biblioteca fue desarrollada en `C++`, haciendo uso de Programación Orientada a Objetos (POO) y es multiplataforma. Finalmente, los resultados demostraron que la biblioteca propuesta es competitiva con otras propuestas descritas en el estado del arte (`CUSP` y `MAGMA`). A diferencia de otras implementaciones propuestas en el estado del arte [2, 50], la biblioteca está compuesta por una base de datos que almacena en el sistema de archivos soluciones obtenidas de ejecuciones realizadas con anterioridad.

Por otro lado, se contribuyó al área de MD con la propuesta de un algoritmo distribuido basado en «protocolos de consenso» [3] y cuya aplicación es proponer una solución a problemas de clasificación y agrupación, a través del refinamiento de los centroides de manera eficiente. A diferencia de los algoritmos propuestos como «agrupamiento por consenso» (CC), la propuesta permite ejecutar de manera simultánea distintos algoritmos de agrupamiento iterativo en distintos nodos de una arquitectura distribuida con diferentes parámetros de implementación. Cada nodo se encarga de analizar al menos el 70 % de los patrones contenidos en el conjunto total de patrones. El algoritmo fue desarrollado utilizando mecanismos de paso de mensaje definidos por la biblioteca `MPI`. Los resultados demostraron un alto desempeño en cuanto a eficacia

[3] Un protocolo de consenso consiste en un algoritmo para poner de acuerdo a múltiples procesos [230].

y eficiencia. Se observa que el algoritmo trabaja mejor para conjuntos de datos de alta dimensionalidad. Cuando el conjunto de datos a analizar es relativamente pequeño, el algoritmo no mejora los tiempos de cómputo y el refinamiento de centroides es más lento, debido a que se requieren operaciones de comunicación entre cada una de las unidades de procesamiento que conforman la arquitectura destino.

Finalmente, se contribuye en el área de las Ciencias Biológicas y la Bioinformática con el análisis diferencial de expresión genética relacionada al desarrollo temporal de hepatocarcinomas en ratas. Este análisis se realizó *in-silico*, y algunos de los genes identificados han sido descritos en el estado del arte. Sin embargo, se requiere del trabajo multidisciplinario para poder evaluar los resultados finales dentro del contexto biológico.

1.6.1. Publicaciones realizadas

Serrano Rubio, A. A., Meneses Viveros, A., Morales Luna, G. B., & Paredes López, M. (2020). **Towards BIMAX: Binary Inclusion-MAXimal Parallel Implementation for Gene Expression Analysis**. Computación y Sistemas, 24(1).

Serrano-Rubio, A. A., Meneses-Viveros, A., Morales-Luna, G. B., & Paredes-López, M. (2018, March). **Generic Methodology for the Design of Parallel Algorithms Based on Pattern Languages**. In International Conference on Supercomputing in Mexico (pp. 35-48). Springer, Cham.

Serrano-Rubio, A. A., Meneses-Viveros, A., Morales-Luna, G. B., & Paredes-López, M. (2017, March). **Parallel BIMAX: Binary Inclusion-MAXimal for Gene Expression Analysis**. In International Conference on High Performance Computing in Vienna Austria.

Serrano-Rubio, A. A. y Morales Luna, G. B., (Noviembre 2019). **Introducción a la Bioinformática.** Registro de propiedad intelectual.

1.7. Organización del documento

El contenido de este documento se divide en seis capítulos incluyendo al capítulo introductorio. La estructura de los capítulos subsecuentes se describe a continuación.

El capítulo 2, describe la principal motivación del HPC aplicado al área de la Bioinformática. Se presentan los conceptos biológicos necesarios para definir el análisis de expresión genética. Finalmente, se introduce el estudio de los perfiles de expresión genética y el concepto de microarreglos, describiendo las metodologías experimentales y de análisis empleadas para este tipo de análisis.

El capítulo 3 describe las propiedades algorítmicas de las técnicas de agrupamiento utilizadas en el análisis de expresión genética. Se describe la forma general de agrupación que tienen estos algoritmos. De manera específica, en este capítulo se hace énfasis en los Algoritmos K-medias, Medias-C difusas y Maximización de la Similitud Gaussiana. También, se describe el enfoque principal del algoritmo distribuido de agrupamiento particional utilizando mecanismos de paso de mensajes (`MPI`). Este

algoritmo ha sido también optimizado a través de una biblioteca paralela de Álgebra Lineal cuyo objetivo es mejorar el rendimiento computacional.

El capítulo 4 describe los componentes de la metodología propuesta y se formaliza el modelo matemático que optimiza el tiempo de ejecución de un algoritmo en una arquitectura específica. En este capítulo se hace énfasis en la unión de dos subsistemas para que integre las características tanto de una arquitecturas homogénea, como de una arquitectura heterogénea.

El capítulo 5 estudia la aplicación de técnicas de autoadaptación en el diseño de algoritmos paralelos que serán ejecutados en arquitecturas homogéneas y heterogéneas a través de un marco de trabajo. Para validar la metodología, en el capítulo 5 se describe el análisis de un conjunto de datos transcriptómicos relacionados a la evolución de hepatocarcinomas [4] en ratones.

Finalmente, el capítulo 6 resume las conclusiones importantes de este trabajo. Además, se proponen líneas futuras de investigación.

[4]El hepatocarcinoma o carcinoma hepatocelular es el tumor primario del hígado más frecuente que constituye el 80-90 % de los tumores hepáticos malignos [98].

Esta página se dejó en blanco intencionalmente.

Capítulo 2

Cómputo de Alto Rendimiento en Bioinformática

La célula es la unidad anatómica y funcional que proviene de la división celular y contiene el material genético que será transmitido de células madres a hijas. Fue descrita por primera vez en 1665 [226] dando paso a la formulación de la teoría celular, la cual establece que todos los seres vivos están constituidos por células [191,209,218].

A diferencia de la célula procariota [1], la célula eucariota [2] es capaz de almacenar información de su composición y estructura, así como de los productos metabólicos y de su capacidad de dar respuesta al medio. Además, es capaz de generar un estímulo durante la división celular y la apoptosis [3]. En otras palabras, la célula es capaz de almacenar toda la información genética que está codificada en la molécula de ADN [4] en forma de elementos genómicos denominados genes [5]. Por lo tanto, el ADN es la molécula responsable del almacenamiento de la información genética, la cual está constituida por una secuencia específica de otras moléculas más pequeñas denominadas nucleótidos. En el caso de regiones codificantes, el orden particular de los nucleótidos es de crucial importancia debido a que definen la secuencia específica de aminoácidos [6] que determinarán la estructura de una futura proteína [91].

Durante este capítulo se hace énfasis en las etapas del «Dogma Central de la Biología Molecular» y especialmente en la etapa de transcripción genética. También se aborda la fuerte relación que existe entre la transcripción y los métodos computacionales determinados por la Bioinformática. Se describe el estudio del ADN como un registro de información celular. Finalmente, se introducen las técnicas de secuenciación basadas en tecnologías de nueva generación y el estudio de los perfiles de expresión, en los cuales se incluyen las técnicas de hibridación de microarreglos.

[1] Una célula procariota es aquella cuyo material genético no se encuentra protegido por una membrana celular y a su vez carece de un núcleo.

[2] Una célula eucariota es aquella en la que el material genético se encuentra en el núcleo de la célula y contiene un citoplasma que resguarda a los organelos celulares.

[3] La apoptosis se define como el mecanismo de muerte celular programada

[4] Ácido desoxirribonucleico

[5] Un gen representa la información genética que se transmite de generación en generación.

[6] Un aminoácido es una unidad básica que constituye una pequeña fracción de una proteína.

2.1. ADN como registro de información

Los primeros estudios de la molécula de ADN datan de 1869 [52], cuando se planteó la hipótesis de que el núcleo de la célula estaba ligado a su función. El análisis de los componentes del ADN (nucleótidos, azúcar ribosa y un grupo fosfato) fue realizado en 1931 [129] y señaló que los nucleótidos se encontraban unidos por el grupo fosfato en un orden determinado. Finalmente, en 1953 se anunció que la molécula de ADN es la responsable del traspaso de información genética de una célula a otra.

En los mecanismos que codifican los genes contenidos en el ADN a una proteína participan cuatro tipos de nucleótidos que, combinados en grupos de tres (codones), imponen el código específico que define el significado de esta información [91]. Cada nucleótido se compone por tres elementos estructurales: una base nitrogenada, un azúcar (la desoxirribosa) y un grupo fosfato. La base nitrogenada es la responsable de la especificidad de la información, la cual se encuentra clasificada en cuatro diferentes tipos: Adenina, Guanina, Citosina y Timina. Para abreviar los nombres, comúnmente se utilizan las letras `A`, `G`, `C` y `T` respectivamente. Por otro lado, el azúcar y el grupo fosfato desempeñan una función estructural que facilita la polimeración a través de la conexión de los diferentes nucleótidos.

Estructuralmente, el ADN es una molécula de doble cadena, donde cada una está dirigida en sentido antiparalelo y ambas cadenas forman una estructura en espiral. El apareamiento específico de los nucleótidos entre ambas cadenas se realiza por medio de puentes de hidrógeno con una extraordinaria selectividad, siguiendo la siguiente regla: `A` con `T` y `C` con `G`, seguido de la repetición irregular de los mismos a lo largo de la secuencia. Cada cadena puede alcanzar extensiones de millones de pares de bases y cada una de ellas funcionan como símbolos lingüísticos que aparecen formando una secuencia aperiódica, con patrones estadísticos similares. Cabe mencionar que cada diez pares de bases dan lugar a una vuelta completa de la hélice [182].

La combinación de nucleótidos forma unidades funcionales denominadas genes. Cada gen representa un mensaje codificado con las instrucciones necesarias para formar un producto funcional que dirige la morfogénesis de un organismo. En la mayoría de los casos, el producto funcional es una proteína. Por lo tanto, es importante preservar el orden de las bases nitrogenadas en conformidad con lo establecido en el código genético. Perturbar levemente el orden de las bases nitrogenadas por medio del reemplazo de un nucleótido por otro, produce una mutación que en la mayoría de los casos es neutra. Sin embargo, en una proporción baja, estas mutaciones derivan en cambios ventajosos que son aprovechados por los mecanismos de selección natural y, en los casos de duplicación genética, el conjunto de estas mutaciones en el genoma de un organismo se denomina como innovación evolutiva [119, 160].

El código genético de un organismo se define como el conjunto de reglas que codifican una secuencia de nucleótidos a una secuencia de aminoácidos que en conjunto forman una proteína. Este código es común en todos los organismos. En 1954, Gamow comenzó a descifrar el código genético [74] y, posteriormente, Khorana [116] en 1965 afirmó que la base fundamental de la vida radicaba en la información contenida en la molécula de ADN. Entender este código fue de gran relevancia en el siglo XX debido

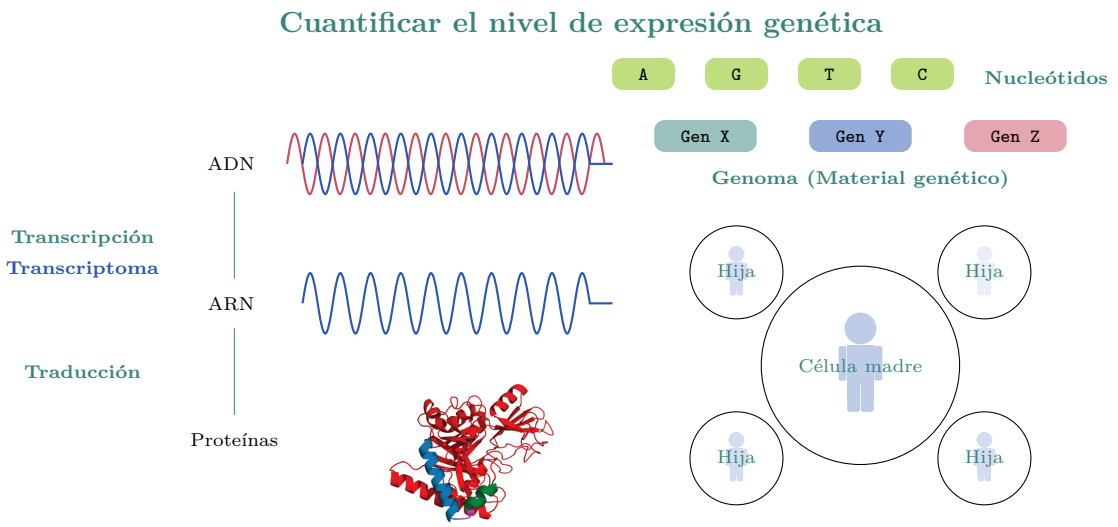


Figura 2-1: El «Dogma central de la Biología Molecular» es un flujo de información unidireccional e irreversible de la información contenida en la molécula de ADN ubicada en el núcleo de la célula. Este flujo se compone de dos etapas consecutivas denominadas «transcripción» y «traducción». Durante la fase de transcripción, la secuencia de ADN de un gen se «vuelve a escribir» en forma de ARN. Enseguida el ARN se procesa con el objetivo de formar un producto final denominado ARN mensajero (ARNm). En la segunda etapa, el ARNm es «decodificado» con el objetivo de construir una proteína que contiene una serie de aminoácidos específicos para realizar una función en la célula.

a que se demostró que existe un lenguaje simbólico que está presente tanto en los organismos más simples como en las formas de vida más complejas e inteligentes, es decir tanto en las bacterias como en los mamíferos. La alta redundancia del código ha sido interpretada como el resultado de un proceso evolutivo tendiente a la teoría de Charles Darwin [164].

Los elementos que conforman al código genético fueron descritos por Crick [46,47]. Gracias a este trabajo se afirmó que la información contenida en el ADN ubicado en el núcleo de una célula es la fuente de información que será decodificada en dos etapas consecutivas denominadas «transcripción» y «traducción». La transcripción involucra la síntesis de ARN [7], molécula constituida por una secuencia de ribonucleótidos. Esta molécula contiene las mismas bases que los nucleótidos que forman parte de la molécula de ADN, con la salvedad que la Timina (T) es sustituida por Uracilo (U). La secuencia de los nucleótidos en una molécula de ARN está definida por el orden que tienen los nucleótidos en la cadena homóloga de ADN. Por último, la traducción implica el cambio del código basado en una secuencia de nucleótidos por otro basado en una secuencia de aminoácidos que darán lugar a una proteína.

Estas etapas forman un flujo de información que es unidireccional e irreversible denominado Dogma Central de la Biología Molecular (ver figura 2-1). Esta formulación tuvo profundas repercusiones puesto que reorientó toda la investigación hacia el análisis de las reglas impuestas por el código genético y su relación con el genoma.

[7] Ácido ribonucléico

El genoma hace referencia a los genes de un organismo, sin considerar la información genética que está codificada por dichos genes. Así, dos organismos comparten un mismo gen que codifica a una misma proteína, aunque la secuencia de nucleótidos que lo conforman sea distinta. La información genética que conforma los factores hereditarios de un organismo, sus genes y por extensión su genoma se denomina genotipo. El genotipo, en unión con los factores ambientales que actúan sobre la información contenida en el ADN, determina las características del organismo, es decir, su fenotipo [53]. La relación genotipo-fenotipo representa la expresión de un gen, y es consecuencia del proceso descrito por el Dogma Central de la Biología Molecular. La interacción del genotipo con distintos factores ambientales hará que se exprese un gen de manera diferente y en tanto se obtendrá un fenotipo diferente.

Se considera que la relación genotipo-fenotipo describe la asociación de los genes específicos asociados a patologías. De aquí, el interés por entender la manera en que se expresan los genes cuando interactúan con distintos factores ambientales.

2.2. Transcripción genética

La transcripción de un gen ocurre cuando una región determinada por una de las cadenas que constituyen a una molécula de ADN, es utilizada como molde por un conjunto de enzimas especializadas denominadas ARN polimerasas, las cuales se encargan de producir una copia complementaria de los nucleótidos que conforman la cadena de ADN. En la fase de transcripción se generan diferentes tipos de ARN, como el ARN mensajero (ARNm) que es traducido a proteínas, el ARN ribosomal (ARNr) que genera los componentes del **ribosoma**, el ARN de transferencia (ARNt) que lleva consigo los aminoácidos al ribosoma durante la traducción del ARNm a proteína, y los ARN no codificantes (ARNc) que no son traducidos a proteína. Cabe mencionar que el ARNc es esencial en los procesos de transcripción y traducción [149].

Un organismo multicelular incluye a todos los seres vivos compuestos por una gran cantidad de células eucariotas. En este caso, y debido a que todas las células del organismo han surgido a partir de una única célula, contienen la misma información genética almacenada en el ADN. Desafortunadamente, las células no son capaces de sobrevivir de manera aislada pues pierden algunas de sus capacidades al especializarse en una función concreta. Por lo tanto, la expresión de cada gen contenido en la molécula de ADN, varía con base al tipo de tejido, estado de desarrollo o del tipo de ambiente en el que se encuentra [28]. El conjunto total de ARN producido por un organismo en una condición determinada, se denomina transcriptoma [34, 141].

2.3. El transcriptoma en Bioinformática

La identificación de perfiles de expresión genética sólo se había logrado gen por gen, mientras que la detección de elementos reguladores del ADN se logró mediante experimentos *in silico* [8] mostrando resultados limitados [91]. Sin embargo, el uso de

[8] Simulación realizada por computadora

nuevas técnicas encargadas del estudio del genoma a gran escala, inicialmente con la técnica de hibridación de microarreglos acompañadas de las tecnologías de nueva generación, ha podido generar información a nivel de todos los genes o transcritos [9] en un momento dado. Esto permite, en principio, definir sistemáticamente los cambios que ocurren a nivel del transcriptoma como respuesta a diferentes estímulos ambientales, a la exposición a patologías y su diagnóstico, en el establecimiento de procesos celulares, o en respuesta a condiciones de estrés en organismos [186, 189].

Las nuevas tecnologías de secuenciación producen una gran cantidad de información que genera la necesidad de desarrollar un apropiado diseño experimental que haga uso de los recursos computacionales para el almacenamiento y análisis de datos. Lo anterior contribuye a una mejor interpretación de la información [190]. Por lo tanto, la Bioinformática ayuda al manejo, análisis y manipulación de datos a gran escala. Se identifican cuatro componentes principales relacionados a estudios ómicos [183]: (1) manejo, almacenamiento y organización de los datos. (2) desarrollo de algoritmos y estadísticas necesarias para establecer las posibles relaciones entre los componentes de una base de datos. (3) desarrollo e implementación de nuevas herramientas computacionales de análisis, visualización e interpretación de datos. (4) desarrollo de metodologías y estrategias que faciliten el descubrimiento biológico como consecuencia de la transformación de información a conocimiento.

2.4. Secuenciación de ARN

Las tecnologías de secuenciación producen decenas de millones de lecturas de secuencias pequeñas denominadas como *reads* [10]. Cada *read* se somete a un control de calidad con el objetivo de remover secuencias que presenten un nivel de calidad bajo (paso 1 en la figura 2-2). Esta limpieza de datos es importante pues permite tener mayor confianza en los resultados obtenidos del análisis posterior de datos [112].

Cada *read* representa una pieza pequeña del ARN. Los *reads* con un valor de calidad razonable se ensamblan en fragmentos más grandes llamados *contigs* [11], Normalmente se permite un número específico de variaciones en un nucleótido al momento de empalmarlos uno con otro (*mismatch*), suponiendo la posible existencia de errores de secuenciación o de variación biológica. En esta fase se alinean las secuencias de los genes o transcritos ensamblados previamente a la referencia, con la finalidad de colocar en el orden correcto los *contigs* generados de acuerdo a su posición en el genoma. Entre los programas que han sido desarrollados para el mapeo de secuencias a un genoma o transcriptoma de referencia se encuentran: Eland [90], SOAP [131], MAQ [96], RMAP [93], SSAHA2 [225], Stampy [138], TopHat [211], RNA-MATE [144], Bowtie [125] y baySeq [92]. Estas herramientas utilizan algoritmos computacionales enfocados en la alineación de lecturas de secuencias [195] (paso

[9] Un transcrito se define como la copia de ADN en una cadena de ARN, la cual contiene la información necesaria para generar una proteína o una subunidad de una proteína [213].

[10] Un *read* es una lectura de secuencia de longitud variable y una estimación de calidad asignado a cada nucleótido

[11] Un *contig* es la secuencia resultante de traslapar varias lecturas de *reads*.

2 en la figura 2-2).

Una vez que se ha representado el transcriptoma, el conteo total de lecturas por gen deben ser normalizadas (paso 3 en la figura 2-2), con el fin de cuantificar la abundancia de cada gen expresado en la muestra (nivel de expresión) [103]. Dentro del análisis, este paso tiene un mayor nivel de dificultad debido a que las familias genéticas [12] se integran generalmente por genes cuyo grado de similitud es de al menos el 95 % y a su vez, cada gen puede derivar en múltiples isoformas. Desafortunadamente, existe un número reducido de herramientas bioinformáticas enfocadas en la normalización y cuantificación. Entre los paquetes bioinformáticos existentes se encuentra ALEXA-seq, MMSEQ y Cufflinks [182, 212], los cuales estiman la abundancia de isoformas basados en diferentes algoritmos [86].

Es claro que los avances tecnológicos permiten estudiar el comportamiento dinámico de los transcritos como respuesta a estímulos. Para un experimento de expresión genética diferencial se seleccionan secuencias transcriptómicas que hayan sido expuestas a distintas condiciones experimentales, con el objetivo de revelar la interrelación entre genes o transcritos que actúan coordinadamente en determinados procesos y mecanismos biológicos. Además, permiten identificar genes involucrados en una determinada enfermedad al contrastar su comportamiento en tejidos sanos.

Actualmente, es posible caracterizar patologías a partir de sus orígenes moleculares y cambios en la expresión de genes específicos, apoyando así, la búsqueda de tratamientos más efectivos y nuevas estrategias de prevención. Encontrar el vínculo existente entre la variación de la secuencia y sus niveles de expresión y la función de los genes es un reto importante al analizar datos ómicos. Así, el último paso en el proceso bioinformático descrito es la anotación funcional de los genes o transcritos identificados, esto es, la búsqueda en bases de datos de una posible función conocida asignada ya a las secuencias de interés. En la actualidad hay disponibles herramientas de anotación poderosas. Aunque existen varios paquetes bioinformáticos desarrollados para tal fin, el más utilizado en la actualidad es el llamado *Basic Local Alignment Search Tool* (Blast) [112], el cual es un conjunto de herramientas desarrolladas por el Centro Nacional de Información Biotecnológica (CNIB) [112].

2.5. Expresión genética

Las funciones celulares se llevan a cabo a través de proteínas o transcritos pequeños no-codificantes, a lo cual se encuentra directamente relacionado el nivel de expresión genética. Cuando un organismo requiere determinadas proteínas (una sobreproducción temporal de adrenalina, por ejemplo), o regular cierta vía metabólica, se alteran los niveles de expresión de los genes que intervienen en su síntesis, pudiendo incrementar (sobre-expresarse) o reducir (inhibirse) su valor. Los cambios en el niveles de expresión genética tiene efectos considerables dentro de las funciones biológicas en

[12] Una familia genética se define como un grupo de genes que se cree que han sido originados a partir de un ancestro común. Por lo general, estos surgen a través de eventos en los que un gen o grupo de genes evolucionan [36].

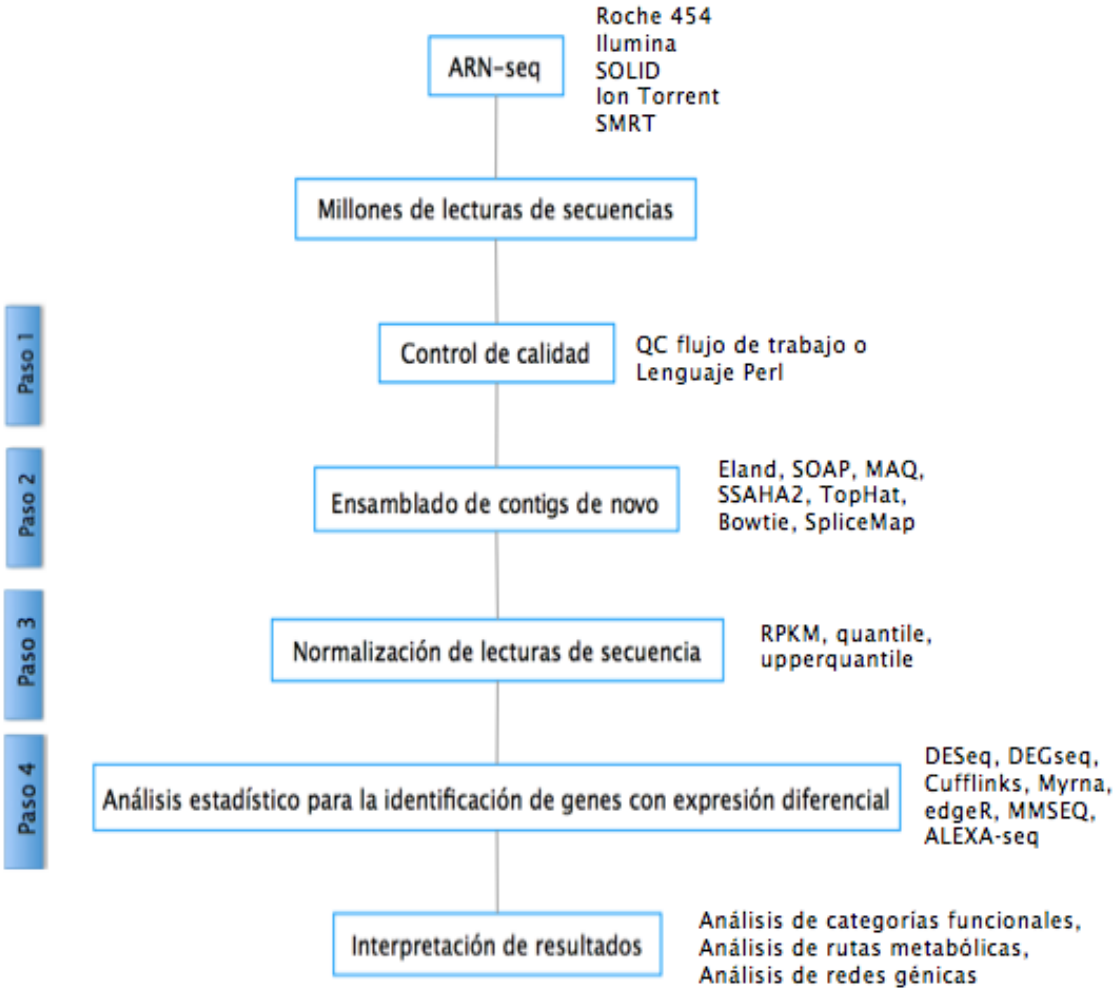


Figura 2-2: Esquema del proceso de análisis de expresión genética utilizando ARN-seq: las lecturas, o reads, de secuencia obtenidas a partir de la secuenciación de muestras de ARN (transcritos), mediante el uso de algunas de las plataformas de secuenciación a gran escala, son sujetas a control de calidad para remover secuencias con pobre calidad (paso 1). Posteriormente, las secuencias son ensambladas, alineadas y mapeadas a un genoma o transcriptoma de referencia utilizando paquetes bioinformáticos (paso 2). El conteo total de lecturas por gen es normalizado utilizando diferentes métodos (paso 3). Se realiza un análisis estadístico para la identificación de genes expresados diferencialmente en un *contig de novo*, el cual hace referencia a secuencias consenso obtenidas a partir del ensamblaje de *reads*. Un *read* representa una secuencia previamente conocida [184] (paso 4). Finalmente, se interpretan los resultados del análisis de expresión diferencial para tratar de inferir la relevancia biológica de las correlaciones encontradas.

las que participan debido a que pueden ser los causantes de alteraciones fisiológicas y procesos patológicos. Los cambios en los niveles de expresión pueden deberse a diversos factores, entre los que se encuentran la diferente localización celular, diferente estado del organismo (sano o enfermo), o en su nivel de desarrollo.

El análisis de la expresión genética plantea un enfoque de búsqueda en las interacciones reales que tienen lugar durante la transcripción de la molécula de ADN en una de ARN. El éxito en los resultados está relacionado con la calidad y cantidad de los datos disponibles. Afortunadamente, cada vez existen más repositorios de alta calidad. Por lo tanto, es de mencionar que la forma en que los datos son normalizados y cuantificados afecta sustancialmente los resultados en este tipo de análisis [26].

Por lo tanto, la selección de los métodos estadísticos empleados actualmente para el análisis de expresión diferencial genera un gran impacto en la precisión de las conclusiones obtenidas respecto al modelo o sistema biológico (paso 4 en la figura 2-2). Esto es una fuerte motivación para proponer y desarrollar nuevas metodologías que además de ser confiables y robustas aseguren el uso eficiente de los recursos.

2.6. Análisis de perfiles de expresión genética

Los avances recientes en las ciencias biológicas han desencadenado una revolución de descubrimientos que han ayudado a entender los sistemas biológicos más complejos. Un paso fundamental en este proceso de investigación es el medir y cuantificar la actividad genética que describe los diferentes niveles de expresión [235].

Como ya se mencionó en la sección 2.1 en la página 15, la transcripción se caracteriza por la información codificada en ARN a partir de la expresión de una región determinada del genoma. Sin embargo, no todos los genes están presentes al mismo tiempo, es decir, los genes son activados por diversos factores que regulan este mecanismo, por ejemplo: la función de la célula en un tejido particular, el estado de desarrollo del organismo, la respuesta ante estímulos o factores externos, y los estados patológicos [115].

El estudio del perfil de expresión genética (ver figura 2-3) es una de las prácticas más comunes dentro de la Biología, ya que permite el estudio de numerosos problemas biológicos que se plantean en organismos de todo tipo. El objetivo de este tipo de análisis es la resolución de problemas biológicos que, por lo general, son costosos. En este sentido, se busca el poder analizar un gran volumen de datos ómicos para extraer la mayor cantidad de información mediante técnicas computacionales.

La realización de análisis de expresión genética mediante métodos computacionales de forma automática involucra el manejo de datos almacenados en archivos de texto plano o bases de datos. Existen técnicas que cuantifican el nivel de expresión genética y que permiten almacenar datos de forma automática [173]. Entre estas técnicas se encuentran las técnicas basadas en secuencias de marcadores de expresión (ESTs) [22], la técnica de análisis en serie de expresión genética (SAGE, por sus siglas en inglés *Serial analysis of gene expression*), basada en el principio de que un segmento corto de ADN permite identificar un único gen transcrito [217], o la tecnología de microarreglos (*microarrays*) [189], la cual es la más utilizada actualmente en

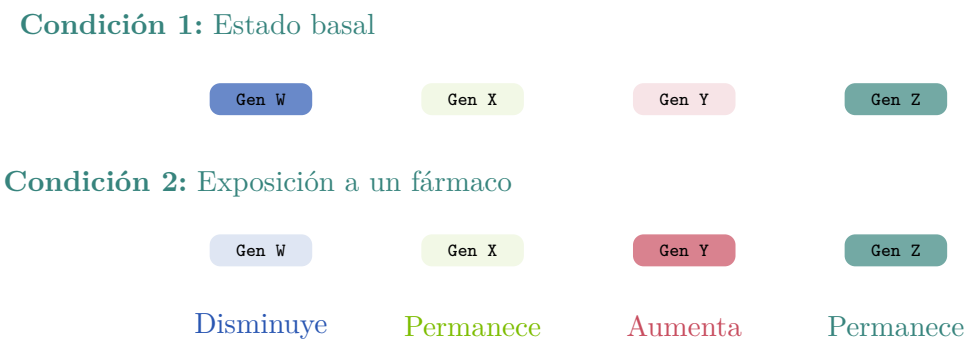


Figura 2-3: Perfil de expresión genética de cuatro genes (gen `W`, gen `X`, gen `Y`, gen `Z`) de un genoma específico frente a dos condiciones experimentales: Estado basal del algoritmo y Exposición a un fármaco. El análisis puede determinar que conjunto de genes están involucrados en un proceso patológico. Se realiza una comparación directa con la condición definida por el estado basal del organismo. Se observa que para el gen `W`, el nivel de expresión disminuye, por tanto se dice que se inhibe. Para el gen `X` el nivel de expresión se mantiene constante en un nivel bajo. Por otro lado, para el gen `Y` el nivel de expresión aumenta, y por tanto se dice que se sobre-expresa y finalmente, para el gen `Z` el nivel permanece constante, sobre-expresándose en ambas condiciones experimentales.

experimentos de expresión genética [56].

Los microarreglos han adquirido gran popularidad gracias a su capacidad de construir valiosas bases de datos de los distintos niveles de expresión de un determinado número de genes expuestos a un conjunto de condiciones experimentales [235].

La técnica de microarreglos consiste en colocar los genes cuyos niveles de expresión requieren ser cuantificados sobre un soporte sólido (de vidrio o membrana de nylon). Posteriormente, tras varios mecanismos bioquímicos (hibridación, lavado, relavado) y ópticos, se obtiene una imagen en la que la intensidad lumínica detectada para cada una de las celdas del microarreglo es proporcional al nivel de expresión de cada uno de los genes depositados inicialmente (ver figura 2-4). Esto permite cuantificar el perfil de expresión de la muestra, sometida a distintas condiciones experimentales [128]. Se realiza un procesamiento digital de imágenes. Posteriormente, se crea una matriz a partir de los patrones de intensidad de las celdas del microarreglo. Generalmente, se aplica una discriminación de la señal del ruido que pudiera existir en segundo plano. Al finalizar, se obtiene una matriz bidimensional de datos numéricos, donde una de las dimensiones representa los distintos genes, y la otra las condiciones bajo las que se ha realizado el experimento. Cada uno de los valores de la matriz representará, por tanto, el nivel de expresión de un determinado gen bajo una cierta condición experimental.

Los microarreglos se han convertido en la técnica más utilizada para el análisis de datos ómicos [174], aumentando continuamente el número de bases de datos de experimentos disponibles en Internet. Se presenta una lista corta con las más relevantes:

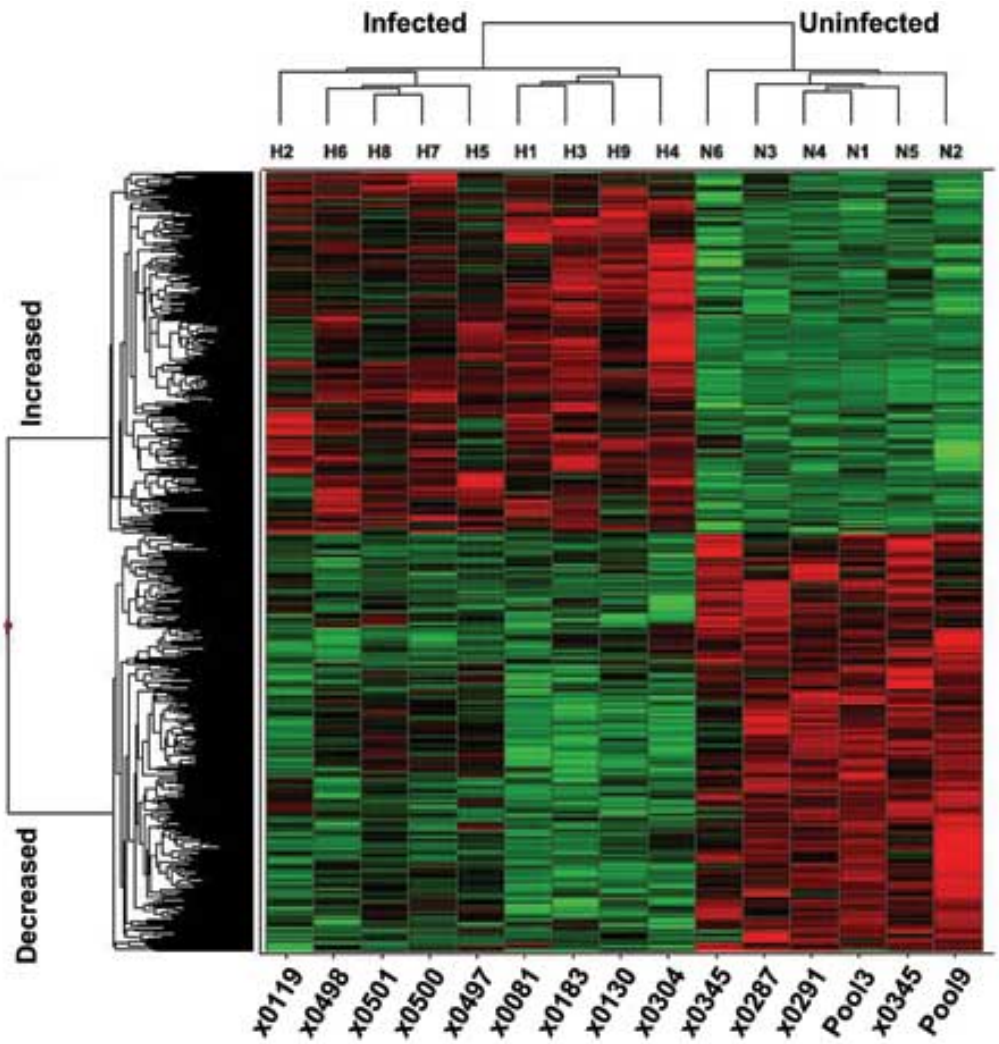


Figura 2-4: Microarreglo procesado con técnicas de agrupamiento jerárquico de genes con nivel de expresión alterado durante infección con HCV (virus de hepatitis C). Se ilustra la agrupación jerárquica en 2 dimensiones de 971 genes identificados como diferencialmente regulados en chimpancés infectados contra chimpancés no infectados. El aumento y disminución en la expresión de los genes específicos se ilustra por el color rojo y verde respectivamente, mientras que el negro indica que no hay cambio respecto al nivel basal. Cada dendograma representa un conjunto de genes que se comportan de manera similar frente a un subconjunto de condiciones, por lo que se pueden observar cuatro agrupaciones diferentes: a) grupo de genes que incrementan su expresión genética en chimpancés infectados (grupo superior izquierdo), b) grupo de genes que incrementan su expresión genética en chimpancés no infectados (grupo superior derecho), c) grupo de genes que disminuyen su expresión genética en chimpancés infectados (grupo inferior izquierdo), d) grupo de genes que disminuyen su expresión genética en chimpancés no infectados (grupo inferior derecho) [16].

- Array express database [167] [13]
- Broad institute, cancer genomics groups [30] [14]
- Gene expression omnibus database [66] [15]
- Jackson laboratory, statistical genomics group [40] [16]
- Microarray gene expression data (MGED) society [25] [17]
- National human genome research institute [94] [18]

Su principal característica reside en la gran cantidad de información que estos pueden generar, ya que es posible representar genomas completos. Sin embargo, existen también algunos inconvenientes, tal como el elevado costo de la producción de microarreglos, lo que puede implicar reducir el número de experimentos [235]. Por otro lado, es necesario establecer ciertas medidas que controlen la calidad del microarreglo, ya que cada una de las fases de producción constituye una posible fuente de ruido. De este modo, resulta imprescindible que los estudios comparativos se hagan sobre las mismas producciones, para no inducir a más errores. Por este motivo, los distintos procesos de exploración y creación de matrices de expresión genética pueden ocasionar valores nulos, con ruido o variaciones producidas como consecuencia de la etapa experimental [175]. Todo ello hace necesario la utilización de una etapa de preprocesado de datos, previa a la aplicación de cualquier técnica de análisis [178,214]. De esta manera, comúnmente se utilizan técnicas de aprendizaje dentro del preprocesado de datos, con el fin de asegurar una mejor eficiencia en la fase de análisis, ya que el objetivo de dichas técnicas se basa en preparar los datos en términos de normalización, o incluso eliminar información que no aporte relevancia significativa [186].

Los microarreglos se enfocan en obtener información que permita inferir conclusiones relevantes, a través de la interpretación de resultados de experimentos biológicos. En los últimos años, gran parte de la investigación bioinformática se enfoca en diseñar nuevas metodologías de análisis, especialmente en técnicas desarrolladas en el campo de la Estadística Descriptiva o de MD. Dentro de este último ámbito son de especial interés para el desarrollo de este trabajo las técnicas de agrupamiento y biagrupamiento, que se detallan en el capítulo 3.

[13] http://www.ebi.ac.uk/arrayexpress/
[14] http://www.broadinstitute.org/cancer/cga/
[15] http://www.ncbi.nlm.nih.gov/geo/
[16] http://research.jax.org/faculty/gary-churchill.html
[17] https://marinemetadata.org/references/mged
[18] https://www.genome.gov

Esta página se dejó en blanco intencionalmente.

Capítulo 3
Técnicas paralelas de algoritmos de Minería de Datos

Las Ciencias Computacionales en el contexto biológico se apoyan de técnicas de MD para el estudio de moléculas relevantes para la vida [172]. Se requiere de la colaboración de expertos en ambas áreas del conocimiento científico para trabajar multidisciplinariamente en el diseño e implementación de nuevos modelos, los cuales se ajusten a las necesidades del problema de estudio con la finalidad de obtener un resultado con significado biológica [175,185]. Cabe mencionar que este resultado debe describir el funcionamiento de los sistemas biológicos, y las relaciones que existen entre ellos.

La motivación de este capítulo es aplicar una metodología de paralelización hecha a la medida para obtener una mejor comprensión del tiempo computacional, comportamiento y solidez de los algoritmos de agrupamiento particional en el análisis de expresión genética, cumpliendo así con el primer enfoque descrito en la sección 1.1.

En este capítulo se proveé al científico bioinformático de un marco de referencia, el cual permite mejorar el rendimiento computacional de los algoritmos de agrupamiento a través de mecanismos de HPC. Para lograrlo, se identifican las propiedades algorítmicas de los métodos de agrupamiento particional más utilizados en el análisis de microarreglos. Se expone la optimización de su rendimiento mediante la propuesta de un algoritmo genérico distribuido basado en protocolos de consenso, el cual contribuye de manera positiva en el refinamiento de los centroides [1]. Este algoritmo ha sido implementado a través de mecanismos de paso de mensajes haciendo uso de la biblioteca `MPI` [87]. Para la optimización del rendimiento de las operaciones elementales, se describe la arquitectura de una biblioteca paralela de Álgebra Lineal enfocada en `CUDA` [68] y `OpenMP` [33].

Cabe mencionar que los resultados obtenidos en este capítulo, demuestran que la aplicación de un algoritmo distribuido basado en protocolos de consenso con aplicación en problemas de agrupamiento particional permiten el refinamiento de los centroides de manera eficiente. Además, esta propuesta permite al usuario ejecutar simultáneamente distintos algoritmos de agrupamiento en distintos nodos con diferentes parámetros. Por otro lado, se observó que la paralelización de las operaciones

[1] Un centroide se define como el punto equidistante de los objetos pertenecientes a un grupo específico.

elementales mejora el rendimiento computacional de los algoritmos. Con lo anterior, es posible reducir el tiempo de ejecución de un algoritmo de agrupamiento sin reducir la calidad de los resultados. Finalmente, y a pesar de que el algoritmo distribuido está enfocado a resolver problemas de agrupamiento, la biblioteca descrita en la sección 3.5 es de propósito general, y puede ser extendida para su posterior aplicación a otras áreas del conocimiento.

3.1. Representación matemática de datos ómicos

El análisis de los datos provenientes del estudio de un microarreglo (ver página 21 de la sección 2.6) permite identificar genes que se expresan de manera específica ante un conjunto de condiciones experimentales, circunstancias patológicas o cambios ambientales. La consecuencia de este tipo de estudios es la comprensión de las relaciones que persisten entre el genotipo y el fenotipo [57]. De manera general, este tipo de análisis involucra conjuntos de datos de alta dimensionalidad, los cuales deben ser preprocesados antes de llevar a cabo algunas tareas de análisis con base en el proceso *Knowledge Data Discovery* (KDD) [69].

Para poder llevar a cabo esta tarea, se ha optado por técnicas de MD cuyo objetivo es la clasificación y agrupación de patrones sobre datos de alta dimensionalidad. De esta manera, el conjunto obtenido del análisis de microarreglos es organizado en una «matriz de expresión genética» $H \in \mathbb{R}^{n \times m}$, tal que

$$H := \begin{pmatrix} h_{11} & h_{12} & \cdots & h_{1m} \\ h_{21} & h_{22} & \cdots & h_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ h_{n1} & h_{n2} & \cdots & h_{nm} \end{pmatrix}, \tag{3.1}$$

donde la i-ésima fila contiene datos de un gen específico y la j-ésima columna representa una condición experimental. Sean $G = \{g_1, \ldots, g_n\}$ y $Z = \{z_1, \ldots, z_m\}$, los conjuntos de n genes y m condiciones experimentales a las que han sido sometidos los genes, respectivamente. Entonces, para cada par (i, j), el valor $h_{i,j} \in H$ representa el nivel de expresión del gen g_i en la condición experimental z_j.

La matriz H puede contener ruido, valores nulos y variaciones sistemáticas, producidas durante la ejecución de los experimentos y, por lo general, hay muchos más genes que condiciones. El procesamiento previo de los datos es esencial antes de aplicar cualquier técnica de análisis bioinformático. Esto permitirá definir hipótesis sobre las posibles vías de flujo de información entre los genes involucrados.

3.2. Antecedentes

El «análisis de agrupamientos», es una tarea de MD que permite identificar agrupaciones naturales dentro de un conjunto de datos multidimensionales a través de métricas de similitud [104]. Por ejemplo, la distancia euclidiana o la distancia de

Hamming [205]. La agrupación de estos datos es un problema $\mathcal{NP}$-hard debido a que no se tiene conocimiento *a priori* del número de grupos o de las características de los datos [84]. Se considera a este proceso de agrupación como una tarea de aprendizaje no supervisado [2]. Los algoritmos de agrupamiento se basan en dos técnicas conocidas como agrupamiento jerárquico y agrupamiento particional [73, 133]. Las técnicas enfocadas en agrupamiento jerárquico generan un árbol denominado «dendograma» [3] mediante el uso de técnicas heurísticas de división y fusión [106]. Además de ser técnicas robustas a la presencia de ruido en conjuntos de datos, estos tipos de algoritmos son independientes de las condiciones iniciales y no necesitan la definición del parámetro K. Sin embargo, son computacionalmente caros ($O(mn^2 \log n)$ en tiempo y $O(mn^2)$ en espacio) [215] y, por lo tanto, no son útiles en el análisis de conjuntos de datos con un gran número de patrones. Por otro lado, los algoritmos de agrupamiento particional dividen el conjunto de datos en K grupos mediante la optimización de una función objetivo, por ejemplo el error cuadrático medio, por lo que son considerados como problemas de optimización. A diferencia de los algoritmos jerárquicos, los algoritmos particionales son sensibles a los parámetros iniciales y a la presencia de ruido en los datos. Las técnicas de agrupamiento particional son más populares que las técnicas jerárquicas durante el análisis de reconocimiento de patrones [142]. Este trabajo se enfoca en algoritmos de agrupamiento particional.

Sea $e_p \in \mathbb{R}^m$ un patrón o vector de características definido por una única observación en el espacio m-dimensional (en el estudio de expresión genética, cada gen puede verse como un patrón). De este modo, una característica o atributo es el componente individual de un patrón. Por otro lado, se define un «grupo» como un conjunto de patrones cuyas características se describen como parecidas basado en una métrica de similitud.

El problema de agrupamiento se define formalmente a continuación.

Sea $E = \{e_1, e_2, \ldots, e_n\}$ un conjunto de patrones, donde $e_p \in E$, $p \in \{1, \ldots, n\}$, es un patrón en el m-ésimo espacio dimensional de características y n es el número de patrones en E; entonces, el problema de agrupamiento es el particionamiento de E en K grupos $C = \{c_1, \ldots, c_K\}$, $c_i \subseteq E$, $i \in \{1, \ldots, K\}$, $K \leq n$, tal que se satisfacen las siguientes condiciones:

- Cada patrón debe ser asignado a un grupo, es decir, $\bigcup_{k=1}^{K} c_k = E$
- Cada grupo tiene al menos un patrón asignado, es decir, $c_k \neq \{\emptyset\}, k \in \{1, \ldots, K\}$
- Cada patrón es asignado a un y solo un grupo, es decir, $c_i \bigcap c_j = \{\emptyset\}$, $i, j \in \{1, \ldots, K\}$, $i \neq j$. Esta propiedad se denomina *agrupamiento duro*

[2] El aprendizaje no supervisado es un método de aprendizaje automático donde un modelo se ajusta a las observaciones. Se distingue del aprendizaje supervisado por el hecho de que no existe un conocimiento *a priori* [76].

[3] Un dendograma es una representación gráfica que resume el proceso de agrupación en el análisis de grupos. Los objetos similares se conectan mediante enlaces cuya posición en el diagrama está determinada por el nivel de similitud o disimilitud entre los objetos [199].

Algoritmo 1 Generalización del algoritmo de agrupamiento de particiones iterativas.

Entrada: $E = \{e_1, \ldots, e_p, \ldots, e_n\}$: conjunto de datos donde $e_p \in \mathbb{R}^m$, n es el número de patrones en E, y K es el número de agrupaciones que se desea obtener.

Salida: $C = \{c_1, \ldots, c_K\}$: conjunto de K agrupaciones tal que, $\bigcup_{k=1}^{K} c_k = E$

1: **procedure** PARTITIONALCLUSTERING
2: Inicializar aleatoriamente los K centroides de cada agrupación, $\{m_1^{(0)}, \ldots, m_k^{(0)}\}$, donde $k \in \{1, \ldots, K\}$.
3: **while** Mientras un criterio de convergencia no se cumpla **do**
4: **for** Cada patrón e_p, en E **do**
5: Calcular la función de pertenencia $u(m_k^{(T)}|e_p)$ del patrón e_p a $m_k^{(T)}$
6: Calcular la función de peso $w(e_p)^{(T)}$ del patrón e_p a cada uno de las agrupaciones $m_k^{(T)}$
7: **End for**
8: Recalcular los K centroides con la función $m_k^{(T)} = \dfrac{\sum_{\forall e_p} u(m_k^{(T)}|e_p) w(e_p)^{(T)} e_p}{\sum_{\forall e_p} u(m_k^{(T)}|e_p) w(e_p)^{(T)}}$
9: **End while**
10: **End procedure**

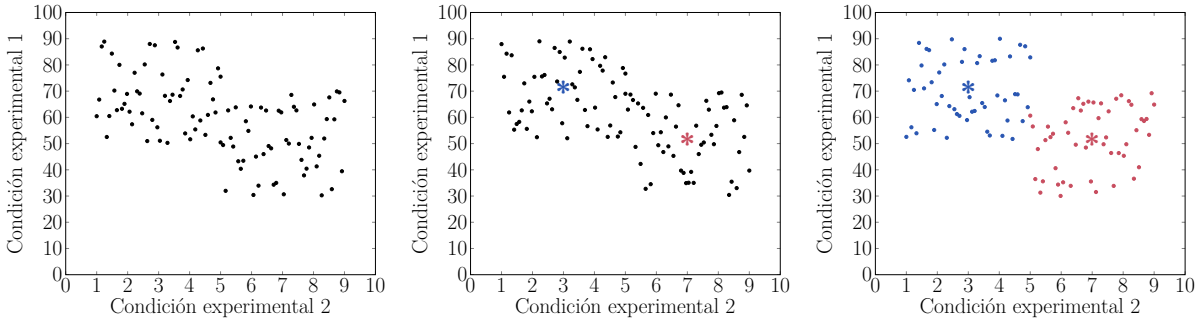
Cabe mencionar que los algoritmos de agrupamiento particional son algoritmos iterativos que convergen a óptimos locales. El pseudocódigo 1 describe la forma general de agrupamiento particional iterativo utilizado en [88]. En general, el algoritmo inicia con la selección aleatoria de K centroides, y alterna cada paso entre una etapa de asignación y una de refinamiento. En la etapa de asignación, cada patrón es como su nombre lo indica, asignado a un centroide tal que, el método $u(m_k|e_p)$ cuantifica la pertenencia de $e_p \in E$ a un centroide $m_k \in \mathbb{R}^m$. De esta forma se define cada una de las agrupaciones $c_k \subseteq E$, $k \in \{1, \ldots, K\}$, las cuales además deben satisfacer las siguientes restricciones:

- $u(m_k|e_p) \geq 0, \quad p \in \{1, \ldots, n\}$ y $k \in \{1, \ldots, K\}$
- $\sum_{k=1}^{K} u(m_k|e_p) = 1, \quad p \in \{1, \ldots, n\}$

En la etapa de refinamiento cada centroide m_k es actualizado con base en los patrones asociados. Los dos pasos son iterados hasta que se alcanza un criterio de convergencia, los cuales son abordados en la sección 3.3.4.

En el pseudocódigo 1 considere que m_k en la $T-$ésima iteración se describe como $m_k^{(T)}$ tal que, el valor de un centroide al inicio del algoritmo es $m_k^{(0)}$.

El dominio del método $u(m_k|e_p)$ en los algoritmos *agrupamiento duro* se define en el intervalo $(0, 1)$; mientras que, para los algoritmos descritos como «agrupación difu-



(a) Conjunto de datos original **(b)** Inicialización de centroides **(c)** Asignación de patrones

Figura 3-1: Ejemplo de ejecución de KM. Se describe un ejemplo de la ejecución de KM para un conjunto de 100 observaciones en $\mathbb{R}^2$ y para $K = 2$. Específicamente en la figura 3-1a se describe el conjunto inicial de dados; en la figura 3-1b se representa la elección de los centroides con una X. Esta elección se realiza de manera aleatoria y se calcula la métrica de similitud; finalmente en la figura 3-1c se describe la asignación de patrones al centroide seleccionado.

sa» se tiene que $u(m_k|e_p) \in [0, 1]$. Por otro lado, el método de peso, $w(e_p)$ propuesto por Zhang en [236], mide cuánta influencia tiene el patrón e_p al volver a calcular los centroides en la siguiente iteración, tal que $w(e_p) > 0$. Generalmente, se utilizan diferentes criterios de convergencia en un algoritmo de agrupamiento particional iterativo.

En la figura 3-1 se describe un ejemplo de la ejecución de KM para un conjunto de 100 observaciones en $\mathbb{R}^2$ y para $K = 2$. Específicamente en la figura 3-1a se describe el conjunto inicial de datos; en la figura 3-1b se representa la elección de los centroides con una X. Esta elección se realiza de manera aleatoria y se calcula la métrica de similitud; finalmente en la figura 3-1c se describe la asignación de patrones al centroide seleccionado una vez que un criterio de convergencia se cumple.

3.2.1. Métricas de similitud

Una métrica de similitud es el componente fundamental en los algoritmos de agrupamiento, debido a que permite la evaluación del parecido de los patrones que pertenecen a cada uno de los grupos [76]. La forma más popular de evaluar el nivel de similitud entre los patrones que pertenecen al mismo cúmulo es el uso de métricas de distancia. La más utilizada es la distancia euclidiana, descrita por las ecuaciones 3.2.

$$d(e_u, e_w) = \sqrt{\sum_{j=1}^{m}(e_{u,j} - e_{w,j})^2} = \|e_u - e_w\|. \tag{3.2}$$

La distancia euclidiana es un caso especial (cuando $\alpha = 2$) de la métrica de Minkowski [54] definida por las ecuaciones 3.3, y cuando $\alpha = 1$, la métrica se refiere

a la distancia de Manhattan [201].

$$d^{\alpha}(e_u, e_w) = \left(\sum_{j=1}^{m} (e_{u,j} - e_{w,j})^{\alpha} \right)^{1/\alpha} = \|e_u - e_w\|^{\alpha}. \tag{3.3}$$

Se ha demostrado que la métrica de Minkowski no es eficiente en agrupamiento de datos de alta dimensionalidad debido a que la distancia entre los patrones se incrementa con el aumento de la dimensionalidad [161]. Para la métrica de Minkowski, la característica de mayor escala tiende a dominar a las otras características. Esto se puede resolver normalizando las características a un rango común mediante el uso de la distancia del coseno, la cual se define como la suma del producto de cada componente de dos vectores definidos, tal como lo describe la ecuación 3.4.

$$< e_u, e_w >= \frac{\sum_{j=1}^{m} e_{u,j} e_{w,j}}{\| e_u \| \| e_w \|}, \tag{3.4}$$

donde $< e_u, e_w > \in [-1, 1]$.

La distancia coseno no es en realidad una distancia sino una métrica de similitud. En otras palabras, la distancia del coseno mide la diferencia en el ángulo entre dos vectores, más no la diferencia en la magnitud entre esos vectores. Entonces la distancia del coseno es adecuada para agrupar datos de alta dimensionalidad [89].

Por otro lado, la distancia de Mahalanobis se define como

$$d_M(e_u, e_w) = (e_u - e_w)\Sigma^{-1}(e_u - e_w)^T, \tag{3.5}$$

donde Σ representa a la matriz de covarianzas de los patrones. La distancia de Mahalanobis proporciona diferentes características de diferentes pesos en función de sus variaciones y correlaciones lineales por pares. Esa métrica supone implícitamente que las densidades de cada una de las agrupaciones es una función Gaussiana multivariada.

En la sección 3.3 se describen los algoritmos de MD enfocados en tareas de agrupación que mayor influencia tienen en Bioinformática. Para cada uno de ellos, se han definido las funciones de pertenencia y peso que se utilizan en el pseudocódigo 1. Finalmente, en la sección 3.4 se describe la paralelización del algoritmo iterativo general descrito en esta sección. En este sentido, se aborda una versión distribuida utilizando el enfoque de paso de mensaje. En la sección 3.5 se añade la propuesta de una biblioteca que incluye las operaciones elementales optimizadas por mecanismos de HPC. En la sección 3.5.2 se presenta una discusión sobre el rendimiento computacional en distintas arquitecturas.

3.3. Particularidades de los algoritmos de agrupamiento

En esta sección se describen la función de pertenencia ($u(m_k|e_p)$) y la función de peso (w_{e_p}) que requiere cada uno de los algoritmos de agrupamiento más populares en Bioinformática. Dada la fuerte dependencia entre el resultado y la elección inicial de los centroides (m_k), una práctica común es ejecutar el algoritmo varias veces y seleccionar de entre todas las soluciones, aquella que se considere como la mejor.

Se debe tomar en cuenta que para el total de las iteraciones (R), para cada uno de los patrones e_p y para cada una de las características z_i de cada patrón, el algoritmo calcula la diferencia con cada uno de los centroides ($u(m_k|e_p)$). Por lo tanto, la complejidad computacional es $O(mKnR)$, donde R se considera como un criterio de convergencia y representa el número máximo de iteraciones definido por el usuario.

Es necesario tener en consideración las siguientes observaciones.

Observación 1 (R1)**.** *Cuando un centroide m_k no es seleccionado en la iteración inicial del algoritmo tiene la posibilidad de que nunca sea seleccionado. Por ejemplo, cuando se elige un centroide m_k lejos de los otros centroides y patrones. La solución es reemplazarlo con un nuevo centroide generado aleatoriamente. Sin embargo, a medida que crece la densidad de los patrones, el riesgo de encontrar este caso singular se vuelve cada vez menos probable.*

Un enfoque más sofisticado para proponer una solución a la observación 1 es la implementación de algoritmos como *K-medias++* [127], el cual selecciona un patrón e_p como centroide inicial m_k, y los restantes se seleccionan de tal forma que se maximice la distancia entre centroides.

Observación 2 (R2)**.** *Las características de los patrones pueden ser ponderadas para obtener un grupo donde algunas de ellas tengan más importancia que otras. Dado un patrón e_p, el patrón puede ser reemplazado por Ae_p donde A es una matriz diagonal $m \times m$ cuyas entradas a_{ii} escalan los pesos de las diferentes características. Como resultado, la función objetivo utilizará el argumento $d(Ae_p, Am_k)$.*

3.3.1. Algoritmo «K-medias»

El algoritmo de agrupamiento, *K-medias* (KM), es un problema computacionalmente difícil (NP-hard) y ha sido ampliamente utilizado en Bioinformática [100, 108, 123, 181, 194, 227]. Sin embargo, se han propuesto heurísticas que convergen rápidamente a un óptimo local mediante un refinamiento iterativo [140]. KM es un algoritmo de aprendizaje no supervisado, por lo tanto, no se cuenta con información relativa a la etiqueta de la clase y en consecuencia, KM se dará a la tarea de agrupar el conjunto de patrones e_p, $\forall p \in \{1, \ldots, n\}$ en K grupos con base en sus características. La naturaleza de KM hace que sea susceptible a valores atípicos debido a que se distorsiona la distribución de los datos.

La función a optimizar es una métrica de similitud basada en el error cuadrático medio. Dicha función se describe en la ecuación 3.6.

$$f(C, M) = \sum_{k=1}^{K} \sum_{e_p \in c_k} d^2(e_p, m_k), \tag{3.6}$$

donde $M = \{m_1, \ldots, m_K\}$ es el conjunto de centroides.

El algoritmo consta de tres pasos (ver pseudocódigo 1): (1) Inicialización, donde se seleccionan de forma aleatoria los $m_k,\ k \in \{1, \ldots, K\}$ centroides que representan a cada una de las K agrupaciones según la iteración dada; (2) Asignación de cada e_p patrón al m_k más cercano; (3) Actualización de los centroides. Este último paso tiene diferentes variantes, siendo la más utilizada la selección del nuevo centroide como la posición del promedio de los objetos pertenecientes a dicho grupo. Los pasos (2) y (3) se repiten hasta que los centroides no presentan algún cambio ni el valor de la ecuación 3.6 es menor a un umbral (ρ) establecido previamente por el usuario.

El método de pertenencia se define por la ecuación 3.7.

$$u(m_k|e_p) = \begin{cases} 1, & \text{si } d^2(e_p, m_k) = \min\limits_{i \in \{1,\ldots,K\}} \left\{ d^2(e_p, m_i) \right\}, \\ 0, & \text{en otro caso.} \end{cases} \tag{3.7}$$

KM es descrito como un algoritmo *agrupamiento duro*. Es decir que un patrón e_p pertenece exclusivamente a un grupo en una iteración dada. El método de peso (ver ecuación 3.8) es constante, por lo tanto, todos los patrones en un grupo tienen el mismo nivel de importancia.

$$w(e_p) = 1 \tag{3.8}$$

La principal ventaja de este algoritmo es su complejidad temporal ($O(n^2)$), lo cual lo hace adecuado para conjuntos de datos muy grandes. Sin embargo, KM es considerado como un algoritmo voraz debido a que siempre selecciona de entre los posibles centroides, aquel que represente la mínima distancia. Este algoritmo depende de las condiciones iniciales y de las características de los datos. Lo anterior puede hacer que el algoritmo converja a soluciones subóptimas. Desafortunadamente, KM requiere la definición del parámetro K por parte del usuario.

3.3.2. Algoritmo «Medias-C difusas»

En 1980, Bezdek propuso una versión difusa de KM denominado *medias-C difusas* [31] (FCM), el cual ha sido ampliamente utilizado en el análisis de expresión genética [6, 27, 114, 122, 123]. Este algoritmo se basa en una extensión difusa del criterio que se utiliza para calcular el error cuadrático medio. La principal ventaja de FCM sobre KM es la asignación de cada patrón a cada grupo con algún grado de pertenencia. Esta característica resulta ser más adecuada para aplicaciones reales donde existen superposiciones entre las agrupaciones.

Las funciones de pertenencia de una partición difusa de E en K grupos satisfacen las siguientes propiedades:

- $0 \leq u(m_k|e_p) \leq 1,\ \forall k \in \{1, \ldots K\}, p \in \{1, \ldots, n\}$
- $0 < \sum_{j=1}^{n} u(m_k|e_j) < 1,\ \ \forall k \in \{1, \ldots K\}$
- $\sum_{i=1}^{K} u(m_i|e_p) = 1,\ \ \forall p \in \{1, \ldots, n\}$

Con el objetivo de proponer una partición difusa óptima de E, se han descrito algunos criterios de agrupación, sin embargo, la función objetivo más común para el enfoque FCM se describe en la ecuación 3.9, la cual está asociada al error cuadrático medio.

$$f(E, M) = \sum_{k=1}^{K} \sum_{\forall p=1}^{n} u(m_k|e_p)^q d^2(e_p, m_k), \tag{3.9}$$

donde q se define como el coeficiente difuso, tal que $q \geq 1$. El incremento del valor q hace que el algoritmo sea más difuso. El método de pertenencia se define por la ecuación 3.10.

$$u(m_k|e_p) = \frac{\parallel e_p - m_k \parallel^{-2/(q-1)}}{\sum_{k=1}^{K} \parallel e_p - m_k \parallel^{-2/(q-1)}}. \tag{3.10}$$

FCM es un algoritmo clasificado como agrupación difusa, por lo que un patrón e_p puede pertenecer a más de un grupo al mismo tiempo. El método de peso se describe en la ecuación 3.11.

$$w(e_p) = 1, \tag{3.11}$$

Esta ecuación hace que la función de pertenencia de FCM sea suave y su función de peso constante. En general, este algoritmo tiene mejor rendimiento que KM y se ve menos afectado por la presencia de incertidumbre en los datos. FCM requiere que el usuario especifique el número de grupos (K) y por lo tanto puede converger a los valores óptimos locales.

Tal como se describe en el pseudocódigo 1, FCM minimiza iterativamente la ecuación 3.9 hasta obtener una partición difusa óptima. El peso asociado a la métrica de similitud ($u_{k,p}^q$), es la q-ésima potencia del grado de pertenencia del p-ésimo patrón al k-ésimo grupo. Cuando $q \to 1$ la partición óptima es cada vez más cercana a una partición exclusiva, mientras que cuando $q \to \infty$ la partición óptima se aproxima a una matriz con todos sus valores iguales a $\frac{1}{n}$. Generalmente $q \in (1, 30]$ y cada valor particular en q denota un algoritmo FCM específico.

3.3.3. Algoritmo «Maximización de la Similitud Gaussiana»

Maximización de la Similitud Gaussiana (EM) es un algoritmo popular en Bioinformática [41, 137, 147, 157, 219] debido a su capacidad para estimar parámetros en

presencia de datos desconocidos. EM divide el conjunto de datos en grupos a través de una mezcla de funciones Gaussianas que se ajustan al conjunto de datos. Cada función Gaussiana tiene una matriz de medias y una de covarianzas.

La función objetivo que EM optimiza según lo definido por Hamerly y Elkan en [188] está definida por

$$f(E, M) = -\sum_{p=1}^{n} \log \left(\sum_{k=1}^{K} P[e_p|m_k]\, P[m_k] \right), \tag{3.12}$$

donde $P[e_p|m_k]$ es la probabilidad de que e_p sea generado por una distribución Gaussiana con centroide m_k, y $P[m_k]$ es la probabilidad *a priori* del centroide m_k.

El algoritmo descrito en el pseudocódigo 1 comienza con la estimación inicial de los parámetros. Enseguida se aplica un paso de «similitud» donde los valores de datos conocidos se utilizan para calcular los valores esperados de los datos desconocidos. De esta forma, los valores conocidos y esperados de los datos se utilizan para generar una nueva estimación de los parámetros. Esta fase es denominada «maximización». Finalmente, los pasos de similitud y maximización se repiten hasta la convergencia de la función objetivo (ver ecuación 3.12). El método de pertenencia es suave y se define por la ecuación 3.13 mientras que, el método de peso del algoritmo es constante (ver ecuación 3.14).

$$u(m_k|e_p) = \frac{P[e_p|m_k]\, P[m_k]}{P[e_p]} \tag{3.13}$$

$$w(e_p) = 1 \tag{3.14}$$

Algunos trabajos han demostrado que KM tiene un rendimiento comparable con EM. Se declara que EM falla en conjuntos de datos de alta dimensión debido a problemas de precisión numérica, además de que depende de la estimación inicial de los parámetros y requiere que el usuario especifique el número de grupos (K) de antemano. Finalmente, EM supone que la distribución de probabilidad de cada grupo es Gaussiana, lo que no siempre es cierto.

3.3.4. Criterios de convergencia

En lo que respecta a los métodos de convergencia de los algoritmos de agrupamiento con enfoque particional descritos en las secciones 3.3.1, 3.3.2, 3.3.3, se han propuesto métodos basados en la selección de parámetros definidos por el usuario. Para este fin, se utilizan tres enfoques basados en: (1) El número total de iteraciones definidos por el parámetro R; (2) En una tolerancia Δ tal que $u(m_k|e_p)_T - u(m_k|e_p)_{T-1} < \Delta$ cuando $\Delta > 0$; y (3) En una tolerancia $\Delta_{\text{máx}}$ tal que $f(T) - f(T-1) < \Delta_{\text{máx}}$, cuando $\Delta_{\text{máx}} > 0$. Se entiende que la f es la función objetivo descrita por las ecuaciones 3.6, 3.9, y 3.12 y que además, los valores de Δ y $\Delta_{\text{máx}}$ son muy cercanos a cero.

A pesar de que este tipo de algoritmos demuestran tener una convergencia lenta en algunas instancias, en la práctica estos casos generalmente no suelen aparecer tal

y como lo demuestra D. Arthur et al. en [9] donde describe el hecho de que KM ha alcanzado una complejidad polinomial.

3.4. Algoritmo paralelo híbrido - heterogéneo

Las implementaciones paralelas y distribuidas de los algoritmos de agrupamiento han sido motivadas por la necesidad de clasificar grandes conjuntos de datos de alta dimensionalidad. Se propone una metodología descentralizada y distribuida del esquema general descrito en el pseudocódigo 1. El objetivo es mejorar el rendimiento de los algoritmos de agrupamiento en dominios más grandes y complejos. En esta sección, se explora un modelo distribuido basado en mecanismos de paso de mensajes con la biblioteca `MPI`.

El problema a resolver se describe a continuación:

> Sea una arquitectura distribuida, se busca que cada nodo resuelva un problema de asignación de patrones a un conjunto de centroides, y que además, contribuya directamente en el refinamiento de los centroides de los otros nodos con los que interactúa. Considere que se trata de un sistema propenso a fallos, con canales de comunicación imperfectos y expuesto a violaciones en la sincronización de datos.

Una posible solución es la implementación distribuida del algoritmo de agrupamiento particional e iterativo descrito en el pseudocódigo 1. En esta solución, considere que se requieren mecanismos para difundir información entre los nodos y refinar la selección de centroides tomando en consideración los resultados obtenidos por cada uno de ellos (ver figura 3-2). Además, considere la posibilidad de que cada nodo pueda ejecutar distintos algoritmos de agrupamiento particional con distintos parámetros de forma colaborativa. Finalmente, se debe considerar que es un sistema propenso a fallas, con canales de comunicación imperfectos y expuesto a violaciones en la sincronización de datos.

Para resolver el problema descrito anteriormente, se abordó el protocolo de consenso [45] como el mecanismo óptimo que define un resultado ideal. Lo anterior se realizó tomando como base el problema agrupación por consenso (CC) y cuyo objetivo es encontrar una agrupación única de diferentes agrupaciones obtenidas previamente. Así, CC se define como la conciliación de las agrupaciones obtenidas sobre el mismo conjunto de datos procedente de diferentes fuentes o de diferentes ejecuciones del mismo algoritmo [152].

Se describen los aspectos formales utilizados en esta propuesta. Cabe mencionar que cuando el problema de CC se presenta como un problema de optimización, la agrupación de consenso se denomina también como partición media [130] y se considera como un problema $\mathcal{NP}$-completo, incluso si $K = 3$. Este método por consenso para el aprendizaje no supervisado es análogo al aprendizaje por conjuntos en el aprendizaje supervisado [187].

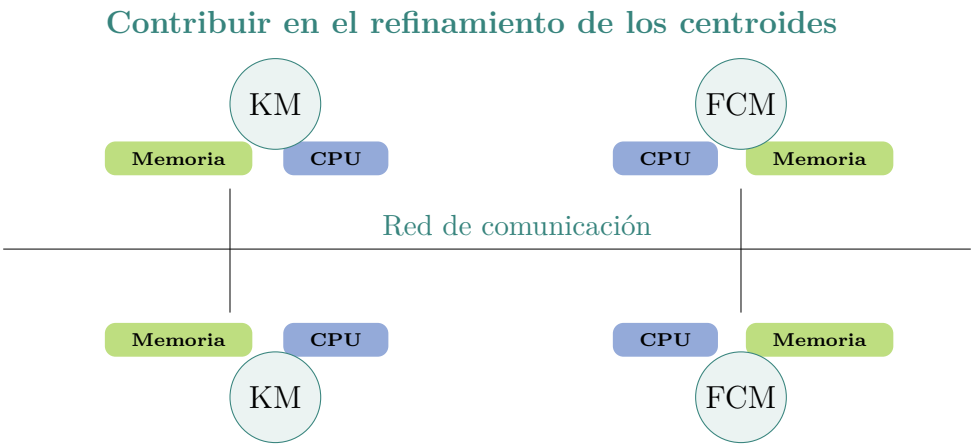


Figura 3-2: Definición del problema en el diseño de un algoritmo híbrido - heterogéneo del algoritmo iterativo básico de agrupamiento particional. La principal motivación de esta propuesta es el poder mejorar el rendimiento computacional durante el análisis de conjuntos de datos de alta dimensionalidad. En este caso, considere una arquitectura distribuida, donde cada nodo (memoria y CPU) es capaz de resolver un problema de agrupamiento (KM, FCM) de forma colaborativa para refinar los centroides. Además, tome en cuenta que es un sistema propenso a fallas, con canales de comunicación imperfectos y expuesto a violaciones en la sincronización de datos.

3.4.1. Definiciones

Sea $G = (V, A, W)$ una gráfica ponderada, donde V es un conjunto finito y no vacío de n vértices $\{v_1, \ldots, v_n\}$, A es el conjunto de aristas (v_i, v_j) y W es el conjunto de pesos $w_{i,j}$ asociados a cada arista (v_i, v_j). Se dice que una gráfica G es dirigida si existe una relación de precedencia entre los vértices; en otro caso se denomina gráfica no dirigida. Una gráfica G es conexa si para cualquier par de vértices $v_i, v_j \in V$ existe un camino que los conecte independientemente de la orientación de las aristas. Una gráfica es balanceada si para cada vértice $v_i \in V$

$$\sum_{j=1}^{n} w_{ij} = \sum_{j=1}^{n} w_{ji}, \tag{3.15}$$

es decir, la suma de los pesos de las aristas de entrada coincide con la suma de los pesos de salida.

Una gráfica G puede ser representada utilizando una matriz de adyacencia $X_{n\times n}$, cuyos elementos $x_{ij} = 1$ si $(v_i, v_j) \in A$ y $x_{ij} = 0$ en cualquier otro caso. Sea $N_i = \{v_j | \{v_i, v_j\} \in A\}$ el conjunto de vértices que son vecinos al vértice v_i.

En el resto de este documento se supone que la gráfica G es conexa y no dirigida. Se denota por T al iterador en el algoritmo distribuido de agrupamiento iterativo, mientras que t representa el iterador de un algoritmo de consenso dentro de cada paso del algoritmo distribuido.

3.4.2. El problema del consenso

El problema de consenso consiste en poner de acuerdo a un grupo de s procesos (nodos) que se comunican por paso de mensajes en un sistema distribuido [206]. Este acuerdo es producido por un protocolo de consenso y se logra incluso, también cuando existen fallos en el canal de comunicación o en la ejecución del proceso. Para cumplir con su objetivo el protocolo de consenso organiza las actividades para que todos los procesos tengan datos consistentes y completos en todo momento e incluso si existe una caída de nodos. Considere que cada nodo tiene tres estados posibles estados: líder, seguidor y candidato, entonces durante la primera fase del protocolo de consenso se inicializan todos los nodos en estado de seguidor y si en un tiempo determinado no reciben información del nodo líder, se postulan como posibles candidatos (ver gráfica 1 de la figura 3.4.2). En la segunda fase, cada nodo solicita la aprobación de los nodos que conforman su vecindario N_i para convertirse en un nodo líder (ver gráfica 2 de la figura 3.4.2) tomando el rol de candidato. En la última fase, el candidato que recibe un mayor número de votaciones cambia su estatus a líder (ver gráfica 3 de la figura 3.4.2). En este paso, el nodo líder será el encargado de orquestar todas las operaciones e intercambiar información entre los nodos seguidores.

Para garantizar la correctitud de un protocolo de consenso se deben satisfacer tres condiciones [206]: (1) Acuerdo. Todos los procesos correctos llegan a la misma decisión; (2) Finalización. Todos los procesos correctos llegan a un estado de decisión; e (3) Integridad. Si todos los procesos correctos han propuesto la misma decisión, entonces cualquier proceso correcto en el estado de decisión ha seleccionado ese valor de decisión.

De manera general y dado un conjunto de n nodos, cada uno asociado a un vértice en la gráfica G, para cualquier protocolo de consenso, en una unidad de tiempo discreta y no negativa $t > 0$, el estatus $z_i \in \mathbb{R}$ de $v_i \forall\ i = 1, \ldots, n$ y $t = 1, \ldots$ se actualiza de manera iterativa con base en la ecuación 3.16

$$z_i(t+1) = z_i(t) + y_i(N_i, t), \tag{3.16}$$

donde $y_i(N_i, t)$ es un parámetro algorítmico para resolver cualquier protocolo de consenso y que depende directamente del conjunto de nodos vecinos al i-ésimo nodo en el paso t.

Sea $Q(z_{10}, \ldots, z_{n0}) \in \mathbb{R}$ un operador que evalúa las condiciones iniciales en todos los nodos. Un problema de Q-consenso consiste en encontrar $y_i(N_i, t)$ que sea función de los estados de los nodos en el vecindario del i-ésimo nodo tal que $\lim_{t\to\infty} z_i(t) = Q(z_{10}, \ldots, z_{n0})$.

Para este caso particular, se consideran *consenso-promedio* [150] y *consenso-máximo* [222] como protocolos de consenso ampliamente utilizados en la composición de patrones locales. En las siguientes secciones se abordan las características de cada uno de ellos.

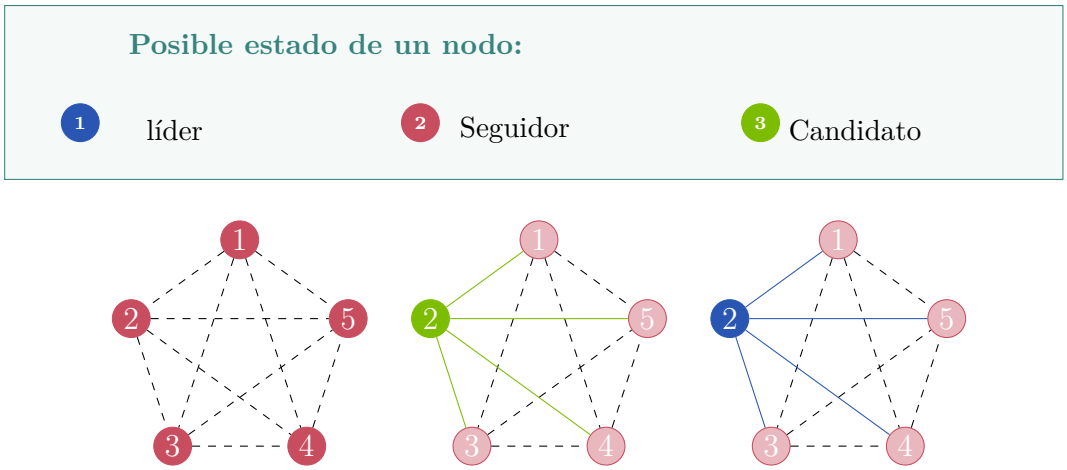


Figura 3-3: Protocolo RAFT es un protocolo de consenso que itera entre dos estados: (1) Elección de un líder y (2) Replicación de información. Considere que un nodo en un protocolo de consenso puede tener 3 posibles estados: líder, seguidor y candidato. Al inicio del consenso, todos los nodos son seguidores y si no reciben información de un líder se postulan como candidatos (ver gráfica 1). En este paso, cada nodo solicita su voto a los nodos que conforman su vecindario N_i para saber si lo aceptan como nodo líder o no (ver gráfica 2) tomando el rol de candidato; finalmente el candidato que recibe un mayor número de votaciones cambia su estatus a líder (ver gráfica 3). En este paso, el líder es el encargado de orquestar todas las operaciones e intercambiar información entre los nodos seguidores.

Protocolo *Consenso-promedio*

En el protocolo *consenso-promedio* cada uno de los nodos convergen al promedio de los componentes que determinan su estado inicial. En este caso Q es el operador *promedio* de sus argumentos. Asumiendo que la gráfica G es conexa y no dirigida o, dirigida y balanceada se debe encontrar una solución asintótica.

Sea la ecuación 3.17 el estatus de cada nodo

$$z_i(t+1) = w_{ii} z_i(t) + y_i(N_i, t), \tag{3.17}$$

tal que,

$$y_i(N_i, t) = \sum_{j=1}^{n} w_{ij} z_j(K), \tag{3.18}$$

y donde $w_{ij} = 0$ si $(v_i, v_j) \in A$. Entonces, la estrategia de actualización para el sistema completo puede ser representada como

$$z(t+1) = (W \otimes I_m) z_i(t), \tag{3.19}$$

donde la matriz $W \in \mathbb{R}$ contiene los valores de w_{ij}, I_m es la matriz identidad de dimensión $m \times m$.

De acuerdo con Wang et al. en [222], el estado de cada nodo converge asintóticamente al promedio por componentes de los estados iniciales si y solo si:

- W tiene un valor propio simple en 1 y todos los demás valores propios tienen una magnitud estrictamente menor que 1.

- Los vectores propios izquierdo y derecho de W correspondientes al eigenvalor 1 son 1_n y $\frac{1}{n}1_n$, respectivamente, donde 1_n es un vector compuesto de n elementos, y todos son iguales a uno.

Una posible elección, asumiendo que G es no dirigida y conexa y que cada nodo conoce n (o un límite superior para n), es que cada nodo i elija independientemente los términos w_{ij} como

$$w_{ij} = \begin{cases} \text{si } v_j \in N_i & \frac{1}{n} \\ \text{si } v_j \notin N_i & 0 \\ \text{si } i = j & 1 - \sum_{l=1}^{l} w_{il} \end{cases} \tag{3.20}$$

obteniendo una matriz W que satisface las condiciones que describen en [222].

Para que el cálculo promedio esté completamente distribuido, cada nodo debe tener su propio criterio de convergencia. Incluso si se maneja un protocolo de consenso de tiempo finito. En este sentido, se establecen como criterios de convergencia a los parámetro $t_{\text{máx}}$ y ϵ definidos por el número máximo de iteraciones en un algoritmo de consenso y el parámetro de convergencia para un protocolo *consenso-promedio* respectivamente [222].

Protocolo *consenso-máximo*

En el protocolo *consenso-máximo*, los nodos convergen al máximo de las condiciones iniciales, es decir Q es el operador *máximo* de sus argumentos iniciales. El problema para una gráfica conexa y no dirigida es encontrar una solución en un tiempo finito a la ecuación 3.21.

$$y_i(N_i, t) = \max_{j \in N_i} z_i(t). \tag{3.21}$$

Con la ecuación 3.21, el problema se resuelve en $r \leq n$ pasos, donde r es la longitud del diámetro de la gráfica G, es decir, el máximo entre los caminos mínimos para cada pareja de nodos que tienen comunicación, respetando la dirección de las aristas en la gráfica, si es dirigida. Sin embargo, y dado que los nodos no conocen r y n, el i-ésimo nodo en la gráfica G selecciona un valor $t_{\text{máx}} > n$.

Este formalismo representa la ejecución de *consenso-máximo* por parte del i-ésimo nodo, y se asume que todos los demás nodos están ejecutando el mismo algoritmo de manera sincrónica, cada uno con su propio estado inicial. Considerando que $|N_i| \approx n$ para cada nodo i y donde $|\bullet|$ es la cardinalidad del conjunto. Entonces la complejidad computacional es $O(nt_{\text{máx}})$.

Observación 3 (R3). *Combinando consenso-promedio y consenso-máximo es posible calcular el número de nodos necesarios en la red en una arquitectura distribuida. Específicamente, suponga que un líder es elegido a través de consenso-máximo (por ejemplo, el nodo con el identificador máximo es elegido como líder a través de consenso-máximo). Ahora, los nodos ejecutan el algoritmo de consenso-promedio con* $z_{i0} = 1$ *si el nodo* v_i *es el líder y* $z_{i0} = 0$*, en otro caso: entonces, el rendimiento de consenso-promedio para* $i = 1, \ldots, n$ *se define por*

$$lim_{t\to\infty} z_i(t) = \frac{1}{n}. \tag{3.22}$$

De esta forma, las ecuaciones 3.17, 3.19 y 3.21 pueden generalizarse fácilmente al caso vectorial, es decir, $z_{i0} \in \mathbb{R}^m$. En este caso es posible ejecutar m *consenso-promedio* o *consenso-máximo* en paralelo. Para el primer protocolo, el resultado es tal que, para un valor de $t_{\text{máx}}$ lo suficientemente grande y para un valor pequeño de ϵ, el vector $z_i(t_{\text{máx}}) \in \mathbb{R}^m$ será el promedio de los vectores iniciales. Por otro lado, para el segundo protocolo cada componente de $z_i(t_{\text{máx}})$ contendrá el valor máximo de los componentes correspondientes a los vectores iniciales.

Respecto a la complejidad computacional de los algoritmos, para cada nodo i, se tiene que $|N_i| \leq n$, y por lo tanto, para las condiciones vectoriales de $z_{i0}\mathbb{R}^m$, la complejidad computacional se define como $O(mnt_{\text{máx}})$ en cada nodo, y donde $t_{\text{máx}}$ define el total de iteraciones a ejecutar.

3.4.3. Algoritmo distribuido de agrupamiento particional

Sean n nodos representados por la gráfica conexa y no dirigida G. Considere que cada nodo contiene al menos el 70 % de patrones $e_p \in E$ y cada uno de ellos tiene un identificador único. Finalmente, considere que los parámetros de convergencia $t_{\text{máx}}$ y ϵ son idénticos en todos los nodos.

El objetivo del algoritmo distribuido de agrupamiento genérico es obtener un conjunto de centroides y de grupos tal que minimicen la función objetivo correspondiente a las ecuaciones 3.6, 3.9 y 3.12 a través de un enfoque totalmente descentralizado y distribuido. En esta sección se asume que el algoritmo distribuido descrito en el pseudocódigo 2 es ejecutado de manera síncrona por cada nodo, y que los nodos intercambian la información necesaria con sus vecinos o con el subconjunto de nodos en su vecindario definido por N_i. Se omite el procedimiento para mitigar el riesgo de obtener un mínimo local iterando el algoritmo varias veces y seleccionando el mejor resultado. Se describe un diagrama de flujo del algoritmo en la figura 3-4.

En esta propuesta se asume que,

Suposición 1 (A1). *Para cada* $T \geq 0$ *la subgráfica inducida por cada grupo* $c_j(T)$ *es conexa.*

Observación 4 (R4). *La suposición 1 se puede verificar a medida que el valor de* ρ *aumenta. En general, la subgráfica inducida por el grupo* $c_j(T)$ *probablemente se divida en varios subgrupos* $c_{jk}(T) \in \sigma(c_j(T))$*, donde* $\sigma(c_j(T))$ *es el conjunto de subgrupos que componen el grupo* $c_j(T)$*.*

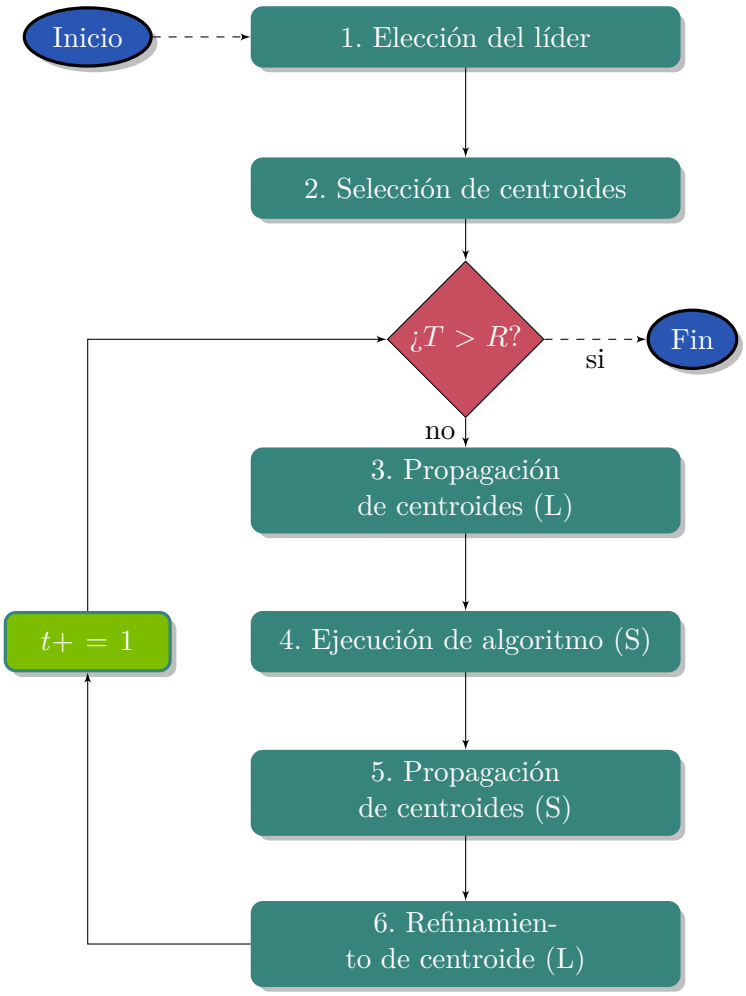


Figura 3-4: Diagrama de flujo del algoritmo distribuido de agrupamiento particional. Durante la fase de inicialización, se selecciona un nodo líder utilizando el algoritmo *maxConsensus* (**1. Elección del líder**) el cual se encargará de seleccionar los centroides iniciales de forma aleatoria (**2. Selección de centroides**). Enseguida y mientras no se alcance el número máximo de iteraciones R, en cada iteración T se difunde la información de los centroides actuales a los demás nodos a través del algoritmo *maxConsensus*, esta fase se denomina **3. Propagación de centroides**. La siguiente etapa se denomina **4. Elección de centroides**, y cuyo objetivo es el cálculo de la función de pertenencia considerando todos los patrones que se encuentran en el nodo, así como la función $u(m_k|e_p)$. Antes de finalizar la iteración, cada nodo comparte sus resultados con los nodos vecinos para ir formando una matriz de consenso (**5. Propagación de centroides**) y selecciona de entre sus vecinos en N_i, aquellos nodos que comparten la misma elección (**6. Asignación de patrones a centroides**). Finalmente, la asignación representa una gráfica de nodos conectados con la misma opción de centroide.

La desconexión de un grupo $c_j(T)$ causa la falla del algoritmo distribuido de agrupamiento particional y se debe implementar un procedimiento más complejo que incluya información sobre la estructura de cada grupo desconectado. Se describe un algoritmo distribuido de agrupamiento particional que resuelve la suposición 1.

Algoritmo distribuido

Sea `averageConsensus`$(z_i, N_i, t_{\text{máx}}, \epsilon)$ y `maxConsensus`$(z_i, N_i, t_{\text{máx}})$ los métodos de los protocolos *consenso-promedio* y *consenso-máximo* respectivamente, para el i-ésimo nodo.

Durante la fase de inicialización, el algoritmo distribuido de agrupamiento particional (ver pseudocódigo 2) comienza con la selección de un nodo líder i^* cuyo identificador es máximo. Esta elección se hace a través del método `maxConsensus` considerando que cada nodo tiene un identificador único. Enseguida, el nodo i^* selecciona el conjunto de centroides iniciales de manera aleatoria. Específicamente, i^* selecciona $m_{i^*k}(0) \in \mathbb{R}^m$ dentro del rango de valores de interés para cada componente $\forall k = 1, \ldots, K$.

Los otros nodos seleccionan $\forall k = 1, \ldots, K$

$$m_{ik}(0) = -[\infty, \ldots, \infty]^T, \ \forall i \neq i^*. \tag{3.23}$$

Después de la fase de inicialización, el ciclo principal es iterado hasta que un criterio de convergencia descrito en la sección 3.3.4 es alcanzado; en este caso se ha establecido a R como el criterio de paro y que establece el límite superior de iteraciones, tal que cada iteración $T \leq R$ está compuesto de las siguientes fases:

Propagación del centroide. En el paso T, la información de la elección del centroide actual por el nodo líder es difundida a los nodos seguidores, de tal forma que cada nodo selecciona un vector $m_i^0(T) \in \mathbb{R}^{Km}$ tal que

$$m_i^0(T) = \begin{cases} \text{si } T = 1 & \{m_{i,1}(0)^T, \ldots, m_{i,k}(0)\} \\ \text{si } T > 1 & \mu_i(T-1) \otimes m_{iK_i^*(T-1)}(T-1) \end{cases} \tag{3.24}$$

donde $m_{ik_i^*(T-1)}(T-1) \in \mathbb{R}^m$ representa el centroide seleccionado por el i-ésimo nodo en la iteración $T-1$ y $\mu_i(T-1) \in R^K$ es un vector que representa el cambio de un centroide en el paso $T-1$ $\forall k = 1, \ldots, K$. Cabe mencionar que el vector $\mu_i(T)$ se calcula con base en la función de peso $w(e_p)$ del algoritmo de agrupamiento que se esté utilizando.

Los vectores $m_i^0(T)$ están estructurados de modo que, si se usan como condiciones iniciales para el método `maxConsensus`, el vector resultante es un vector fila $m(T)$ que contiene todos los K centroides en el paso T. De hecho, en el paso $T = 1$ el único vector con las componentes más grandes que $-\infty$ es $m_i^*(T)$ y el resultado de el método `maxConsensus` es $m_{i^*}(T)$ para todos los nodos. Cuando $T > 1$ cada nodo selecciona un vector $m_i(T)$, el cual tiene componentes distintos de cero solo en correspondencia

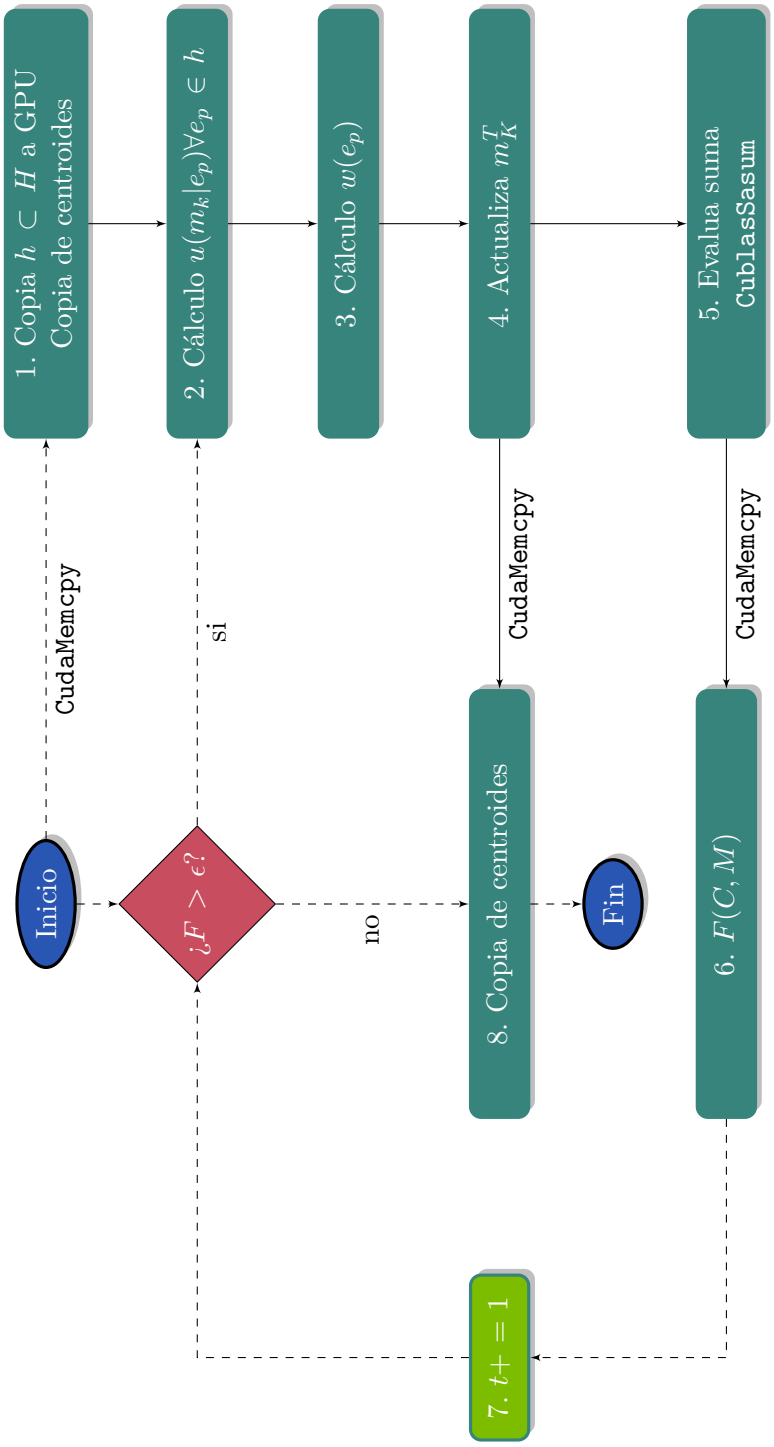


Figura 3-5: Esquematización de la implementación del algoritmo de agrupamiento en una `GPU`. Este proceso es realizado en la **etapa 4** del algoritmo distribuido. Si la arquitectura no cuenta con soporte para `CUDA`, en cada iteración del ciclo `while` se genera un conjunto de hilos que evalúan la función de pertenencia y de peso de forma paralela. Se considera que las demás operaciones se ejecutan secuencialmente. La variable m_k^T representa un arreglo de centroides. Tanto m_k^T como H son variables compartidas. El resto son privadas.

del centroide seleccionado, por lo tanto, el resultado del método `maxConsensus` es un vector pila que contiene todos los centroides.

Elección del centroide más cercano. Cada patrón es asignado al centroide más cercano entre los centroides actuales, para poder calcular el método de pertenencia $u(m_k|e_p)$ a cada uno de los patrones que se encuentran almacenados en cada nodo. El índice del centroide más cercano a un nodo es almacenado en $K_i^*(T)$, es decir

$$K_i^*(T) = u(m_K|v_i), \tag{3.25}$$

enseguida se actualiza el vector $\mu_i(T) \in \mathbb{R}^m$ con base en el método de peso $w(e_p)$ definido por el algoritmo de agrupamiento particional.

De esta forma, cada nodo comparte la elección de los centroides a los nodos de su vecindario N_i.

Propagación de la elección del centroide más cercano. En esta etapa cada nodo da a conocer su elección de $K_i^*(T)$ a los nodos de su vecindario N_i.

Elección de los nodos vecinos. Cada nodo selecciona de entre sus vecinos en N_i, aquellos nodos j que compartan la misma elección de $K_i^*(T)$. Este valor se utilizará para actualizar los centroides.

Refinamiento del centroide. El nuevo vecindario representa una subgráfica $G^m(T)$ de G, donde cada nodo está conectado solo a nodos con la misma opción de centroide. Durante esta fase, se utiliza el método `averageConsensus` sobre las observaciones de los nodos v_i sobre $G^m(T)$ para obtener $m_{iK_i^*}(t)$

Lema 1 (L1)**.** *Bajo la suposición 1, si la gráfica G es conexa, el pseudocódigo 2 resuelve el problema genérico de un algoritmo de agrupamiento particional. La complejidad computacional es $O(Kmnt_{\text{máx}}R)$ considerando que R es el número máximo de iteraciones y es el principal criterio de convergencia.*

P1. Si G es conexa, entonces el método `averageConsensus` o `maxConsensus` (posiblemente con un estado vectorial para cada nodo) sobre G tendrá éxito para un valor de $t_{\text{máx}}$ lo suficientemente grande y un valor de ϵ lo suficientemente pequeño.

Considere un paso genérico T en el ciclo principal. Durante la fase de propagación del centroide, cada nodo obtiene por medio del método `maxConsensus` un vector que contiene los K centroides actuales (para el primer paso los centroides son seleccionados aleatoriamente por el líder). Como resultado de las siguientes tres etapas, cada nodo selecciona un vecino $N_i^m(T) \subset N_i$.

Con base en la suposición 1, la gráfica $G_k^m(T)$ inducida por el vecindario $N_i^m(T)$ de todos los nodos que seleccionan el mismo centroide es conectado.

La gráfica $G^m(T)$ está dada por: $G^m(T) = \bigcup_{k=1}^{K} G_k^m(T)$ por lo que, durante la fase final de refinamiento de centroides, el método `averageConsensus` que se aplica a cada nodo v_i sobre $G^m(T)$ converge, bajo la suposición 1, al promedio de las observaciones de los nodos en su grupo, y por lo tanto al centroide del grupo en estudio.

Para el análisis de complejidad computacional del pseudocódigo 2, se observa que la operación más costosa que se realiza en el ciclo principal es la propagación de los centroides, donde el método `maxConsensus` es ejecutada con valores iniciales de dimensión Km. En este sentido, la propagación de un centroide tiene una complejidad

Algoritmo 2 Algoritmo distribuido de agrupamiento particional para el i-ésimo nodo

Entrada: $Iter_{max}$: máximo número de iteraciones; $E = \{e_1, \ldots, e_p, \ldots, e_n\}$: conjunto de patrones donde e_i es un patrón en el espacio m dimensional de características, y n es el número de patrones en E; R: Número máximo de iteraciones en un algoritmo de agrupación; $t_{\text{máx}}$: Número máximo de iteraciones para un algoritmo de consenso; ϵ: Parámetro de convergencia para un protocolo de consenso; N_i: Vecindad del nodo i, y K: número de agrupaciones a formar.

Salida: $\{m_{i1}(R), m_{i2}(R), \ldots, m_{iK}(R)\}$: conjunto de K agrupaciones tal que, $\bigcup_{k=1}^{K} C_k = E$, $N_i^{(c)(R)}$: subconjunto de N_i que contiene los nodos con las mismas elecciones del patrón al centroide más cercano en el paso R, $K_i^*(R)$: id del centroide actual más cercano al nodo i en el paso R.

1: **procedure** DISTRIBUTEDPARTITIONALCLUSTERING
2: $\mu_{ij} = 0 \forall j = 1, \ldots, k$ ▷ Inicialización
3: $m_{ij}(0) = -[\infty, \ldots, \infty]^T \forall j = 1, \ldots, K$
4: $N_i^c(0) = N_i$
5: $K_i^*(0) = 0$
6: $i^* =$MAXCONSENSUS$(i, N_i, t_{\text{máx}})$ ▷ Elección del lider
7: **if** $i = i^*$ **then**
8: Aleatoriamente inicializar los K centroides $(m_{ij} = 0 \forall j = 1, \ldots, K)$
9: **End if**
10: **for** $T = 1$ to R **do** ▷ Ciclo principal
11: $m_i^0(T) = \begin{cases} \text{si } T = 1 & \{m_{i,1}(0)^T, \ldots, m_{i,K}(0)\} \\ \text{si } T > 1 & \mu_i(T-1) \otimes m_{iK_i^*(T-1)}(T-1) \end{cases}$ ▷ Propagación de los centroides
12: $m_i(T) =$MAXCONSENSUS$(m_i^0(T), N_i, t_{\text{máx}})$
13: $K_i^*(T) = u(m_K | e_p)$ ▷ Elección del centroide más cercano
14: $\mu_{ij}(T) = w(v_i)$
15: Cada nodo entrega $K_i^*(T)$ a su vecindario N_i ▷ Envío de la elección del centroide más cercano
16: Cada nodo selecciona $N_i^c(T) \subset N_i$ considerando $K_j^*(T)$ ▷ Elección de la agrupación
17: **for** $j \in N_i$ **do**
18: $m_{iK_i^*(T)}(T) =$AVERAGECONSENSUS$(\{e_p\}, N_i^c, t_{\text{máx}}, \epsilon)$
19: **End for**
20: **End for**
21: **return** $m_{i1}(R), \ldots, m_{ik}(M), N_i^c(M), K_i^*(M)$ ▷ Refinamiento del centroide
22: **End procedure**

computacional de $O(Kmnt_{\text{máx}})$. Si R es considerado como el criterio de convergencia en el método principal, entonces algoritmo tiene una complejidad computacional de $O(Kmnt_{\text{máx}}R)$. □

Observación 5 (R5)**.** *De manera similar a un algoritmo de agrupamiento particional centralizado, existe el riesgo que algunos centroides no sean seleccionados por los nodos. En este caso, la propuesta distribuida sugiere sobreescribir cada uno de los centroides con un vector cuyos componentes sean cero.*

Lo anterior se debe particularmente a la elección del centroide $m_i(T)$; de hecho, además del líder en el primer paso, cada nodo selecciona solo un centroide y el vector $m_i(T)$ es llenado con ceros para los centroides no seleccionados. Si un centroide no es seleccionado por cualquier nodo, entonces los componentes correspondientes a $m_i(T)$ serán iguales a cero en cada nodo.

Sin embargo, este problema es fácil de solucionar. Si un centroide m_j no es seleccionado, entonces cada nodo tendrá un vector $m_i(T)$ tal que $m_j = 0$. En este caso el líder puede seleccionar otro centroide aleatoriamente y un método de propagación puede ser implementada utilizando el método `maxConsensus`*.*

3.4.4. Evaluación del algoritmo

En esta sección se describe la evaluación de la eficiencia y eficacia del algoritmo propuesto. En primera instancia se realiza el análisis de eficiencia, seguido del análisis de eficacia. Cabe mencionar que los resultados demostraron una alta eficiencia cuando los conjuntos de datos son relativamente grandes. Sin embargo, cuando el conjunto de datos es pequeño, es más caro realizar procesos de comunicación entre nodos que realizar las operaciones de cómputo en cada uno de ellos. En este caso, se recomienda el uso de un algoritmo centralizado que no requiera el uso mecanismos de paso de mensaje.

Análisis de eficiencia

Para describir la eficiencia del algoritmo se ha realizado una comparación con los resultados obtenidos por tres algoritmos: el algoritmo secuencial, el centralizado y el distribuido. Cuando del algoritmo secuencial se trata, se ha analizado la versión genérica de cada algoritmo siguiendo el enfoque de pseudocódigo 1. En la versión centralizada del algoritmo se ha considerado que solo existe un nodo para la ejecución del algoritmo.

Se han considerado dos experimentos principales: (1) En el primer experimento se ha analizado la selección de los centroides y su posterior refinamiento para los algoritmos descritos en la sección 3.3; (2) En el segundo experimento se ha abordado el análisis de convergencia de la función objetivo y finalmente, el tiempo computacional requerido para cada una de las etapas del algoritmo distribuido descritas en la figura 3-4. En ambos experimentos se analiza la convergencia de la función objetivo.

Para realizar estos experimentos se construyeron dos matrices de expresión gené-

tica *in silico* [4] (1) La primera matriz representa un conjunto de datos conformado por 100 patrones en $\mathbb{R}^2$, los cuales han sido clasificados en dos clases; (2) La segunda matriz está conformada por 50,000 patrones y 30 condiciones experimentales. Este conjunto de datos ha sido creado de manera aleatoria con números reales en el intervalo $(0, 1)$ y además no se tiene ningún conocimiento previo de los datos.

Cabe mencionar que el primer conjunto de datos ha sido creado de esa manera por conveniencia al momento de graficar el refinamiento de los centroides cuando $K = 2$. Para el primer conjunto de datos se ha considerado una arquitectura distribuida con $n = 4$ nodos. El criterio de paro de los algoritmos en este caso ha sido $R = 50$ y además, se ha establecido el parámetro $t_{\text{máx}} = 20$ y $\epsilon = 0,0001$. Finalmente, la métrica utilizada fue la distancia euclidiana descrita en la sección 3.2.1.

Para poder realizar una comparación justa de los algoritmos en el refinamiento de los centroides y debido a que los resultados de estos algoritmos dependen directamente de su elección inicial, estos fueron seleccionados de manera *a priori* ($\{(10, 50), (90, 50)\}$) durante el inicio de ejecución de los algoritmos. En la figura 3-6 se describe el refinamiento de los centroides en la iteración $T = 1, T = 3, T = 7, T = 12, T = 50$. La primera fila describe el refinamiento de KM, la segunda fila de FCM, y la tercera de EM. En este caso, cada algoritmo fue ejecutado 32 veces y se obtuvo el promedio de la posición de los centroides obtenida en cada una de las iteraciones.

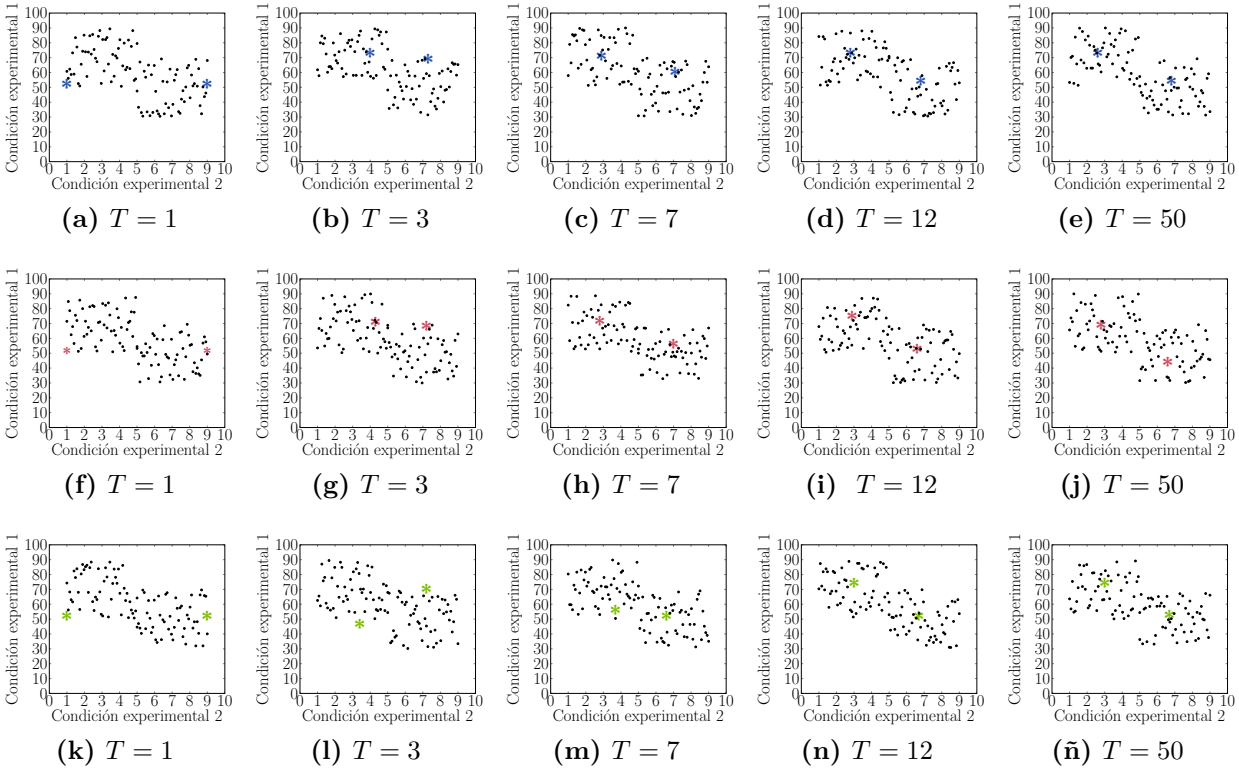
El valor promedio de los centroides es $\{(27, 70), (69, 51)\}$. Se observa que a pesar de que los algoritmos utilizan una función objetivo, una función de pertenencia y una función de peso diferente, cuando se refinan los centroides, basta de un par de iteraciones para llegar a un consenso entre cuales son los centroides óptimos, específicamente se requieren al menos siete iteraciones para este conjunto de datos.

Por otro lado, en la figura 3-7 se describe la convergencia de la función objetivo del algoritmo secuencial, centralizado y distribuido durante la implementación de KM, FCM y EM respectivamente. Cada uno de los algoritmos se ejecutó 32 veces y se obtuvo el promedio del valor obtenido por la función objetivo en cada una de las iteraciones. A pesar de que se realizaron 50 iteraciones, por conveniencia solo se reportan las primeras 20 iteraciones. Los resultados demuestran que la convergencia es similar en los tres casos (algoritmo secuencial, centralizado, distribuido). Sin embargo, se observa que la convergencia del algoritmo descentralizado es más lenta que en los otros casos. Lo anterior puede deberse a que las unidades de procesamiento requieren conocer la información de los otros nodos para poder llegar a un consenso.

Con base en la figura 3-7 se observa que el algoritmo propuesto es capaz de converger a un valor óptimo de centroides, sin embargo, para lograr su objetivo requiere un mayor número de iteraciones que la versión secuencial. Un aspecto importante a analizar es si existe un ahorro en el tiempo de ejecución cuando se analizan conjuntos de datos de alta dimensionalidad.

Por otro lado, el segundo experimento está relacionado al análisis de la segunda matriz de datos y está conformada por 50,000 patrones y 30 condiciones experimentales. Estos datos han sido generados a través de una distribución Gaussiana. La frecuencia de aparición de cada uno de los datos se representa en la figura 3-8a.

[4] La expresión *in silico* se refiere a «hecho por computadora» o vía simulación computacional.

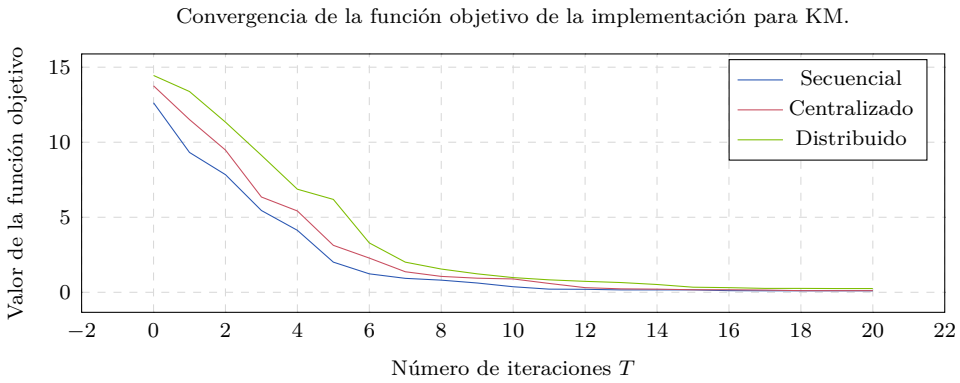


(a) $T = 1$ **(b)** $T = 3$ **(c)** $T = 7$ **(d)** $T = 12$ **(e)** $T = 50$

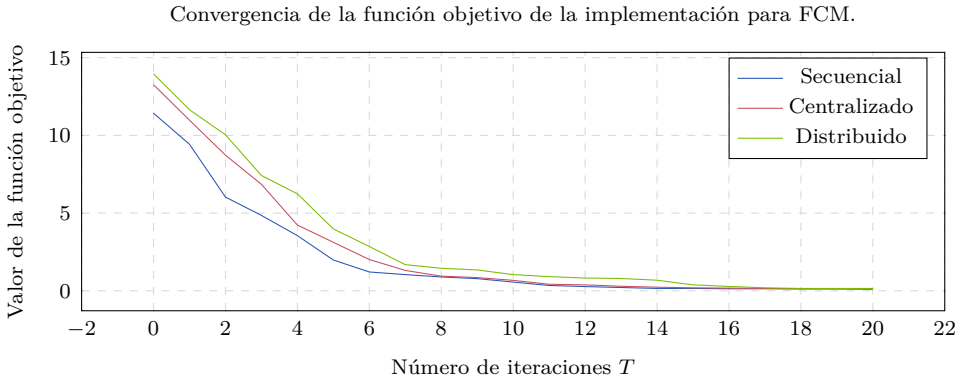
(f) $T = 1$ **(g)** $T = 3$ **(h)** $T = 7$ **(i)** $T = 12$ **(j)** $T = 50$

(k) $T = 1$ **(l)** $T = 3$ **(m)** $T = 7$ **(n)** $T = 12$ **(ñ)** $T = 50$

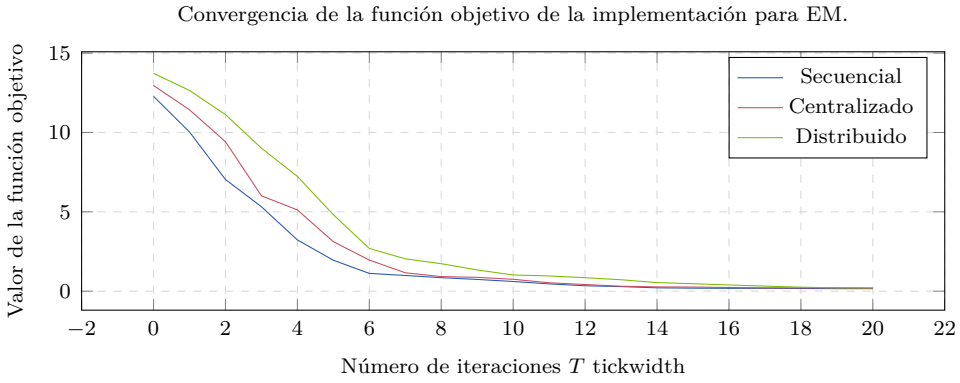
Figura 3-6: Comparación durante la etapa de refinamiento de los centroides utilizando el algoritmo distribuido en la implementación de KM, FCM y EM. La primera fila describe el refinamiento de KM, la segunda fila de FCM, y la tercera de EM. Por motivos prácticos durante la representación de los datos, se ha generado una matriz de expresión genética con 100 genes y dos condiciones experimentales clasificados en dos clases distintas (ver figuras 3-6a, 3-6f y 3-6k). Debido a que el algoritmo es susceptible a sus condiciones iniciales y con el objetivo de realizar una comparación justa, se ha definido el conjunto de centroides iniciales como $\{(10, 50), (90, 50)\}$. Se ha utilizado un parámetro $t_{\text{máx}} = 20$ y $\epsilon = 0,0001$ para los algoritmos de consenso y $R = 50$ como el total de iteraciones del algoritmo distribuido. Se reportan las iteraciones $T = \{1, 3, 7, 12, 50\}$. En este caso, cada algoritmo fue ejecutado 32 veces y se obtuvo el promedio de la posición de los centroides obtenida en cada una de las iteraciones. El valor promedio de los centroides obtenido fue $\{(27, 70), (69, 51)\}$. Se observa que a pesar de que los algoritmos utilizan una función objetivo, una función de pertenencia y una función de peso diferente, cuando se refinan los centroides, basta de un par de iteraciones para llegar a un consenso entre cuales son los centroides óptimos, específicamente se requieren al menos siete iteraciones para este conjunto de datos.



(a) Convergencia de la función objetivo de la implementación para KM



(b) Convergencia de la función objetivo de la implementación para FCM



(c) Convergencia de la función objetivo de la implementación para EM

Figura 3-7: Comparación durante la convergencia de la función objetivo en la implementación del algoritmo secuencial, centralizado y distribuido de KM, FCM y EM. Se ha utilizado un parámetro $t_{\text{máx}} = 20$ y $\epsilon = 0,0001$ para los algoritmos de consenso y $R = 50$ como el total de iteraciones del algoritmo distribuido. Para fines prácticos, solo se reporta el valor obtenido de la evaluación de la función objetivo durante las primeras veinte iteraciones. En este caso, cada algoritmo fue ejecutado 32 veces y se obtuvo el promedio de la posición de los centroides obtenida en cada una de las iteraciones. Los resultados muestran una convergencia similar en las tres implementaciones, sin embargo, se observa que la versión secuencial para este conjunto de datos es más rápida que la versión aquí propuesta.

Debido a que en este conjunto de datos no se conoce el número de clases (K), se ha explorado el número de grupos contenidos en el conjunto de datos. Para fines prácticos, se ha tomado una muestra del 20 % de los datos para poder ejecutar KM de forma secuencial. La idea es calcular el valor de la función objetivo utilizando diferentes valores de K. En la figura 3-8b se describe la varianza que existe en cada uno de los grupos. Se observa que la varianza disminuye conforme el valor de K aumenta, pero existe una curva cuando $K = 5$, la cual se denomina «*elbow*» y denota el número óptimo de grupos. Este valor indica que si $K > 5$, los grupos tienen poco valor estadístico. Esta información fue confirmada por la función `fviz_nbclust` de la paquetería `factoextra` de `R-Studio` [5].

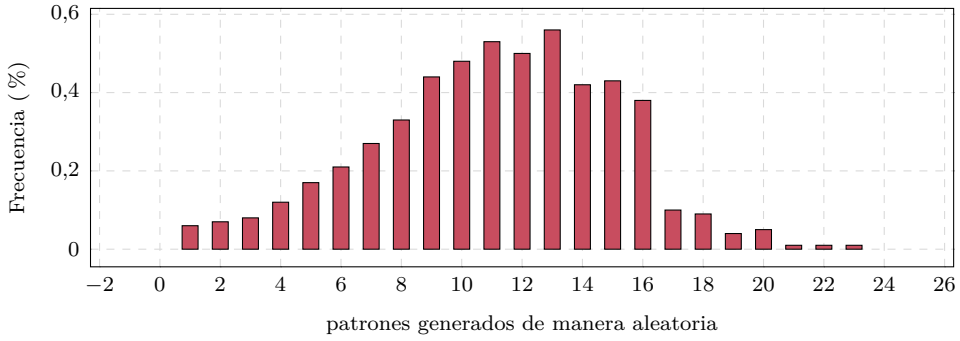
En lo subsecuente solo se reportan los resultados derivados de la ejecución de KM. En la figura 3-8c se describe la convergencia de la función objetivo utilizando el algoritmo secuencial, centralizado y distribuido respectivamente. En este caso se observa que en el algoritmo distribuido, la función objetivo converge de manera más rápida a un valor óptimo. Sin embargo, un aspecto importante que también debe ser analizado es si existe un ahorro en el tiempo de ejecución cuando se analizan conjuntos de datos de alta dimensionalidad.

En la figura 3-8d se describe el porcentaje de tiempo computacional que requieren las operaciones de comunicación, ejecución y de lectura/escritura respectivamente durante la ejecución del algoritmo distribuido enfocado en KM. Se describe el tiempo que requiere la versión secuencial, centralizada y distribuida. Se observa que el algoritmo secuencial no requiere tiempo de comunicación y que el 84 % del tiempo necesario para obtener una solución se consume durante la ejecución del algoritmo. El tiempo restante (16 %) es ocupado por las operaciones de entrada y salida del algoritmo. Por otro lado, el algoritmo centralizado, es decir, cuando $n = 1$ consume un porcentaje de tiempo mínimo (1 %). Finalmente, en promedio, el algoritmo distribuido ocupa al menos el 33 % del tiempo total de ejecución en la fase de comunicación (Comm), el 52 % del tiempo total de ejecución en la fase de ejecución (Exec) y el 15 % en tareas de escritura y lectura (I/O).

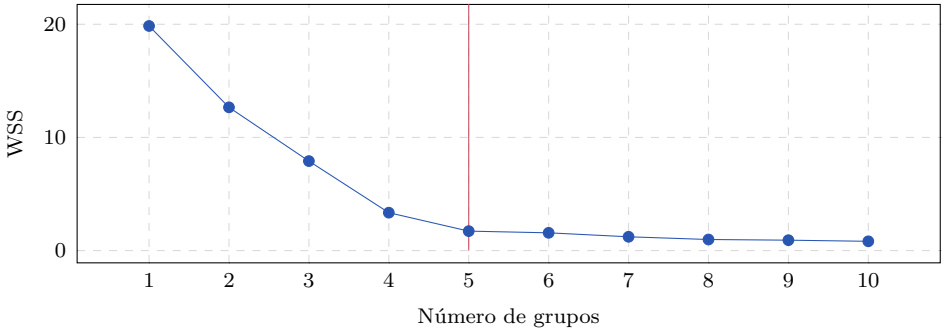
También se analiza el tiempo de ejecución necesario para cada una de las etapas descritas en el diagrama de flujo de la figura 3-4. Las siete etapas que se consideran son: (1) Elección de un nodo líder, (2) Elección aleatoria de los centroides, (3) Propagación de centroides considerando al algoritmo de consenso `maxConsensus`, (4) Elección del centroide más cercano, (5) Propagación de centroide más cercano, (6) Asignación de patrones a centroides, y (7) Refinamiento de centroides a través del algoritmo de consenso `averageConsensus`. Considere que la elección del nodo líder también utiliza el algoritmo `maxConsensus`.

En general, los resultados demuestran que la etapa (4) elección de centroides consume en promedio un 34 % del tiempo computacional necesario para obtener un resultado óptimo. La etapa (7) refinamiento de centroides consume al menos el 25 % de la ejecución total. Por otro lado, la etapa (2) elección aleatoria de centroides es la etapa que menos tiempo requiere con al menos el 2 % del tiempo computacional.

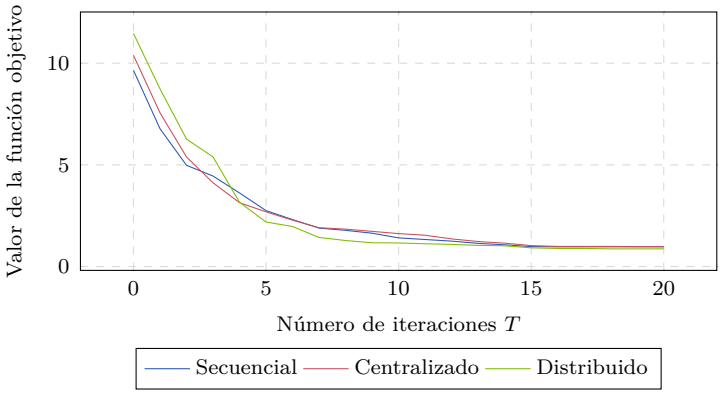
[5] `R-Studio` es un entorno de desarrollo integrado (IDE) para el lenguaje de programación `R` dedicado a la computación estadística y gráfica. [210]



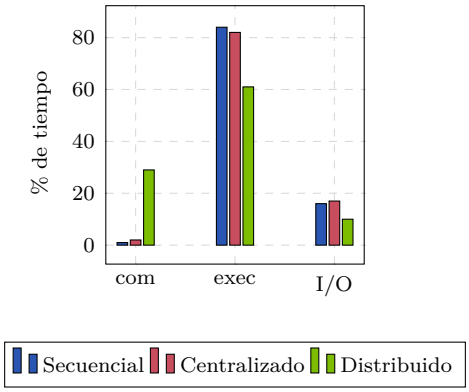
(a) Distribución de patrones generados aleatoriamente (distribución Gaussiana)



(b) Evaluación para obtener el número óptimo de grupos

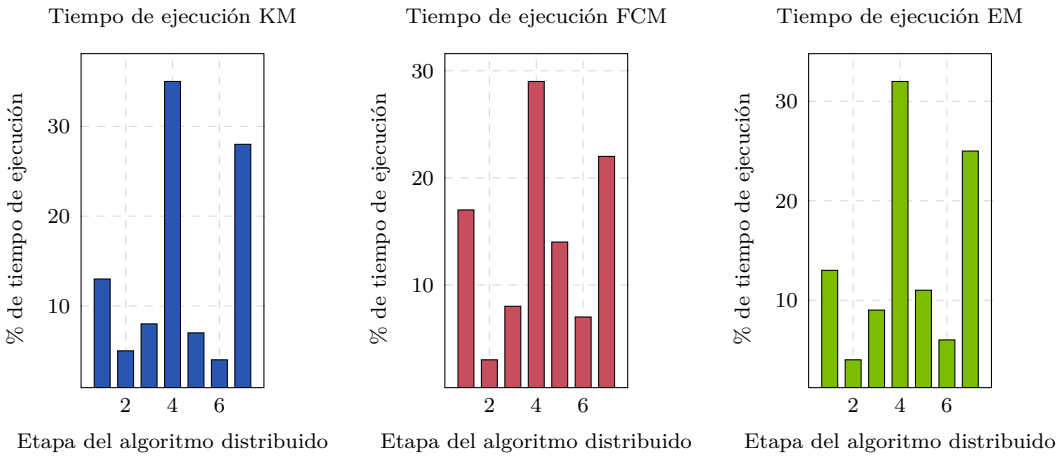


(c) Convergencia de la función objetivo para KM



(d) Porcentaje de tiempo total

Figura 3-8: Análisis del algoritmo distribuido en conjuntos de alta dimensión. Se ha creado un conjunto de 50,000 patrones y 30 condiciones experimentales a través de una distribución Gaussiana. La frecuencia de aparición de cada uno de los datos se representa en la figura 3-8a. En la figura 3-8b se describe el método para obtener el número óptimo de grupos (K), sin embargo, no se conoce información de este conjunto debido a que ha sido creado de forma aleatoria. En la figura 3-8c se describe la convergencia de la función objetivo. En este caso se observa que la función objetivo tiene una convergencia más rápida en comparación con la figura 3-7. En la figura 3-8d se describe el porcentaje del tiempo computacional que requiere la implementación secuencial, centralizada y distribuida del algoritmo enfocado en KM. Cabe mencionar que al menos el 50 % de tiempo computacional es consumido por las operaciones computacionales que lo conforman, mientras que las operaciones de comunicación al menos el 33 %.



(a) KM **(b)** FCM **(c)** EM

Figura 3-9: Comparación del tiempo de ejecución necesario para cada una de las etapas descritas en el diagrama de flujo de la figura 3-4. Los resultados demuestran que la etapa (6) asignación de patrones a centroides consume en promedio un 34 % del tiempo computacional necesario para obtener un resultado óptimo. La etapa (7) refinamiento de centroides consume al menos el 25 % de la ejecución total. Por otro lado, la etapa (2) elección aleatoria de centroides es la etapa que menos tiempo requiere con al menos el 2 % del tiempo computacional. En conclusión, las operaciones que no requieren comunicación con otro proceso o nodo necesitan al menos el 61 % del tiempo computacional. El resto es consumido por operaciones que requieren operaciones de comunicación.

Estas tres etapas no involucran procesos de comunicación con otros nodos dentro de un ambiente distribuido y representan el 61 % del tiempo total de computación, lo cual motiva el hecho de la optimización de las operaciones por mecanismos paralelos.

Por otro lado, las etapas (1), (3), (5), y (6) además de las operaciones computacionales, requieren operaciones de comunicación. Estas operaciones representan el 39 % del tiempo total de comunicación. Sin embargo, en la tabla 5.14 se observa que se puede reducir el tiempo computacional con mecanismos paralelos basados en las bibliotecas `OpenMP` y `CUDA`. Se describen seis experimentos con diferentes tamaños de matrices en los cuales se realiza la comparación entre el tiempo serial, la versión `OpenMP` y `CUDA` de KM. Para este experimento se utilizó un nodo con un procesador Xeon E5-1603 v3 con 4 núcleos en Ubuntu 18.04 y con soporte para las bibliotecas `OpenMP` 4.5 y `CUDA` 10.2. Además, se utilizó una GPU NVIDIA Quadro RTX 4000 con 2304 núcleos y 8 GB de memoria RAM DDR6 de microarquitectura Turing. Los resultados son prometedores ya que se observa una aceleración máxima de 3.96x rn versión `OpenMP` y de 56.98x en `CUDA`.

Análisis de eficacia

Para describir la eficacia del algoritmo se han utilizado tres conjuntos de datos: (1) *Patterns01.mat* caracterizado por 3,000 patrones de entrenamiento y 300 patrones de prueba; (2) *NonLinearPatterns01.mat* que es no linealmente separable. Este con-

Tabla 3.1: Comparación entre el tiempo serial, la versión OpenMP y la versión CUDA de KM. Para este experimento se utilizó un nodo con un procesador Xeon E5-1603 v3 con 4 núcleos en Ubuntu 18.04 y con soporte para las bibliotecas OpenMP 4.5 y CUDA 10.2. Se utilizó una GPU NVIDIA Quadro RTX 4000 con 2304 núcleos y 8 GB de memoria RAM DDR6 (microarquitectura Turing).

Experimento	Total	Tamaño (MB)	Serial(s)	O(s)	O+C(s)
$D_{351\times319}$	111,969	59.23	283.23	71.52 (3.96x)	6.87 (41.22x)
$D_{394\times376}$	148,144	63.88	279.86	87.31 (3.02x)	5.34 (52.40x)
$D_{482\times427}$	205,814	103.73	294.53	81.35 (3.62x)	7.31 (40.29x)
$D_{512\times468}$	239,616	117.65	387.50	118.86 (3.26x)	6.94 (55.78x)
$D_{534\times596}$	318,264	147.28	695.87	200.53 (3.47x)	12.21 (56.98x)
$D_{613\times749}$	459,137	256.87	1037.12	313.32 (3.31x)	18.68 (55.52x)

junto de datos está compuesto por un conjunto de entrenamiento de 12,500 patrones y por una muestra de patrones de prueba de 1,250; y (3) *Patterns02.mat* compuesto por 50,000 patrones de entrenamiento y 5,000 patrones de prueba. En todos los casos se utilizaron solamente el conjunto de entrenamiento debido a que se conoce la etiqueta de la clase a la que pertenece cada uno de los patrones.

En la figura 3-10 se describe la dispersión de los datos que corresponden al primer conjunto de datos (*Patterns01.mat*). Cabe mencionar que cada patrón está conformado conformado por dos características, siendo así, el eje de las ordenadas la Característica 1 y el eje de las ordenadas la Característica 2. Se puede apreciar que el conjunto de datos no es linealmente separable y que a simple vista da la impresión de estar compuesto por tres clases, razón por la cual se definió $K = 3$ considerando $R = 1000$ iteraciones, un valor $t_{\text{máx}} = 200$ y $\epsilon = 0,001$.

Por otro, en la segunda gráfica de la figura 3-10 se describe la distribución de los datos con los centroides descritos por el símbolo x. Alrededor de cada centroide se describe un círculo que engloba a aquellos patrones que pertenecen a esa clase. Los patrones que quedan fuera del círculo se clasifican como valores atípicos. En la tercera gráfica se describen las funciones Gaussianas correspondientes a cada centroide. Se observa una función de mayor tamaño, en este caso, los puntos están más cerca de la media y tienen muy poca dispersión, dicho caso corresponde al círculo más pequeño donde se aprecia que los patrones tienen un grado de similitud mayor.

Debido a que se tenía conocimiento de etiqueta de cada uno de los patrones, se obtuvo el número de patrones que habían sido clasificados de forma correcta por el algoritmo. Se observó que la clasificación del 93 % de los patrones correspondía a la etiqueta de clase correspondiente.

Para el conjunto de datos *NonLinearPatterns01.mat* se ha descrito su distribución en la figura 3-11. Se observa que está clasificado en dos clases, por lo tanto, se determinó el parámetro $K = 2$ y se consideró $R = 1000$ iteraciones, un valor $t_{\text{máx}} = 200$ y $\epsilon = 0,001$. En la segunda gráfica de la figura 3-11 se describen los centroides de cada uno de los grupos obtenidos por KM. Se ha añadido un círculo que denota los patrones que pertenecen a esa clase. Finalmente, las funciones Gaussianas obtenidas por EM comparten las mismas características, es decir, tienden a tener la misma forma y altura por lo tanto, los conjuntos de datos guardan una estrecha relación en la distribución de los patrones. Debido a que se tenía conocimiento de etiqueta de cada uno de los patrones, se obtuvo el número de patrones que habían sido clasificados de forma correcta por el algoritmo. Se observó que la clasificación del 95 % de los patrones correspondía a la etiqueta de clase correspondiente.

Finalmente, para el conjunto de datos *Patterns02.mat* se presenta la distribución de los patrones en la figura 3-12. Se observa que está clasificado en cinco clases, por lo tanto, se determinó el parámetro $K = 5$ y se consideró $R = 1000$ iteraciones, un valor $t_{\text{máx}} = 200$ y $\epsilon = 0,001$. En la segunda gráfica de la figura 3-12 se describen los centroides de cada uno de los grupos obtenidos por KM. Se ha añadido un círculo que denota los patrones que pertenecen a esa clase. Finalmente, tres de las funciones Gaussianas obtenidas por EM comparten las mismas características, es decir, tienden a tener la misma forma y altura por lo tanto, los conjuntos de datos guardan una estrecha relación en la distribución de los patrones. Debido a que se tenía conocimiento

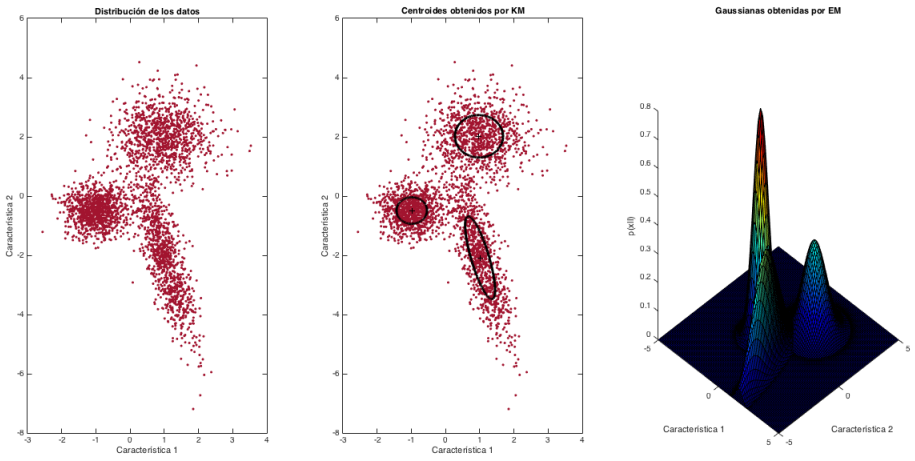


Figura 3-10: Distribución de los patrones y representación de los centroides para el conjunto de datos *Pattern01.mat*. En la primera imagen se describe la distribución de los patrones, los cuales se describen por un conjunto de 3,000 patrones con 2 características. En primera instancia se observan tres conjuntos de datos. Enseguida se ha ejecutado el algoritmo distribuido para la implementación de KM considerando que $K = 3$, un criterio de convergencia definido por un número máximo de iteraciones establecido en $R = 1000$, un valor $t_{\text{máx}} = 200$ para los algoritmos de consenso y $\epsilon = 0,001$. En la segunda imagen se describen los centroides definidos por KM. Finalmente, se describen funciones Gaussianas que describen la densidad de cada uno de los centroides. En este aspecto, la función Gaussiana de mayor tamaño representa al círculo más pequeño del conjunto de datos. Esto quiere decir que este conjunto de datos tiene muy poca dispersión de datos.

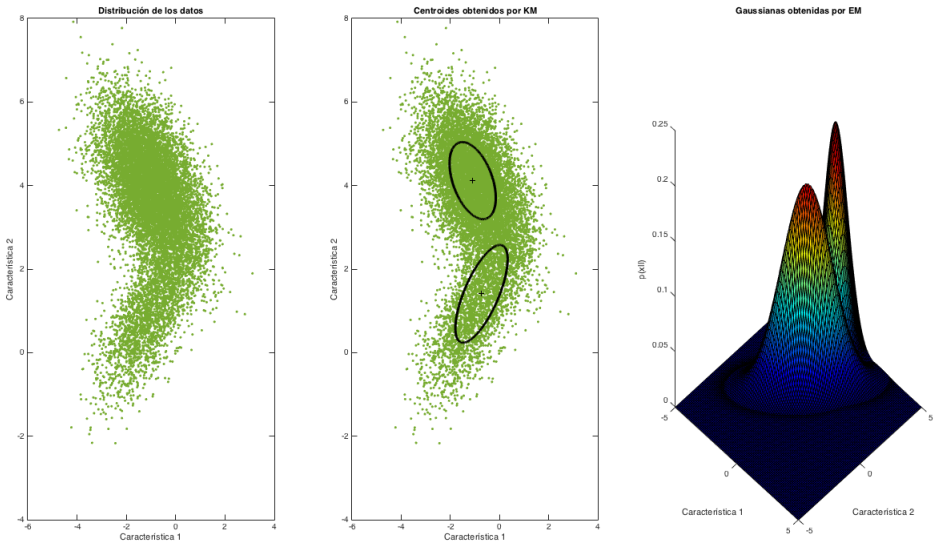


Figura 3-11: Distribución de los patrones y representación de los centroides para el conjunto de datos *NonLinearPatterns01.mat*. En la primera imagen se describe la distribución de los patrones, los cuales se describen por un conjunto de 12,500 patrones con 2 características. En primera instancia se observan dos conjuntos de datos. Enseguida se ha ejecutado el algoritmo distribuido para la implementación de KM considerando; $K = 2$, $R = 1000$, $t_{\text{máx}} = 200$ y $\epsilon = 0,001$. En la segunda imagen se describen los centroides definidos por KM. Finalmente, se describen funciones Gaussianas que describen la densidad de cada uno de los centroides. En este aspecto, las dos funciones Gaussianas tienen características similares, y por lo tanto, los conjuntos de datos guardan una estrecha relación en la distribución de los patrones.

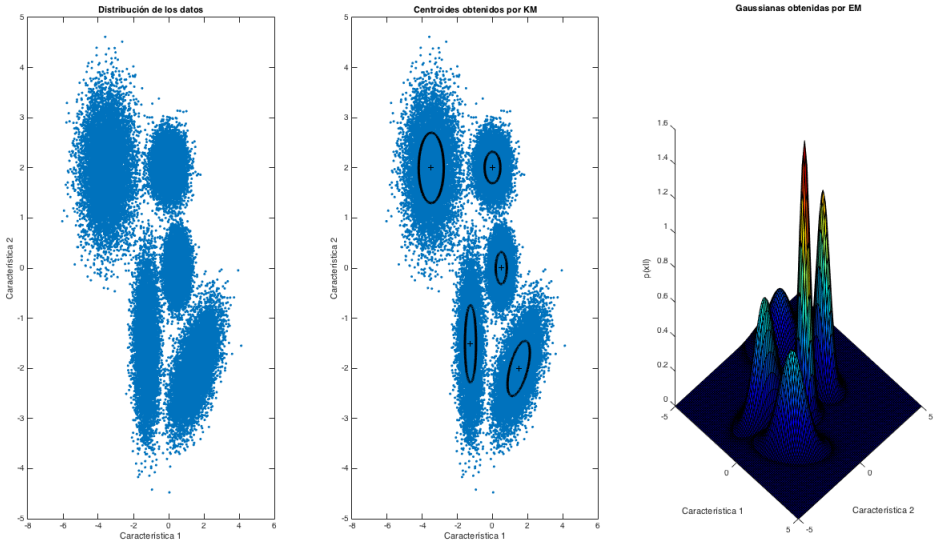


Figura 3-12: Distribución de los patrones y representación de los centroides para el conjunto de datos *Patterns02.mat*. En la primera imagen se describe la distribución de los patrones, los cuales se describen por un conjunto de 50,000 patrones con 2 características. En primera instancia se observan cinco conjuntos de datos. Enseguida se ha ejecutado el algoritmo distribuido para la implementación de KM considerando; $K = 5$, $R = 1000$, $t_{\text{máx}} = 200$ y $\epsilon = 0,001$. En la segunda imagen se describen los centroides definidos por KM. Finalmente, se describen funciones Gaussianas que describen la densidad de cada uno de los centroides. En este aspecto, la función Gaussiana de mayor tamaño representa al círculo más pequeño del conjunto de datos. Esto quiere decir que este conjunto de datos tiene muy poca dispersión de datos

de etiqueta de cada uno de los patrones, se obtuvo el número de patrones que habían sido clasificados de forma correcta por el algoritmo. Se observó que la clasificación del 91 % de los patrones correspondía a la etiqueta de clase correspondiente.

En esta sección, se ha descrito la eficacia y la eficiencia de la propuesta de un algoritmo de agrupamiento distribuido basado en CC. La principal ventaja que presenta con esta propuesta es el refinamiento de los centroides utilizando tres enfoques diferentes: (1) La ejecución distribuida de un algoritmo de agrupamiento en particular en cada uno de los nodos considerando los mismos parámetros de ejecución; (2) La ejecución distribuida de un algoritmo de agrupamiento en particular en cada uno de los nodos considerando diversas permutaciones de parámetros de ejecución; y (3) La ejecución distribuida de diversas ejecuciones de distintos algoritmos de agrupamiento en cada uno de los nodos.

Finalmente, se ha observado que el algoritmo consume al menos el 56 % del tiempo computacional total, mientras que el 44 % restante es utilizado por las operaciones de comunicación. Por lo tanto, en la sección 3.5 se describe una biblioteca de propósito general de operaciones matemáticas que mejoren el rendimiento computacional. En este aspecto, solo se han considerado aquellas que tienen influencia en los algoritmos

de agrupamiento.

3.5. Paralelización de operaciones básicas

En la sección 3.4 se describió un algoritmo distribuido de agrupamiento particional basado en mecanismos de paso de mensaje a través de la biblioteca `MPI`. La motivación de esta sección es la implementación de mecanismos híbridos de Cómputo Paralelo que permitan reducir el uso de la memoria, explotar niveles adicionales de paralelismo, mejorar la escalabilidad del algoritmo distribuido de la sección 3.4.3, equilibrar la carga de trabajo y reducir los costos de cómputo de las operaciones que no requieren procesos de comunicación. Se expone un paradigma basado en «paralelismo de datos», el cual se encarga de optimizar las operaciones vectoriales y matriciales que se ejecutan en cada nodo. Este tipo de paradigma permite ejecutar la misma secuencia de operaciones sobre un subconjunto de datos.

En este sentido, se propone una biblioteca de operaciones elementales optimizadas a través de las bibliotecas `OpenMP` y `CUDA`. Se ha optado por estas bibliotecas dado su amplio uso en el diseño de software paralelo. La biblioteca propuesta es capaz de determinar valores razonables durante la ejecución de un algoritmo, pero al mismo tiempo proporciona condiciones de optimización para las operaciones en las que el rendimiento es importante. Dichas condiciones de optimización están disponibles en tiempo de ejecución para evitar que el usuario este expuesto al proceso de compilación; de esta manera, el proceso de compilación se convierte en la interfaz de programación. Se consideran operaciones matriciales y vectoriales, las cuales son consideradas como el factor limitante en el rendimiento general de todo tipo de algoritmos y para este trabajo, de los algoritmos de MD.

Para introducir la biblioteca, considere el producto $y = A \times B \times x$, donde A y B son matrices descritas por la plantilla `matrix<>` de 15 columnas y 15 filas; además, x, y y son vectores representados por la plantilla `vector<>` de 15 elementos (ver código 3.1). Cabe mencionar que estos objetos representan elementos vectoriales y matriciales de punto flotante de doble precisión (`double`). Considerando la sobrecarga del operador $*$ que se realiza en algunas implementaciones en `C++`, la operación se resuelve de izquierda a derecha obteniendo un resultado realmente costoso al momento de obtener el producto de $A \times B$ en vez de considerar dos productos vectoriales más baratos.

Código 3.1: Código que describe el producto de dos matrices y un vector. Generalmente, este tipo de operaciones en lenguaje `C++` se realiza a través de la sobrecarga de operadores. Esto hace que se produzca un resultado realmente costoso al momento de realizar las operaciones algebraicas de izquierda a derecha.

```
1 matrix<double> A(15,15);
2 matrix<double> B(15,15);
3 vector<double> y(15);
4 vector<double> x(15);
5
6 /* Initialize A and x with data */
7 $y = A * B * x$
```

Ahora considere el código 3.2, en el cual se describe el producto de la matriz A y el vector x, el producto punto entre el vector x y el vector y y la actualización del vector y utilizando la biblioteca `uBLAS`. El método `prod` se encarga de realizar el producto matriz-vector, mientras que `innerProd` realiza el producto interno entre dos vectores. Finalmente, se realiza la actualización del vector y a través de la sobrecarga del operador $*$.

Código 3.2: Ejemplo utilizando la biblioteca `C++` incluida en la biblioteca `uBLAS`. Se realiza la declaración de una matriz A y un vector x y se omite su inicialización con datos. Se describe el producto de la matriz A y el vector x, el producto punto entre el vector x y el vector y y la actualización del vector y a través de la sobrecarga de los operadores $*$ y $+$.

```
matrix<double> A(15,15);
vector<double> x(15);

/* Initialize A and x with data */
vector<double> y = prod(A,x);
double s = innerProd(x,y);
y += s * x;
```

La biblioteca propuesta permite que el código 3.2 se ejecute con cualquiera de los paradigmas paralelos; de tal forma que el usuario final solo debe incluir los archivos de encabezado y usar el espacio de nombres correspondiente. Los métodos que se incluyen en la biblioteca se describen en la tabla 3.2. Se implementó un conjunto de operaciones elementales de manera similar a las descritas en *BLAS* [6]. Cabe mencionar que estas operaciones están clasificadas en tres conjuntos denominados «niveles», los cuales fueron descritos principalmente por Dongarra en [20]. El primer nivel se define por operaciones vectoriales ($O(n)$); el segundo nivel incluye las operaciones matriz-vector de complejidad computacional $O(n^2)$; finalmente, el tercer nivel describe operaciones matriz-matriz, es decir, operaciones de complejidad $O(n^3)$.

La biblioteca ha sido desarrollada con base en el estandar 2003 de `C++` asegurando la compatibilidad con sistemas operativos de licencia `GNU` y `GPL v3`. Es compatible con sistemas operativos `Linux`, `Windows` y `MacOS`. Se incluye soporte para las bibliotecas de `CUDA v9.4` y `OpenMP v4.0`. Tiene la capacidad de cambiar entre cualquiera de ellas en tiempo de ejecución. Cuando `CUDA` no está disponible en la arquitectura destino, las funciones que utilizan `CUDA` se deshabilitan estáticamente durante la fase compilación. Finalmente, si `OpenMP` no está disponible, también se deshabilita y la ejecución se realiza de manera secuencial.

El uso del paradigma basado en Programación Orientada a Objetos (`POO`) permite que la biblioteca sea escalable con la mínima cantidad de cambios en el código fuente. De esta forma, solo se realizan cambios en los objetos necesarios de una clase determinada. El usuario final es libre de seleccionar de entre todas las opciones disponibles en el de espacio de operaciones elementales, la que más le convenga. Las operaciones que involucran matrices y vectores densos pueden no solo llamarse para los objetos completos, sino que también pueden llamarse para submatrices y subvectores. Lo anterior permite un anidamiento casi ilimitado. Finalmente, se implementó un mecanismo de

[6] *BLAS*:*Basic Linear algebra Subprograms*

Tabla 3.2: Descripción general de las funciones implementadas por la biblioteca. Se implementó un conjunto de operaciones elementales de manera similar a las descritas en *BLAS* y aquellas que tienen utilidad dentro de los algoritmos de MD.

	Método de Álgebra Lineal
Niveles *BLAS*	1, 2, 3
Métodos de factorización	Factorización LU, factorización QR
Métodos para eigenvalores	Método QR, método de bisección, método de la potencia, método de la potencia inversa
Métodos genéricos	Producto de Strassen, producto de Kronecker

manejo de excepciones para informar de errores durante la ejecución de tareas.

Se describe la arquitectura de la biblioteca y el diseño del análisis de comparación con las bibliotecas ya propuestas en la literatura (CUSP [50] [7] y MAGMA [2] [8]).

3.5.1. Arquitectura de la biblioteca

En esta sección se describe la arquitectura básica de la biblioteca propuesta. Para comenzar, considere la operación codificada en el código 3.3 para los vectores x, y, y z.

Código 3.3: Punto de partida para describir la arquitectura de la biblioteca. Se realiza la declaración de una cuatro vectores u, x, y, y z y se omite su inicialización con datos. Se describe la suma de tres vectores en un vector a través de la sobrecarga del operador +.

```
1 vector<double> x(15);
2 vector<double> y(15);
3 vector<double> z(15);
4 vector<double> x(15);
5
6 /* Initialize x, y, z  with data*/
7 u = x+y+z;
```

Esta operación es asignada a un conjunto de operaciones predefinidas que son ejecutadas por la biblioteca correspondiente. La optimización de su ejecución a través de la biblioteca propuesta se realiza mediante la interacción de dos componentes principales: (1) Conjunto de *templates* que describen expresiones convencionales, las cuales son asignadas a un conjunto de operaciones de tipo *BLAS*; (2) Polibiblioteca, la cual describe una interfaz entre el usuario y las bibliotecas de funciones implementadas en `CUDA` y `OpenMP` los cuales se describen a continuación:

Templates. El primer componente de la biblioteca es la implementación de plantillas, las cuales definen funciones genéricas, clases y estructuras de datos. La principal

[7] `http://cusplibrary.github.io/`
[8] `http://icl.cs.utk.edu/magma/`

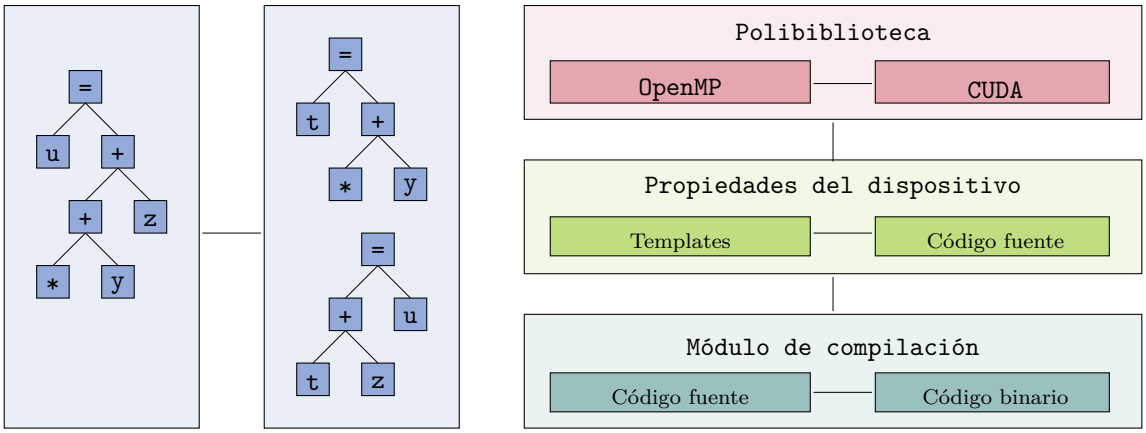


Figura 3-13: Descripción general de los pasos que requiere la biblioteca para ejecutar una operación vectorial. Considerando la expresión descrita en el código 3.3, se crea un objeto temporal `t` para almacenar el resultado de la operación `x+y`, enseguida se crea un objeto temporal `u` para almacenar el resultado `t+z`. Se implementó una «polibiblioteca» basada en `CUDA` y `OpenMP` que es capaz de administrar las diferentes representaciones del búfer de memoria.

ventaja de utilizar este tipo de mecanismos es que permiten definir algoritmos genéricos que se aplican a cualquier tipo de datos generando un polimorfismo estático y en consecuencia mejorando el rendimiento. Específicamente en la realización de cálculos en tiempo de ejecución. Considerando el código 3.3 se realiza la sobrecarga del operador + a través de `templates` de tal forma que el objeto temporal que almacena el resultado de la multiplicación del vector `x+y` es un `template` que representa un objeto relativamente barato y que a su vez, es optimizado por el compilador.

Cada `template` representa a cada una de las expresiones convencionales. Sin embargo, su ejecución se asigna a un conjunto predefinido de operaciones (ver tabla 3.3) las cuales se asemejan al conjunto de operaciones en los subprogramas básicos de Álgebra Lineal (*BLAS*) del nivel 1. Los escalares α y β pueden ser de tipo `int`, `float`, `double` o un `template` de tipo `scalar<T>`, el cual representa un solo escalar en la unidad de procesamiento. De esta forma, las expresiones algebraicas más largas se descomponen en operaciones elementales ya predefinidas logrando así el desacoplamiento de los `templates` con bibliotecas como `CUDA` y `OpenMP`.

Con base en la figura 3.5.1, la operación se descompone en dos operaciones donde el resultado obtenido a través de la polibiblioteca es almacenado en dos objetos temporales t y u respectivamente.

Polibiblioteca. Debido a que no existe un estándar en la representación de las direcciones de memoria entre `CUDA` y `OpenMP`, se implementó una clase `memoryHandle` la cual es capaz de administrar las diferentes direcciones de memoria de forma abstracta. De esta manera, un objeto de esta clase puede ser un puntero en `CUDA` o un apuntador a una dirección de memoria compartida en `OpenMP`. El ambiente a utilizar por parte de la biblioteca se define a través de las banderas `#define` durante la implementación

Tabla 3.3: Conjunto predefinido de `templates` que representan la sobrecarga de operadores y cuyo objetivo es la implementación de los operadores básicos. Se implementó un conjunto de «*templates*» que representan expresiones convencionales. Su ejecución se asemeja a operaciones tipo `BLAS` de nivel 1. Los valores A, B y C son vectores o matrices, mientras que α y β denotan valores escalares. De esta forma, las expresiones algebraicas más largas se descomponen en operaciones elementales ya predefinidas.

$A = B$ $A = \pm\alpha B$	$A+ = B$ $A+ = \pm\alpha B$	$A- = B$ $A- = \pm\alpha B$
$A = \pm B \pm C$ $A = \pm\alpha B \pm C$ $A = \pm\alpha B \pm \beta C$ $A = \pm B \pm \beta C$	$A+ = \pm B \pm C$ $A+ = \pm B \pm C$ $A+ = \pm\alpha B \pm \beta C$ $A+ = \pm\alpha B \pm \beta C$	$A- = \pm B \pm C$ $A- = \pm B \pm C$ $A- = \pm\alpha B \pm \beta C$ $A- = \pm\alpha B \pm \beta C$

del código fuente, tal como se describe en el código 3.4. Sin embargo, estas banderas son actividades desde la línea de comandos durante la compilación.

Código 3.4: Banderas utilizadas en la cabecera del código fuente para activar el ambiente a utilizar por parte de la biblioteca.

```
/* Para OpenMP */
#ifdef OPENMP
  #pragma omp parallel for
#endif
  for(long row=0; row< static_cast<long>(numRows); rows++)
  {
    unsigned int eqRow = deleteRowIndex[row];
  }

/* Para CUDA */
#ifdef CUDA
  char * cuda_mem;
#endif
```

Para poder trabajar con cada una de las direcciones de memoria, se implementó un método de escritura (`write`), uno de lectura (`read`), uno de creación de un *buffer* de memoria (`create`) y uno de copia (`copiying`). En el código 3.5 se describe la implementación de la función inline que permite copiar un búfer de memoria a otro (`copiying`). Todas estas funciones fueron implementadas utilizando la palabra reservada `inline` con el objetivo de que cada llamada a la función sea sustituida por el código implementado. Lo anterior evita que el código crezca considerablemente y acelerar la ejecución de un programa si se realizan muchas llamadas a estas funciones.

Código 3.5: Implementación de la función inline que permite copiar un búfer de memoria a otro. La función se declara de forma `inline` con el objetivo de acelerar la ejecución del programa si se realizan muchas llamadas a la función. En primera instancia se verifica que el espacio de memoria se encuentre reservado. Al finalizar, se realiza la copia de la información contenida en un búfer de memoria fuente a un búfer de memoria destino.

```
/* óFuncin que realiza la copia de un bufer de memoria a otro*/
inline int copiying(handleType const &srcBuffer, sizeT srcOffset,
    handleType &dstBuffer, sizeT dstOffset, sizeT bytesCopy)
{
  if(!assert(dstBuffer.get() != NULL) && bool("Not initialized!"))
    return 1;
  if(!assert(srcBuffer.get() != NULL) && bool("Not initialized!"))
    return 1;

  #ifdef OPENMP
    #pragma omp parallel for
  #endif
    for (i=0; i<long(bytesCopy); i++)
      dstBuffer.set()[sizeT(i)+dstOffset] = srcBuffer.get()[sizeT(i)
   +srcOffset];
  return 0;
}
```

Finalmente, la polibiblioteca se describe como una interfaz capaz de llamar a un método específico previamente optimizado con mecanismos basados en `CUDA` u `OpenMP`. Cada mecanismo fue clasificado con base en dos opciones principales: (1) `cuda` y (2) `openmp`. Considere el código 3.6 que incluye a la matriz A, los vectores x, y y que describe el producto matriz-vector.

Código 3.6: Abstracción de identificadores de memoria a través del método `memoryHandle`. Se realiza la declaración de una matriz A y un vector B y se omite su inicialización con datos. Se describe el producto de la matriz A y el vector x a través de la función `prod`.

```
matrix<double> A(15,15);
vector<double> x(15);

/*  Initialize A and x with data*/
y = prod(A,x);
```

Es importante mencionar que con el uso de `templates` se evita la generación de un vector temporal y la computación, se reduce a una instancia del método `prodMB(A,x,y)` descrito en la interfaz de la polibiblioteca. De esta forma, el método inspecciona los operandos y llama a las operaciones necesarias desde la opción respectiva (`cuda` u `openmp`) tal como se describe en el código 3.7. Cuando `A.handle()` devuelve el tipo de objeto correspondiente, se solicita un identificador para el dominio de memoria actualmente activo. Enseguida se instancia la implementación del método en la opción específica desde la interfaz de la polibiblioteca asociada.

Código 3.7: `prodMB` define un método utilizado para que la biblioteca principal instancie una llamada a la biblioteca específica definida por `CUDA` o `OpenMP`.

```
void prodMB(matrix<T> const &A, vector<T>& x, vector<T> &y){
```


```
  switch(A.handle().getActiveHandleId()){
    case MAIN_MEMORY:
      openmp::prodMP(A,x,y);
      break;
    case CUDA_MEMORY:
      cuda::prodMP(A,x,y);
      break;
    default:
      errorHandling();
  }
}
```

La implementación de cada uno de los métodos descritos en las dos opciones son diferentes entre si y su nombre corresponde al mecanismo de paralelización utilizado. Cualquier operación nueva, primero debe ser implementada para la biblioteca `openmp` bajo el supuesto de un gran número de subprocesos (grano fino a grano grueso). Esto requiere que se utilicen métodos similares a los implementados para `CUDA`. La implementación se prueba a través de las herramientas disponibles para depurar código `OpenMP`. Cuando se completa la implementación del método en la biblioteca `openmp`, el nuevo método es codificado en el enfoque de `CUDA`. Esto se logra copiando y alternando el código para que `openmp` considere las especificaciones de `cuda`. Las diferencias significativas solo surgen si se utilizan operaciones de reducción en ambientes de memoria compartida. Enseguida, la implementación de `cuda` se prueba con el conjunto de experimentos, independientemente de la biblioteca ya existente. Debido a que hay una implementación de referencia disponible en `openmp` y debido a que `CUDA` proporciona un amplio conjunto de herramientas de depuración, los errores se pueden localizar y corregir rápidamente.

La biblioteca transforma `templates` en código eficiente permitiendo manipular vectores y matrices completas, así como subvectores y submatrices descritos por desplazamientos de índices y pasos no unitarios tales cómo,

- Operaciones vectoriales sin reducción, la cual involucra operaciones vectoriales elemento a elemento. Por ejemplo, suma de vectores, resta de vectores, y multiplicación de elemento a elemento
- Operaciones vectoriales con reducción, la cual describe operaciones vectoriales elemento a elemento seguidas por la operación de reducción sobre todo el vector. Por ejemplo, el producto interno
- Operaciones matriciales sin reducción, la cual extiende las operaciones vectoriales sin reducción a operaciones matriciales
- Operaciones matriciales con reducción de filas o columnas. Por ejemplo, producto matriz-vector o el cálculo de la norma de una fila
- Producto de matrices

Este resulta ser el enfoque más robusto debido a que evita posibles errores al interactuar de otra manera con el entorno del sistema.

Para probar el rendimiento de la biblioteca, se requieren diferentes arquitecturas de prueba físicas que consten de dispositivos GPU de gama baja a alta de diferentes generaciones de hardware. Si se consideran tres sistemas operativos principales (Windows, Linux, MacOS), y dos proveedores de GPU con al menos tres arquitecturas de hardware diferentes cada uno, se requieren 18 configuraciones diferentes para una cobertura de prueba completa. La complejidad combinatoria incrementa si también se toman en cuenta diferentes versiones de sistemas operativos o compiladores. Incluso después de un esfuerzo considerable, el conjunto de arquitecturas de prueba cubrirá solamente la mitad de las configuraciones posibles. Por lo tanto, es vital la propuesta de técnicas que estimen el tiempo de ejecución.

3.5.2. Análisis de rendimiento de la biblioteca

Se describe el rendimiento de la biblioteca propuesta considerando el producto matriz-vector (PSpMV) y el producto matriz-matriz (PSpMM). El conjunto de matrices dispersas utilizadas para medir el rendimiento de la biblioteca se describe en la tabla 3.4, siendo este conjunto muy similar a los conjuntos de matrices utilizados para evaluar bibliotecas propuestas en el estado del arte. Todas las pruebas fueron ejecutadas en arquitecturas Linux equipadas con hardware AMD, INTEL y NVIDIA (utilizando CUDA SDK 10.2) como se enumera en la tabla 3.5.

Como punto de referencia para el producto de una matriz dispersa con un vector, también se realiza una comparación con dos GPU NVIDIA adicionales (GeForce GTX 470, GeForce GTX 750 Ti) para evaluar la portabilidad del rendimiento. Dado que las cuatro plataformas de hardware son comparables en su potencia de diseño térmico (TDP), los resultados de referencia permiten una comparación justa también en términos de rendimiento por vatio [9].

El producto de matrices dispersas (PSpMV) es una operación elemental en la mayoría de los algoritmos. Esta es la principal motivación por la que en el estado del arte se han centrado en el análisis del PSpMV para explotar el paralelismo disponible. La biblioteca propuesta proporciona implementaciones optimizadas de PSpMV para varios formatos de matrices (incluyendo matrices dispersas) que reducen la resistencia al uso de formatos específicos para almacenar una matriz dispersa. Se implementó el algoritmo PSpMV. Las evaluaciones obtenidas han demostrado que la adaptación también es eficiente para matrices muy dispersas (menos de cinco números distintos de cero por fila en promedio) en las GPU NVIDIA. Si el número medio de no ceros por fila supera los cinco, se asignan 8, 16 o 32 subprocesos por fila en las GPU NVIDIA.

En la figura 3-14 y 3-15 se describe una comparación del rendimiento de la rutina PSpMV de la biblioteca con las rutinas PSpMV en CUSP y MAGMA respectivamente. El experimento se ha realizado para comparar el rendimiento de la propuesta en arquitecturas equipadas con una GPU NVIDIA Geforce GTX 750 Ti (arquitectura Maxwell), una tarjeta NVIDIA Tesla K20m (arquitectura Kepler) y una tarjeta NVIDIA GeForce

[9]En las Ciencias Computacionales, el rendimiento por vatio es una medida de la eficiencia energética de una arquitectura computacional. En particular, el rendimiento por vatio se encarga se cuantificar la tasa de cálculo que puede entregar una computadora por cada vatio de energía consumido

Tabla 3.4: Propiedades de las matrices propuestas por *The Florida Sparse Matrix Collection* utilizadas para el análisis de rendimiento [85].

Identificador	Dimensión	NNZ	máx NNZ/row	*avg* NNZ/row
Cantilever	62 451	4 007 383	78	64.2
Economics	206 500	1 273 389	44	6.2
Epidemiology	525 825	2 100 225	4	4.0
Harbor	46 835	2 374 001	145	50.7
Protein	36 417	4 344 765	204	119.3
qcd	49 152	1 916 928	39	39.0
Ship	140 874	7 813 404	102	55.5
Spheres	83 334	6 010 480	81	72.1
Windtunnel	217 918	11 634 424	180	53.4
Accelerator	121 191	2 624 331	81	21.7
Amazon0312	400 727	3 200 440	10	8.0
ca-CondMat	23 133	186 936	280	8.1
cit-Patents	3 774 768	16 518 948	770	4.4
circuit	170 998	958 936	353	5.6
email-Enron	36 692	367 662	1 383	10.0
p2p-Gnutella31	62 586	147 892	78	2.4
roadNet-CA	1 971 281	5 533 214	12	2.8
webbase1m	1 000 005	3 105 536	4700	3.1
web-Google	916 428	5 105 039	456	5.6
wiki-Vote	8 297	103 689	893	12.5

NNZ = No ceros, máx = máximo, *avg* = amedia

Tabla 3.5: Descripción general del hardware utilizado en la comparación del rendimiento computacional. Todos los valores son picos teóricos. Los picos prácticos para el ancho de banda de la memoria y las operaciones de punto flotante por segundo son alrededor del 70 % al 80 % de los picos teóricos, excepto para `Xeon Phi`, donde solo se puede obtener el 50 % del ancho de banda máximo teórico.

	FirePro W9100	Dual Xeon E5-2670 v3	Xeon Phi 7120P	Tesla K20m
Característica	AMD	INTEL	INTEL	NVIDIA
Memoria (GB/seg)	320	136	352	208
$GFLOP/seg$ (float)	5 238	1 766	2 416	4 106
$GFLOP/seg$ (double)	2 619	884	1 208	1 173
TDP (watt)	275	240	300	244

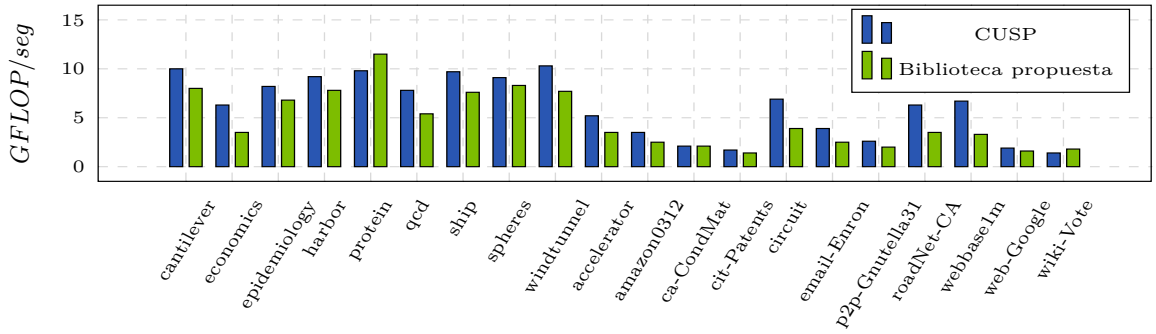
GTX 470 (arquitectura Fermi). Estos dispositivos representan tres tipos de arquitecturas de NVIDIA y han sido seleccionados para evaluar su rendimiento con base en su portabilidad. Se aprecia que la biblioteca propuesta supera a CUSP en la tarjeta GTX 750 Ti en un 36 % y en la tarjeta Tesla K20m en un 24 % en promedio con base en $GFLOP/seg$. Por otro lado, en la tarjeta GTX 470, el rendimiento de la biblioteca propuesta es un 13 % más bajo que el de CUSP. Se observa un mejor rendimiento de la biblioteca sobre CUSP del 31 %, 32 % y 31 % para las tres GPU, respectivamente. En general, la biblioteca propuesta proporciona un rendimiento general más alto que CUSP debido a las optimizaciones de rendimiento propuestas inicialmente para las GPU de AMD.

Para poder medir el rendimiento de la biblioteca en el producto de matrices utilizando matrices dispersas (PSpMM), se ha utilizado el algoritmo RMerge, el cual se encarga de calcular cada una de las filas de resultados para el producto $C = B$ fusionando simultáneamente varias filas de B cuyas entradas son distintas de 0. Se extiende la idea de fusionar varias filas a GPU generalizando la idea de *subwarps* en GPU NVIDIA a *subwavefronts* en GPU AMD, y a CPU, así como en Xeon Phi con la fusión de varias filas en una.

La comparación de rendimiento se describe en la figura 3-16 y 3-17 para el producto $C = AA$. Los resultados muestran que la biblioteca propuesta supera a CUSP en un 25 %. Se obtiene una ganancia de rendimiento del 30 % en promedio con una CPU. La situación se invierte en el caso de Xeon Phi, donde la biblioteca es en promedio un 65 % más lenta, lo que puede explicarse por un menor grado de optimización para Xeon Phi en la biblioteca. Cabe mencionar que no se cuenta con otra implementación disponible para poder realizar una comparación justa entre las unidades GPU. Una comparación de los promedios en términos de $GFLOP/seg$ de la implementación de la biblioteca en la tarjeta NVIDIA Tesla K20m con la implementación de la biblioteca en AMD FirePro W9100 muestra un 53 % más de rendimiento en la GPU de AMD. Si bien los resultados para PSpMV en el experimento anterior mostraron un mejor rendimiento en las GPU de NVIDIA al aprovechar las optimizaciones para las GPU de AMD, los resultados para los productos de matrices dispersas en esta sección extienden las optimizaciones para las GPU de NVIDIA a las GPU de AMD y también a las CPU.

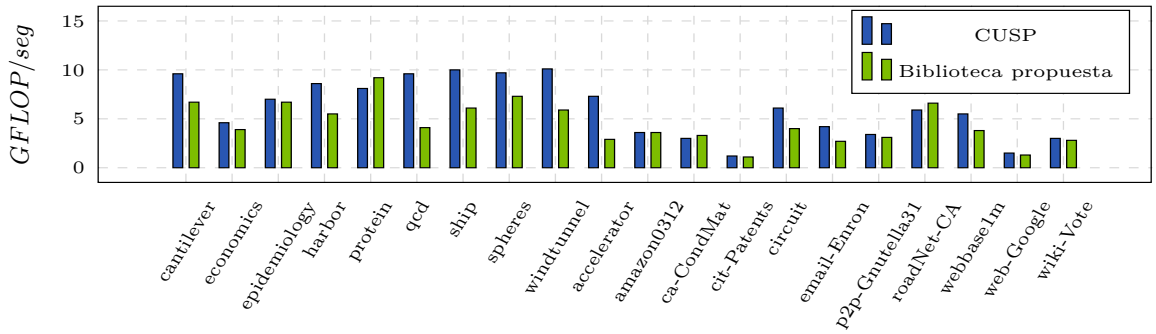
En general, los resultados de referencia sugieren que cuando la memoria caché de una CPU es grande se obtiene como resultado un rendimiento de producto de matrices dispersas significativamente mejor de las CPU en comparación con las GPU. Lo anterior se debe a que las filas a las que se accede repetidamente en B pueden ya estar disponibles en la caché. Sin embargo, esto también implica que el patrón de dispersión de la matriz A sea especialmente importante para la memoria caché. Las arquitecturas orientadas al rendimiento de las GPU y Xeon Phi tienen memoria caché mucho más pequeña, lo que resulta en muchas recargas ineficientes de datos de la memoria global.

Comparación de rendimiento de rutinas en la biblioteca con otras que usan una `GeForce GTX 470`.



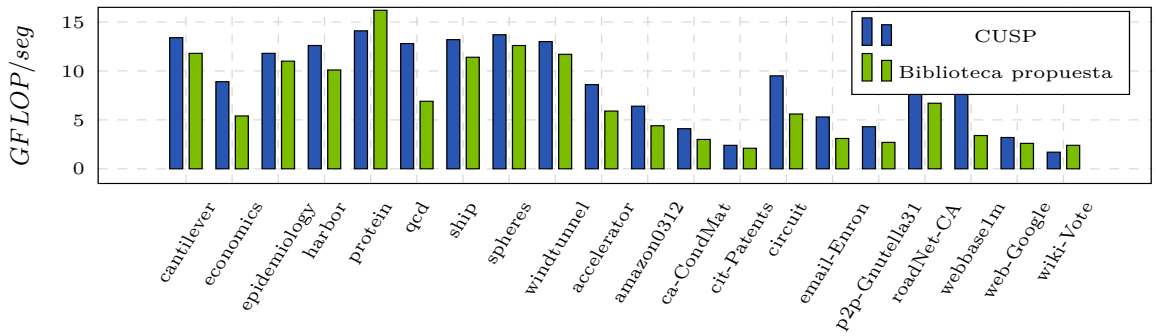
(a) GeForce GTX 470

Comparación de rendimiento de rutinas en la biblioteca con otras que usan una `GeForce GTX 750 Ti`.



(b) GeForce GTX 750 Ti

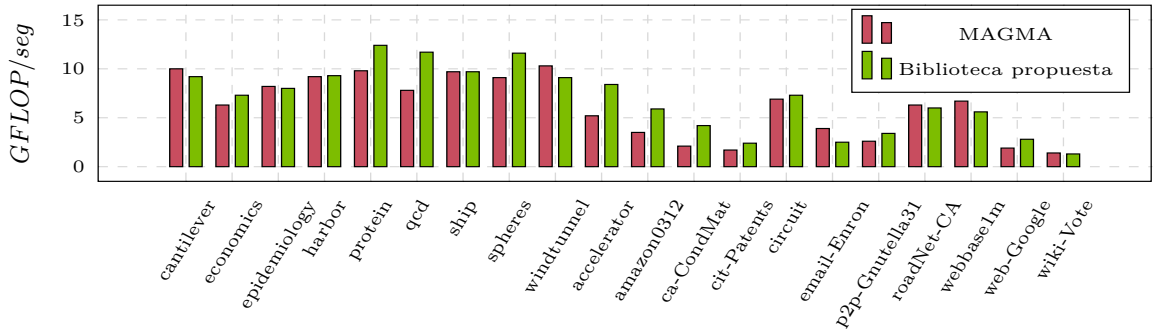
Comparación de rendimiento de rutinas en la biblioteca con otras que usan una `Tesla K20m`.



(c) Tesla K20m

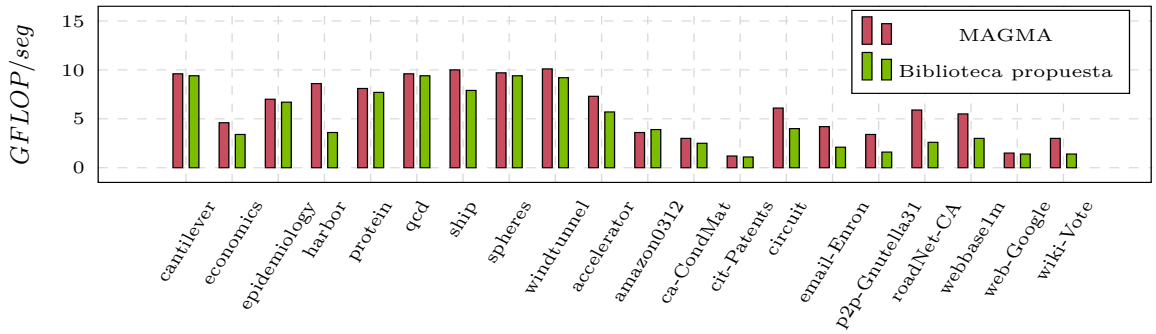
Figura 3-14: Comparación de rendimiento de `PSpMV` por la biblioteca propuesta y `CUSP v7.0` utilizando: `NVIDIA Geforce GTX 470` (arquitectura Fermi), `NVIDIA Tesla K20` (arquitectura Kepler), y `NVIDIA GTX 750 Ti` (arquitectura Maxwell) El promedio de todas las ejecuciones de prueba en los tres dispositivos son 5,9 $GFLOP/seg$ para la biblioteca propuesta, y $5,2$ $GFLOP/seg$ para `CUSP`.

Comparación de rendimiento de rutinas en la biblioteca con otras que usan una `GeForce GTX 470`.



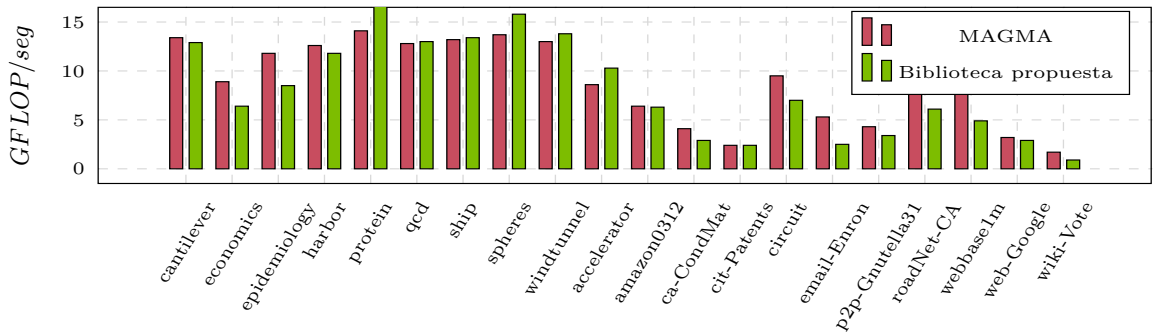
(a) GeForce GTX 470

Comparación de rendimiento de rutinas en la biblioteca con otras que usan una `GeForce GTX 750 Ti`.



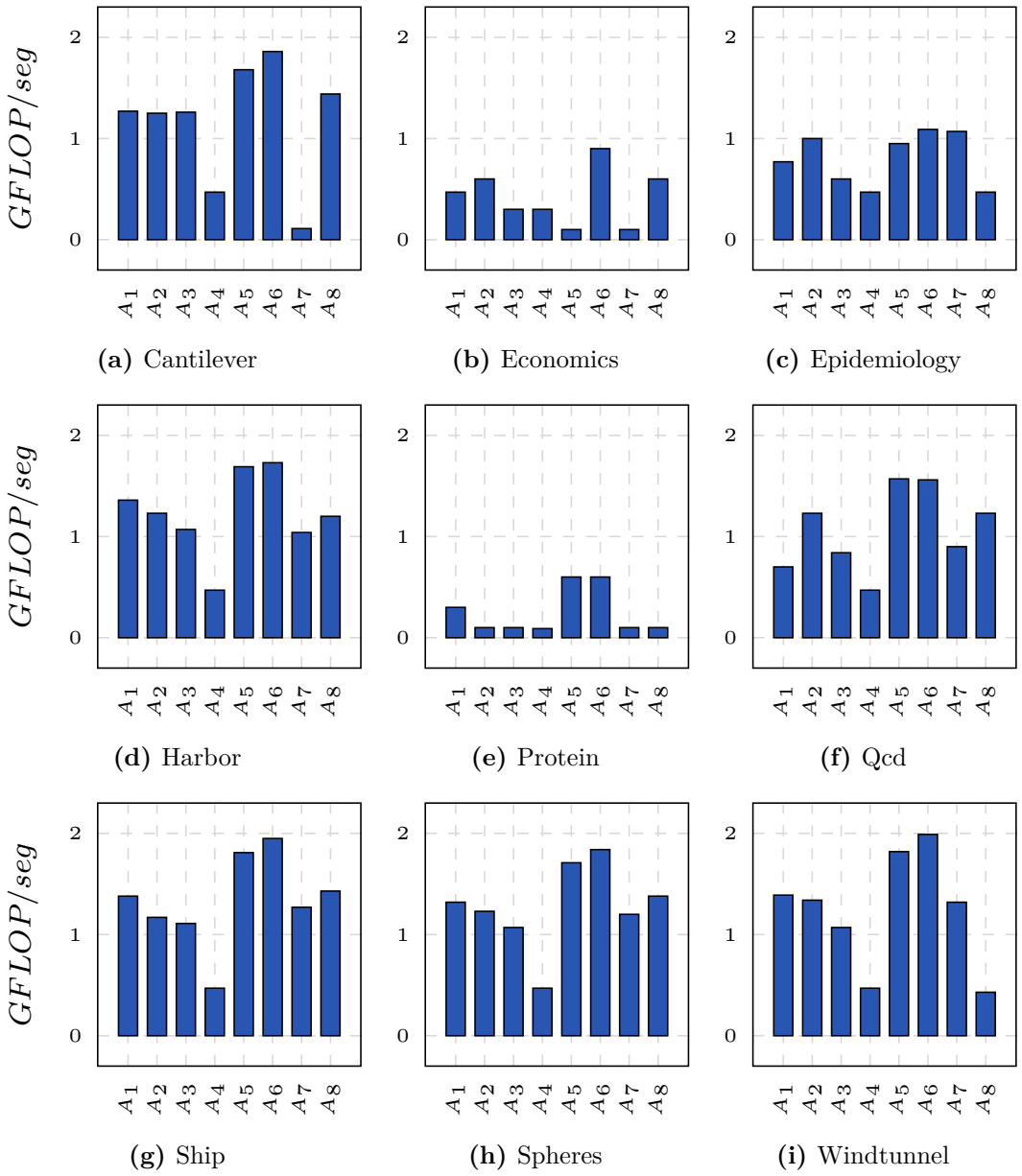
(b) GeForce GTX 750 Ti

Comparación de rendimiento de rutinas en la biblioteca con otras que usan una `Tesla K20m`.



(c) Tesla K20m

Figura 3-15: Comparación de rendimiento de `PSpMV` por la biblioteca propuesta y `MAGMA` utilizando: `NVIDIA Geforce GTX 470` (arquitectura Fermi), `NVIDIA Tesla K20` (arquitectura Kepler), y `NVIDIA GTX 750 Ti` (arquitectura Maxwell) El promedio de todas las ejecuciones de prueba en los tres dispositivos son 5,9 $GFLOP/seg$ para la biblioteca propuesta, y 4,5 $GFLOP/seg$ para `CUSP`.

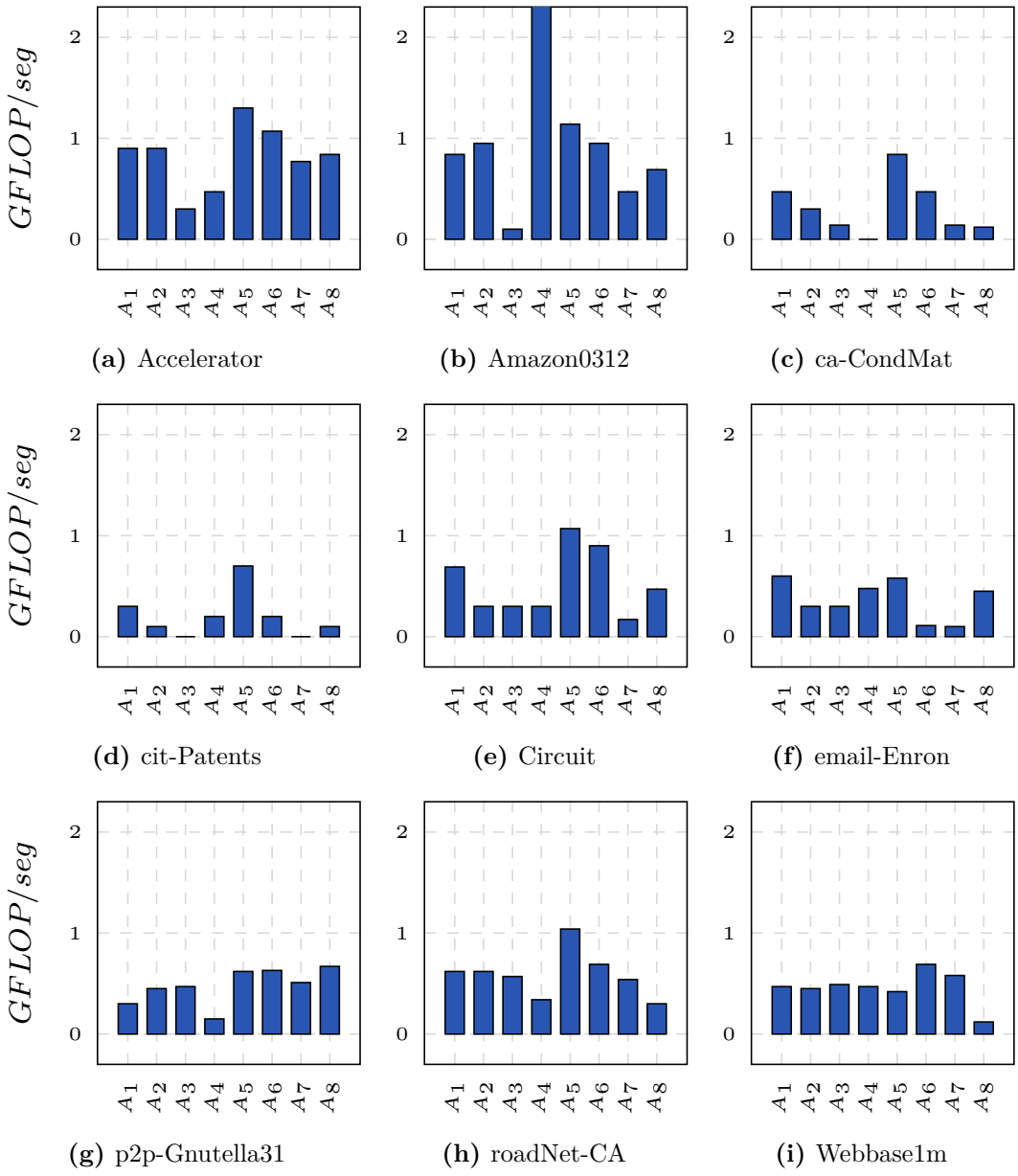


(a) Cantilever **(b)** Economics **(c)** Epidemiology

(d) Harbor **(e)** Protein **(f)** Qcd

(g) Ship **(h)** Spheres **(i)** Windtunnel

Figura 3-16: Comparación de rendimiento de rutinas dispersas de multiplicación matriz-matriz utilizando la biblioteca propuesta, `CUSP` y `MAGMA` para matrices utilizando el hardware descrito en la tabla 3.4. El mejor rendimiento general se obtiene en `CPU`. La biblioteca propuesta superó en un 30 % a `MAGMA`. Finalmente, la biblioteca propuesta supera a `CUSP` en un 25 %.



(a) Accelerator **(b)** Amazon0312 **(c)** ca-CondMat

(d) cit-Patents **(e)** Circuit **(f)** email-Enron

(g) p2p-Gnutella31 **(h)** roadNet-CA **(i)** Webbase1m

Figura 3-17: Comparación de rendimiento de rutinas dispersas de multiplicación matriz-matriz utilizando la biblioteca propuesta, `CUSP` y `MAGMA` para matrices utilizando el hardware descrito en la tabla 3.4. El mejor rendimiento general se obtiene en `CPU`. La biblioteca propuesta superó en un 30 % a `MAGMA`. Finalmente, la biblioteca propuesta supera a `CUSP` en un 25 %.

Esta página se dejó en blanco intencionalmente.

Capítulo 4
Auto-adaptación en técnicas paralelas de algoritmos

Los resultados descritos en la página 52 del capítulo 3 hacen evidente la importancia del procesamiento paralelo en las ciencias Biológicas. La razón es el aumento de la potencia de las unidades de procesamiento y la presencia de problemas donde el tiempo de resolución es inaceptable. En este capítulo se describe el contexto de las técnicas de auto-adaptación paralela descritas como segundo enfoque de solución al problema en la sección 1.1. Se enuncian los componentes de la metodología propuesta y se formaliza el modelo matemático para optimizar el tiempo de ejecución de un algoritmo en una arquitectura determinada considerando el tamaño de bloque que cada unidad de procesamiento debe resolver. Cabe mencionar que la metodología es válida tanto en arquitecturas homogéneas como arquitecturas heterogéneas.

4.1. Contexto de la investigación

El paralelismo permite utilizar múltiples unidades de procesamiento para la resolución de un problema, proponiendo así, una solución a las restricciones impuestas por las computadoras con solo una unidad de procesamiento [166]. El paralelismo ofrece soluciones más rápidas mediante algoritmos capaces de resolver problemas grandes y complejos, cuyos datos de entrada y operaciones computacionales exceden la capacidad de memoria de una computadora con características mínimas [159]. Sin embargo, la correcta implementación del HPC es todavía un reto. Lo anterior se debe a la curva de aprendizaje de los científicos en programación paralela y su esfuerzo intensivo con nula experiencia en el campo de la Bioinformática.

No existe un sistema paralelo, compuesto por el algoritmo paralelo y la arquitectura computacional que haya logrado imponerse de manera definitiva para la resolución de un problema. Las múltiples características que los conforman y su rango de variabilidad confieren al HPC un nivel de dificultad adicional. Lo ideal en este caso son sistemas cuyos «algoritmos paralelos» logren adaptarse de forma automática a una arquitectura para la cual no fueron diseñados.

La comunidad científica está haciendo un esfuerzo considerable para estudiar y comprender la auto-adaptación de «esquemas algorítmicos paralelos» [153], los cua-

les son el enfoque principal de este trabajo doctoral. Se hace hincapié en el diseño de herramientas de propósito general que puedan ser incluidas en lenguajes paralelos de alto nivel [169]. En lo que respecta al estado del arte del diseño de esquemas algorítmicos se hace mención de los mecanismos que evalúan cuantitativamente el tiempo de ejecución. Estos mecanismos están agrupados en: lenguajes de programación [121, 170], técnicas *pipeline* [4, 18, 79, 154], técnicas maestro-esclavo [5, 13] y esqueletos [43, 171, 179, 192].

En programación, los esqueletos se basan en el reemplazo de la programación paralela explícita, usando un lenguaje paralelo, por la instanciación de una variedad de formas algorítmicas paralelas prediseñadas. Por ejemplo, el enfoque que utilizan los sistemas operativos descrito en [43, 44], el cual además sigue el principio de *Backups* [10]. Este enfoque propone que la clave para una programación efectiva es la disponibilidad de una biblioteca de «operadores elementales» que permitan la creación de nuevas funciones a partir de otros operadores ya predefinidos. La biblioteca de operaciones elementales descrita en la sección 3.5 es un claro ejemplo de este tipo de paralelización.

Este trabajo se centra en mecanismos basados en «esqueletos» debido a que los algoritmos de MD, enfocados en tareas de clasificación y agrupación, comparten un conjunto común de patrones de interacción (ver página 28 de la sección 3.2). La metodología propuesta se divide en tres componentes principales: (1) Diseño de esqueletos, (2) Diseño del modelo de rendimiento y (3) Transformación del algoritmo. En este trabajo, el diseño de los esqueletos se ha desarrollado utilizando funciones polimórficas de orden superior en un lenguaje de programación funcional no estricto. Para este proceso se identificaron los procesos o tareas [1] genéricos que describen a los algoritmos de MD. De esta manera, la transformación del código del algoritmo supone la inserción de código específico dentro del esqueleto.

El estudio de parámetros óptimos que minimicen el tiempo de ejecución de un algoritmo que ha sido construido a partir de mecanismos definidos por esqueletos ya ha sido abordado en [35, 48, 75, 80, 231, 232]. Estos trabajos proponen el análisis del tiempo de ejecución con base en dos enfoques: (1) Pruebas de ejecución durante la instalación del algoritmo y (2) Modelado del tiempo de ejecución. En el primer enfoque, se realizan múltiples ejecuciones del algoritmo durante su instalación, de tal forma que se ajuste el algoritmo a las características físicas de la nueva arquitectura mediante decisiones tomadas sobre la mejor ejecución [62, 72, 197, 224, 234]. En el segundo enfoque, se plantea la optimización mediante el estudio y construcción de un modelo matemático que refleje las características del sistema paralelo.

Para el diseño del modelo matemático propuesto en este trabajo, la metodología se basa en el segundo enfoque debido a que el primero carece de mecanismos de generalidad para el diseño de herramientas y metodologías de auto-adaptación. Sin embargo, es necesario mencionar que a pesar de que el segundo enfoque es lo suficientemente genérico, generalmente es criticado por la dificultad inherente que resulta de la propuesta de un modelo matemático y su posterior optimización.

Se requiere una metodología robusta que se enfoque tanto en arquitecturas homo-

[1] Un proceso o tarea se define como un bloque de programa secuencial, con flujo de control propio.

géneas como en arquitecturas heterogéneas. En una arquitectura homogénea, las unidades de procesamiento tienen las mismas características (velocidad, memoria, entre otras.) y la red de comunicación es idéntica en cada uno de los nodos computacionales. Por otro lado, en una arquitectura heterogénea, las unidades de procesamiento son distintas o la red de comunicación es asimétrica [2]. Sin embargo, las características que definen a una arquitectura se pueden relajar y considerar a una arquitectura como homogénea, aunque no lo sea totalmente.

En la siguiente sección se describen los distintos enfoques propuestos en el estado del arte para modelar matemáticamente el tiempo de ejecución.

4.2. Enfoques para modelar el tiempo de ejecución

Se han propuesto estudios que plantean el modelado de sistemas paralelos [29,110]. Algunos de estos modelos han sido puramente teóricos, mientras que otros han sido utilizados para intentar predecir el comportamiento de un algoritmo paralelo a través de estudios comparativos de los tiempos de ejecución teóricos. En esta sección, se describe un planteamiento que sirvió como referencia para la formalización de un modelo matemático que refleja el rendimiento computacional de un algoritmo. Enseguida, motivados por la importancia del costo de las comunicaciones en el tiempo total de ejecución, se describe una propuesta que ha centrado su estudio en el modelado de dicho costo.

4.2.1. Modelo BSP

El modelo *Bulk-Synchronous Parallel* definido por sus siglas como BSP, fue descrito por primera vez en [82,83]. Este modelo fue propuesto como una interface estándar entre la arquitectura y el algoritmo paralelo. La metodología de este trabajo demostró que el modelo BSP es capaz de estimar el tiempo de ejecución asintótico y predecir tendencias de desempeño de un algoritmo independientemente de la arquitectura.

Este modelo se describe como un sistema paralelo compuesto por p unidades de procesamiento con memoria local, una red de interconexión con un ancho de banda limitado y un sistema de barreras con costo fijo. Se supone que la ejecución del programa se realiza en una serie de etapas denominadas *super pasos*. La computación realizada en cada *superstep* se realiza de manera local, de tal forma que los procesadores trabajan de manera independiente con datos almacenados en memoria local. Las comunicaciones en cada *superstep* se caracterizan por el envío de como máximo h mensajes desde una unidad de procesamiento hacia h procesadores destino. Cada mensaje estará disponible en el siguiente *superstep*.

El número de unidades de procesamiento (p), el costo por cada comunicación (t_{com}) y el tiempo mínimo entre sincronizaciones impuesto por la arquitectura (t_{sinc}), son parámetros que definen el costo de este modelo. Este último parámetro define la longitud mínima de un *superstep*.

[2]Las unidades de procesamiento tienen distintas velocidades de transmisión de datos entre pares de procesadores distintos.

Sea t_{comp} el tiempo dedicado a la computación de un algoritmo específico y sea t_{Spaso} el costo total de un *superstep*. Se definen dos enfoques:

- En cada *superstep* se definen dos etapas independientes, una de cómputo y otra de comunicación. Entonces $t_{Spaso} = \text{máx}(t_{comp} + ht_{com}, t_{sinc})$
- Los mensajes se envían en cualquier momento del *superstep*, con intersección entre la etapa de cómputo y la de comunicación. Es decir, $t_{Spaso} = \text{máx}(t_{comp}, ht_{com}, t_{sinc})$

Sin embargo, esta metodología carece de precisión para calcular el tiempo real de ejecución de un algoritmo, debido a que no se consideran los retrasos originados por la saturación de la red de interconexión cuando los procesadores están enviando mensajes al mismo tiempo. Finalmente, introduce un alto costo de sincronización al definir una barrera global al final de cada *superstep*.

Variantes a este modelo han sido propuestas en la literatura ([55, 65, 78, 105, 111, 146]) en las cuales, se han considerado los patrones de comunicación desbalanceados, es decir, aquellos en los que las unidades de procesamiento envían y reciben diferentes cantidades de datos. Se han introducido diferentes niveles de memoria jerárquica. Finalmente, se ha permitido que en un momento dado diferentes unidades de procesamiento puedan estar en distintos *super pasos*, pero sin modificar el número máximo de *super pasos* que son ejecutados por cada unidad de procesamiento, así como el costo de las operaciones de comunicación, las cuales se consideran como terminadas cuando la unidad de procesamiento receptor finaliza el *superstep* en el que el remitente se encontraba cuando envió el mensaje.

4.2.2. Modelo LogP

El modelo LogP se describió en [49] como una metodología que modela el comportamiento de sistemas paralelos. Su planteamiento es similar al modelo descrito en la sección 4.2.1, es decir, se integra de un conjunto de procesadores con memoria local unidos a través de una red de comunicación. Sin embargo, a diferencia del modelo BSP, no existe un mecanismo de sincronización, excepto el que conlleva implícitamente cada instrucción de comunicación incluido en el algoritmo paralelo. No se define un número máximo de mensajes a enviar (parámetro h), sino que el límite lo define la velocidad de los puertos de la red de interconexión [118].

El modelo matemático en este caso considera los siguientes parámetros: Latencia (t_{lat}), la cual se define como la cota superior del tiempo de comunicación de un mensaje pequeño definido por una palabra; *sobrecosto* (t_{over}) que representa el tiempo que requiere una unidad de procesamiento para iniciar el envío o recepción de un mensaje. Cabe mencionar que, durante este proceso, la unidad de procesamiento no es capaz de realizar alguna otra tarea; *Gap* (t_{Gap}), el cual representa el periodo de tiempo que existe entre la recepción y/o transmisión de dos mensajes consecutivos en una unidad de procesamiento; y p que define el número de unidades de procesamiento.

La capacidad de la red de interconexión está limitada por estos parámetros, de manera que, a lo sumo, $\lceil t_{lat}/t_{Gap} \rceil$ mensajes pueden encontrarse en tránsito desde una unidad de procesamiento a cualquier otra en un momento determinado.

Sea t_{com} el costo total de la comunicación de un mensaje corto entre dos unidades de procesamiento, el cual se define en la ecuación 4.1.

$$t_{com} = 2t_{over} + t_{lat}. \tag{4.1}$$

Entonces, el costo de comunicar n mensajes cortos de manera consecutiva entre dos unidades de procesamiento se define en la ecuación 4.2.

$$t_{com} = 2t_{over} + t_{lat} + (n-1)\,\text{máx}(t_{over}, t_{Gap}). \tag{4.2}$$

El modelo LogGP descrito en [3,102] es una variante al modelo LogP. Este modelo considera los mensajes largos de manera diferenciada. Se puede decir que el parámetro t_{Gap} representa el *Gap* que existe entre los *bytes* consecutivos de un mensaje largo. Este parámetro cuantifica el ancho de banda requerido para enviar este tipo de mensajes. De esta manera, el costo de comunicación de un mensaje formado por k bytes entre dos unidades de procesamiento se define en la ecuación 4.3.

$$t_{com}(k) = 2t_{over} + t_{lat} + kt_{Gap}. \tag{4.3}$$

Otras variantes han sido propuestas en las que se tratan mensajes de longitud variable y comunicaciones colectivas. Los parámetros t_{Gap} y t_{over} son considerados como funciones que dependen del tamaño de los mensajes que se envían [117, 196]. Finalmente, en [102] se dispone al *sobrecosto* como un parámetro variable que simula la sincronización implícita entre procesadores, antes de la transmisión de mensajes largos, en paradigmas basados en mecanismos de paso de mensajes (`MPI`).

4.2.3. Modelo de comunicación clásico

En los sistemas paralelos ha destacado el análisis del costo de comunicaciones entre unidades de procesamiento [70, 120, 143], especialmente en paradigmas basados en paso de mensajes. Por esta razón, en el estado del arte se han descrito trabajos que modelan el costo de las comunicaciones entre unidades de procesamiento durante la ejecución de un algoritmo paralelo en una arquitectura específica.

El modelo propuesto por Hockney en [95], representa una referencia para proponer una función que suponga el costo de las comunicaciones durante la ejecución de algoritmos paralelos en una arquitectura específica. Este modelo define al costo de comunicar un mensaje de n bytes entre dos unidades de procesamiento tal como lo describe la ecuación 4.4.

$$t_{com}(n) = \frac{n + n_{1/2}}{r_\infty}, \tag{4.4}$$

donde r_∞ representa el ancho de banda asintótico de una comunicación cuando el tamaño del mensaje tiende a infinito, y $n_{1/2}$ es el tamaño de mensaje que permite alcanzar la mitad de dicho valor asintótico. De esta forma, la ecuación 4.4 puede reescribirse como lo expresa la ecuación 4.5.

$$t_{com}(n) = t_{ini} + nt_{trans}, \tag{4.5}$$

donde t_{ini} representa el tiempo de inicio de una comunicación (*startup time*) y t_{trans} el tiempo de transmisión por byte (*byte send time*), considerando que:

$$t_{ini} = \frac{n_{1/2}}{r_\infty}, \qquad t_{trans} = \frac{1}{r_\infty}. \tag{4.6}$$

Se han propuesto diferentes enfoques, basados en la aplicación del modelo de Hockney, en los sistemas de comunicación actuales. Uno de estos métodos es el «modelo postal», propuesto en [14], el cual introduce un parámetro denominado «latencia». Este parámetro cuantifica el inverso de la relación entre el tiempo que requiere una unidad de procesamiento para enviar un mensaje y el tiempo que pasa hasta que el destinatario lo recibe. Sea $\mathcal{P}$ el conjunto de unidades de procesamiento de una arquitectura paralela con una latencia t_{lat}, medida en ciclos de reloj, tal que $|\mathcal{P}| = p$ y $t_{lat} \geq 1$. La latencia de comunicación de un modelo postal se define de la siguiente manera. Si en un instante de tiempo $t_i \in \mathbb{R}^+$, el procesador $p_1 \in \mathcal{P}$ envía un mensaje msg al procesador p_2, entonces el procesador p_1 permanece ocupado enviando el mensaje msg durante el intervalo de tiempo $[t_i, t_{i+1}]$, y el procesador p_2 esta ocupado recibiendo el mensaje msg durante el intervalo $[t_i + t_{lat}, t_{i+1} + t_{lat}]$.

Este modelo es relativo al modelo de Hockney si se considera que $t_{trans} = 1$ y $t_{lat} = t_{ini}$. Ocurre lo mismo para el modelo LogP, descrito en la sección 4.2.2, si se caracteriza t_{com} mediante los parámetros $t_{ini} = (t_{lat} + 2t_{over})$ y $t_{trans} = t_{Gap}$.

Finalmente, en [71] se refina el modelo con el objetivo de considerar las propiedades físicas de la red de interconexión. Se introdujo la competencia por el ancho de banda, disponible a partir de la modificación de la ecuación 4.5 (ver ecuación 4.7).

$$t_{com}(n) = t_{ini} + nt_{trans}p_s, \tag{4.7}$$

donde p_s representa el número de procesadores que necesitan enviar concurrentemente una cantidad predefinida de mensajes sobre el mismo canal de comunicaciones tal que, p_s refleja la idea de que el ancho de banda efectivo disponible para cada procesador es $\frac{1}{p_s}$ del ancho de banda real.

A pesar de que la ecuación 4.7 no considera el costo de contención adicional que puede incurrir si los mensajes colisionan y deben ser retransmitivos, la experiencia muestra que es lo suficientemente precisa para propósitos prácticos. Finalmente, el valor de p_s depende de las propiedades del algoritmo paralelo y de la red de interconexión.

4.2.4. Modelo DLAM

En 1996 se propuso un modelo teórico para sistemas paralelos denominado DLAM (por sus siglas en inglés, *Distributed Linear Algebra Machine*). En este modelo se especifican las características de la arquitectura mediante parámetros que definen el tiempo de cómputo y de comunicación [51]. El tiempo de cómputo se define por el tiempo que requiere cada una de las operaciones descritas en la biblioteca `BLAS`, la cual contiene un conjunto de rutinas de bajo nivel cuyo objetivo es optimizar operaciones

comunes de Álgebra Lineal [19]. Por otro lado, el costo de comunicación se define a través de una red de intercomunicación de tipo `BLACS` [3], la cual constituye una biblioteca de paso de mensajes enfocada en operaciones de Álgebra Lineal [59]. De esta forma, los datos se operan mediante rutinas `BLAS` y son intercambiados entre las unidades de procesamiento mediante primitivas `BLACS`.

A pesar de que el costo de cada una de las operaciones elementales que pertenecen al i-ésimo nivel de `BLAS` es $\delta_i \in \mathbb{R}^+,\ i = 1, 2, 3$, en este modelo se asume que el costo de cualquier operación es constante.

Sean dos unidades de procesamiento cualesquiera, p_s el número de procesadores necesarios para enviar mensajes concurrentes, t_{inicio} el tiempo necesario para iniciar una comunicación y $t_{envío}$ el tiempo para enviar una palabra. El modelo clásico descrito en la ecuación 4.5 se puede describir como

$$t_{com}(n) = (t_{star} + nt_{envío})p_s. \tag{4.8}$$

El factor p_s pondera el modelo clásico con base en la arquitectura y el canal de comunicación tal como se describe en la ecuación 4.7. Cabe mencionar que tanto el costo de cada una de las operaciones elementales clasificadas conforme a la jerarquía propuesta por Dongarra et al. en [60], como el tiempo de comunicación $(p_s, t_{inicio}, t_{envío})$ son considerados valores constantes propios de cada plataforma. Sus valores se obtienen experimentalmente mediante diferentes ejecuciones y tamaños de datos.

4.3. Formalización del modelo matemático

Con base en los estudios descritos en el estado del arte, se debe considerar que la propuesta de modelos matemáticos con alto grado de abstracción permitirá entender el comportamiento del algoritmo con mayor independencia de la arquitectura destino. Por otro lado, conforme se detalle más, el modelo será capaz de mejorar las características de la arquitectura, pero a costa de la pérdida de portabilidad. Por lo tanto, se requiere establecer un compromiso entre un modelo con la suficiente abstracción que permita la portabilidad entre distintas plataformas paralelas; pero que, a su vez, tenga suficiente grado de consistencia, para que la estimación del tiempo de ejecución sea lo suficientemente realista. Dicho modelo representa una herramienta útil para la auto-adaptación de algoritmos paralelos.

Es de vital importancia un modelo que refleje fielmente el costo de las comunicaciones, debido al gran peso que suelen tener en el costo total de ejecución del programa paralelo en las arquitecturas actuales. El modelo clásico descrito en la sección 4.2.3 representa una aproximación a la propuesta de este trabajo, sin embargo, es necesario considerar las características de los datos que se envían y reciben, principalmente en el esquema de almacenamiento. Es necesario enfatizar las comunicaciones colectivas, teniendo en cuenta su implementación en una arquitectura dada.

A partir del modelo DLAM, descrito en la sección 4.2.4, se describe un modelo matemático para estimar el tiempo de ejecución de un algoritmo. Sin embargo, no

[3] `BLACS`: *Basic Linear Algebra Communication Subprograms*

modela exactamente las características físicas de la arquitectura, sino, mediante un conjunto de parámetros, refleja los efectos que se obtienen en el tiempo de ejecución considerando las características del sistema paralelo.

El modelo propuesto consiste en $p \in \mathbb{N}$ unidades de procesamiento, cuyas características pueden ser similares cuando de una arquitectura homogénea se habla. En caso contrario, las características difieren y entonces se habla de una arquitectura heterogénea. Estas unidades de procesamiento se encuentran conectadas entre si y son caracterizadas por el *sobrecosto*, la latencia y el ancho de banda necesario para establecer una comunicación entre cualquier par de procesadores. Cuando se ejecuta una sección del algoritmo, la primera tarea es la selección de las p_s unidades de procesamiento a utilizar, es decir, $p_s \leq p$. La topología lógica de las p_s unidades de procesamiento será, en un principio, una malla bidimensional de p_{s_r} filas y p_{s_c} columnas, tal que $p_s = p_{s_r} \times p_{s_c}$, aunque se podría tomar cualquier otra topología sin modificar el modelo planteado.

En cada unidad de procesamiento, las funciones que completan el esqueleto que caracteriza a los algoritmos de MD son descritas mediante la biblioteca de operaciones elementales descrita en la sección 3.5. De igual forma, las comunicaciones entre dos unidades de procesamiento se realizan mediante cualquier biblioteca de paso de mensajes. Para efectos de este trabajo, se utilizó `MPI`.

Sea $\mathcal{S}$ un subsistema de cómputo, el conjunto de características que definen a la arquitectura destino y las bibliotecas de operaciones elementales que definen a los algoritmos de MD; y sea $\mathcal{C}$ un susbsistema de comunicaciones, el conjunto de características que definen a la arquitectura destino y las bibliotecas de comunicaciones para paso de mensajes; entonces, se propone un modelo compuesto por estos dos submodelos. En la sección 4.3.1 y 4.3.2 se describirán las propiedades de cada uno de ellos.

4.3.1. Subsistema de cómputo

El modelo de cómputo se basa en operaciones elementales de tipo `BLAS` previamente paralelizadas y optimizadas a través de la metodología descrita en la sección 3.5. Su uso se debe a que son de propósito general y comúnmente utilizadas en la codificación de algoritmos de MD. Cabe mencionar que la implementación particular de cada una de estas rutinas se ha hecho con el objetivo de conseguir una mayor aceleración en una arquitectura en particular, de tal forma que su uso conlleve un incremento sustancial del rendimiento.

Sea $\delta_i \in \mathbb{R}^+,\ i = 1, 2, 3$, el tiempo de ejecución de una operación de punto flotante en cada uno de los niveles definidos anteriormente, tal que $\delta_1 < \delta_2 < \delta_3$, la metodología `BLAS` considera que todas las operaciones elementales que pertenecen al i-ésimo nivel tienen el mismo costo computacional. A diferencia de la metodología `DLAM`, la cual considera un tiempo de ejecución constante para cualquier operación elemental, en ésta metodología cada parámetro δ_i no es constante. La motivación de este hecho es que cada patrón de acceso a los datos definido por las operaciones que pertenecen al i-ésimo nivel, no necesariamente es el mismo. Un cambio en el patrón de acceso a los datos provoca cambios en la localidad de los datos operados y, por tanto, un

tiempo de ejecución distinto. De este modo, cada operación del i-ésimo nivel tendrá una complejidad determinada por $\delta_{i_j},\ 1 \leq j \leq k_i$, donde k_i representa el total de operaciones elementales descritas en el i-ésimo nivel de la biblioteca.

4.3.2. Subsistema de comunicación

Para modelar el tiempo de comunicación que permite transferir n palabras entre dos unidades de procesamiento cualesquiera, se considera el modelo clásico descrito en la ecuación 4.8 de la sección 4.2.4 ponderado por un factor p_s, cuyo valor dependerá de la arquitectura y del tipo de comunicación que se utilice (ver ecuación 4.9).

$$t_{com}(n, t_{inicio}, t_{envío}) = (t_{inicio} + nt_{envío})p_s, \tag{4.9}$$

donde el parámetro definido por t_{inicio} representa el tiempo de inicio de una comunicación (*start up time*) y $t_{envío}$ define el tiempo necesario para enviar un mensaje de tamaño de una palabra (*word send*).

En este trabajo, los parámetros t_{inicio} y $t_{envío}$ dependen del tamaño de datos enviados y del esquema de almacenamiento que éstos tengan. No se restringe el esquema de comunicación a operaciones de tipo `BLACS`, sino que también se consideran operaciones determinadas por la biblioteca `MPI`.

Dada una operación de comunicación colectiva, el parámetro p_s caracteriza a la red de interconexión de la arquitectura cuando adquiere una topología lógica determinada para llevar a cabo una operación, incluyendo las colisiones que se generen al usarse. En comparación con la metodología impuesta por DLAM en una arquitectura de p procesadores unidos por una red local, el valor de p_s para una operación de *propagación*, implementada con un algoritmo de distribución en forma de árbol binario, es $p_{s_{broad}}(p) = \log_2(p)$, debido a que la red de interconexión tiene capacidad de comportarse como un hipercubo lógico. En esta propuesta no se determina un valor específico para $p_{s_{broad}}(p)$, ya que éste depende de la capacidad dinámica que tenga cada red específica para gestionar una operación de *propagación*. Por tanto, $p_{s_{broad}}(p) \in [\log_2(p), p]$, tal como lo muestran los resultados experimentales obtenidos. Puede ocurrir que p_s dependa del tamaño del mensaje enviado, por lo que es conveniente medir experimentalmente el valor de $p_{s_{broad}}(p, n)$ en cada sistema paralelo.

Debido a que este subsistema se compone por los parámetros t_{inicio}, $t_{envío}$, y p_s, para cada biblioteca de paso de mensajes, este modelo caracteriza la capacidad de comunicación mediante la unión de la arquitectura y el sistema de cómputo $\mathcal{S}$ que describe a los algoritmos de MD.

4.3.3. Modelo para algoritmos de MD

El modelo del sistema completo se conforma por la unión de los subsistemas de cómputo y de comunicación antes definidos. El conjunto de todos los parámetros del modelo caracteriza, por tanto, la capacidad de cómputo y de comunicaciones del sistema completo. Los valores de estos parámetros son calculados mediante experimentación sobre el sistema durante la instalación del algoritmo correspondiente.

En la propuesta de este trabajo, el tiempo de ejecución de un algoritmo se ha descrito por medio de la minimización de una función objetivo, la cual refleja las características del sistema paralelo. Por tanto, la función objetivo se define como

$$T_{ejecución} = arg\,\text{mín}\, f(s, \mathcal{C}, \mathcal{S}), \tag{4.10}$$

donde s es el tamaño del problema, $\mathcal{C}$ es el conjunto de parámetros que caracterizan a la arquitectura (incluyendo las capacidades de cómputo y la red de interconexión entre cada unidad de procesamiento) y $\mathcal{S}$ es el conjunto de parámetros algorítmicos, cuyo valor podrá ser escogido al momento de ejecutar el algoritmo paralelo.

El valor de cada parámetro del sistema no se considera constante. Por lo tanto, y para poder proponer una solución de la ecuación 4.11, se deben de considerar dos aspectos necesarios: (1) Parámetros algorítmicos definidos por el número de procesadores a utilizar de los disponibles en una arquitectura, el tamaño de bloque en el mapeo de procesos a procesadores considerando el tiempo de distribución, entre otros; y (2) Tamaño del problema.

$$\mathcal{C} = arg\,\text{mín}\, g(s, \mathcal{S}). \tag{4.11}$$

Gracias a la estructura jerárquica de la biblioteca de operaciones elementales que describen a los algoritmos de MD, el modelo del tiempo de ejecución de una rutina perteneciente a una biblioteca de alto nivel, se construye con base en los diferentes modelos de las rutinas a las que llama en su código, mientras que para las rutinas de los niveles inferiores se modela su tiempo de ejecución directamente a partir de los parámetros básicos del sistema.

En una arquitectura heterogénea, el número de parámetros algorítmicos se incrementa de manera importante, ya que cada unidad de procesamiento puede soportar diferentes tamaños de bloque, o bien, el número de procesos a asignar en cada procesador puede variar. En este tipo de arquitecturas el tiempo de ejecución de una operación elemental δ_{ij} varía en función del procesador considerado. Generalmente, es más sencillo representar en el modelo descrito en la ecuación 4.10 los parámetros algorítmicos que los parámetros del sistema, debido a que solo se calculan una vez.

Por tanto, en arquitecturas homogéneas, los componentes del sistema paralelo tienen la misma capacidad entre todos los elementos. En el caso de arquitecturas heterogéneas los componentes varían, por lo que se supone una arquitectura con p unidades de procesamiento en el que a cada parámetro aritmético le corresponde un vector de p componentes ($\delta_{i_j} = (\delta_{i1_j}, \ldots, \delta_{ip_j})$), donde cada componente representa el tiempo de ejecución de la j-ésima operación elemental en el p-ésimo procesador clasificada en el i-ésimo nivel de la biblioteca. Lo mismo ocurre para el esquema de comunicación entre procesadores, el cual es ordenado en un arreglo de dimensión $p \times p$: $t_{inicio} = (t_{inicio_{1,1}}, \ldots, t_{inicio_{1,p}}, \ldots, t_{inicio_{p,p}})$. y $t_{envío} = (t_{envío_{1,1}}, \ldots, t_{envío_{1,p}}, \ldots, t_{envío_{p,p}})$, donde $t_{inicio_{i,j}}$ y $t_{envío_{i,j}}$ representan el *start-up* y el *world-send*de un proceso en la i-ésima y j-ésima unidad de procesamiento. Cuando $i = j$ entre dos procesos en la misma unidad de procesamiento, los parámetros$t_{inicio_{i,j}} = 0$ y $t_{envío_{i,j}} = 0$ debido a que no se requieren procesos de comunicación para transferir datos de una unidad de procesa-

miento a otra.

En la adaptación de algoritmos homogéneos a arquitecturas heterogéneas, los parámetros algorítmicos consideran el número de procesos asignados a cada unidad de procesamiento, de modo que el modelo matemático es empleado para la «asignación o mapeo» [4]. Es decir, la distribución equitativa en el entorno homogéneo.

Una vez que un algoritmo ha sido modelado mediante la función objetivo descrita en la ecuación 4.10 se obtiene una aproximación al tiempo de ejecución del algoritmo sin necesidad de realizar su ejecución, la cual, puede ser muy costosa. Sea P_{arq_0} una arquitectura paralela, la optimización de un algoritmo sobre ella consiste en buscar los parámetros $P_{alg_{\text{mín}}}$ que hacen que la función $T_{ejecución}$ alcance su valor mínimo.

4.4. Metodología de implementación

Se describe la metodología general de trabajo, la cual se caracteriza por ser un marco de trabajo que permite la generación de mecanismos de auto-adaptación basados en metodologías de HPC que aseguren el uso eficiente de los recursos computacionales durante la implementación de algoritmos paralelos.

El esquema general de la metodología está organizado en tres fases principales: (1) Diseño, en la cual se diseña el algoritmo y se construye una función o modelo analítico que representa el costo temporal de ejecución y que, además, considera tanto los parámetros algorítmicos como los del sistema; (2) Instalación, en la cual se obtienen los parámetros del sistema determinados por las características de la arquitectura destino y se incluyen en la función objetivo descrita en la ecuación 4.10 con el objetivo de que refleje fielmente el comportamiento del algoritmo; (3) Ejecución, en la que se calculan los parámetros algorítmicos que minimicen la función objetivo. Una vez calculados, se procede a la ejecución del algoritmo (ver figura 1-1).

La dificultad surge en la construcción del modelo analítico en la fase de Diseño. Por lo que se ha propuesto una submetodología que permite obtener una solución de calidad. Esta metodología se compone de seis etapas (ver figura 4-1): (1) Estudio del algoritmo, en la cual se analiza el código que se pretende auto-adaptar. Para ello, se identifican las características computacionales del código con base en su complejidad y el tamaño del problema; (2) Construcción del modelo matemático que refleje fielmente las características del sistema paralelo; (3) Diseño de pruebas que permitan evaluar si dicho modelo se asemeja a la realidad. Estas pruebas se realizan a través de simulaciones con el objetivo de probar la bondad del modelo matemático; (4) Validación del modelo. En esta fase se comparan los resultados obtenidos con simulaciones y ejecuciones en sistemas reales. Cuando los resultados son similares, se concluye que el modelo matemático es fiable y congruente con la realidad. Sin embargo, si existen diferencias significativas, el modelo requiere ser rediseñado; (5) Uso del modelo en

[4] El problema de «asignación de procesos» o «mapeo de procesos» se describe como el ajuste de un algoritmo a los recursos de la arquitectura con el objetivo de optimizar su tiempo de ejecución. Este problema es considerado NP-completo para una arquitectura general con p unidades de procesamiento y por tanto, la tarea de encontrar una asignación de costo mínimo es computacionalmente intratable salvo para arquitecturas con un número relativamente pequeño de procesadores.

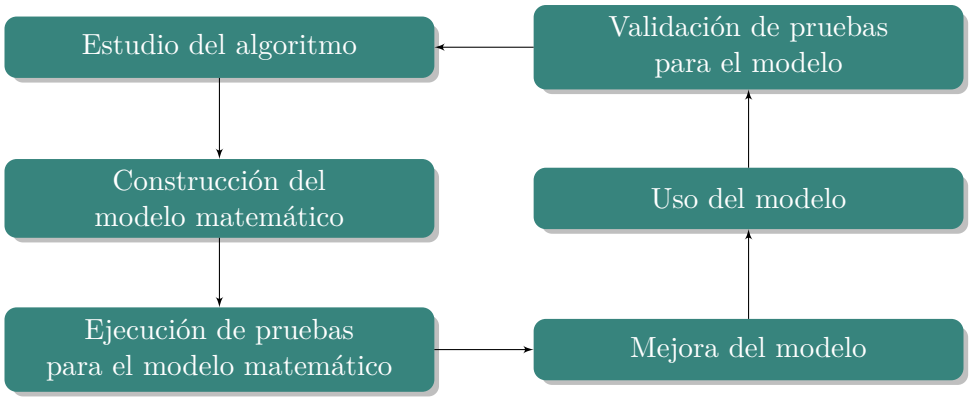


Figura 4-1: Metodología de la fase de diseño compuesta por seis etapas: (1) Estudio del algoritmo, en la cual se analiza el código que se pretende auto-adaptar; (2) Construcción del modelo matemático que refleje fielmente las características del sistema paralelo; (3) Diseño de pruebas que permitan evaluar si dicho modelo se asemeja a la realidad; (4) Validación del modelo. En esta fase se comparan los resultados obtenidos con simulaciones con ejecuciones en sistemas reales; (5) Uso del modelo en el que se dispone de una herramienta adecuada que permite que los algoritmos puedan ser auto-adaptables; y (6) Mejora del modelo, en la cual se busca reducir el tiempo empleado en la obtención de parámetros.

el que se dispone de una herramienta adecuada que permite que los algoritmos sean auto-adaptables; y (6)Mejora del modelo, en la cual se busca reducir el tiempo empleado en la obtención de parámetros que permiten obtener una buena solución del problema a resolver.

Capítulo 5
Auto-adaptación de esquemas paralelos de Minería de Datos

Siguiendo la metodología descrita en la sección 4.4, se describe la auto-adaptación del esquema algorítmico que utilizan las técnicas de MD enfocadas en tareas de clasificación y agrupación presentadas en el capítulo 3. De esta forma, el objetivo de esta sección es la implementación de heurísticas que resuelvan el modelo matemático que minimiza el elevado costo computacional en arquitecturas homogéneas y heterogéneas de manera automática. Se analizan los esquemas secuenciales que están basados en patrones iterativos, por ejemplo el algoritmo secuencial 1 que consiste en la ejecución repetitiva de un conjunto de instrucciones.

Se describe una aplicación para validar la metodología en el campo de la Bioinformática. En este sentido, se describe el análisis de datos transcriptómicos relacionados a la evolución de hepatocarcinomas [1] en ratones. Los datos han sido obtenidos *in vitro* [2], y a su vez, han sido preprocesados y normalizados con el objetivo de ordenarlos en una matriz de expresión genética. Finalmente, se añade una discusión de los resultados.

5.1. Definición de esquemas algorítmicos

Los esquemas algorítmicos se definen como patrones básicos que constituyen un esquema de referencia para la resolución de problemas reales. Su aplicación contribuye en la realización de un análisis más fino de los algoritmos y por tanto, a la definición de mecanismos de paralelización. Sin embargo, en la mayoría de los casos, la solución a un problema no se consigue con la aplicación de un esquema simple. Entonces se requiere de su modificación o la combinación de esquemas. Lo anterior se debe a que permiten determinar el conjunto de características que son comunes a una amplia variedad de algoritmos.

La propuesta tiene como objetivo completar funciones secuenciales con operaciones elementales en el algoritmo que se desea implementar con base en el esquema

[1] El hepatocarcinoma es un tumor hepático cuyo pronóstico está ligado a su detección temprana.

[2] *In vitro* se refiere a una técnica para realizar un experimento en un tubo de ensayo, o generalmente en un ambiente controlado fuera de un organismo vivo.

Algoritmo 3 Esquema iterativo secuencial

Entrada: Parámetros de entrada
Salida: Parámetros de salida

```
1: procedure SequentialIterativeSchema
2:     Conjunto de instrucciones
3:     while ConditionEvaluation(Parámetros) do
4:         Conjunto de instrucciones
5:         Actualizar parámetros y condiciones
6:     End while
7:     Conjunto de instrucciones
8: End procedure
```

paralelo al que corresponde el esqueleto. De esta forma, se obtiene una paralelización transparente al usuario. Se pueden integrar técnicas de auto-adaptación en los esqueletos con el objetivo de que sea la propia metodología quien obtenga los parámetros específicos que permitan una ejecución de un algoritmo con tiempo de ejecución reducido.

Este trabajo se enfoca en esquemas algorítmicos iterativos debido a que la ejecución de los algoritmos de agrupamiento descritos en la sección 3.3 se divide en un número de pasos que se repiten constantemente, y donde cada paso inicia una vez que el anterior ha terminado. Algunas variantes de este esquema básico se describen con base en el criterio de convergencia utilizado (ver sección 3.3.4). También puede existir variación en la computación de cada iteración, la cual puede ser constante o depender de la iteración que se encuentre en ejecución. A pesar de que el enfoque son algoritmos de agrupamiento concretos (KM, FCM y EM), la metodología trabaja de forma general a las variaciones del esquema básico.

En general, los algoritmos de agrupamiento se enfocan en la ejecución de iteraciones sucesivas en las que se realiza la misma computación y se detienen cuando un criterio de convergencia se cumple. Por lo tanto, siguen el mismo esquema iterativo básico descrito anteriormente. Cabe destacar que KM, FCM y EM son algoritmos de costo computacional alto, lo cual hace que sea interesante el desarrollo de algoritmos paralelos auto-adaptables a arquitecturas de alto rendimiento.

5.2. Esquemas iterativos

Un esquema iterativo se define por la ejecución sucesiva de un conjunto de instrucciones que se detienen hasta que se cumple un criterio de convergencia. El esquema que define a un algoritmo iterativo se describe en el pseudocódigo 3.

Sea $t_{Iterativo}$ el tiempo computacional del esquema iterativo, $t_{NoIterativo}$ el tiempo computacional de la sección del algoritmo que no pertenece al esquema iterativo, No_{iter} el número de veces que se ejecutará un conjunto de intrucciones (iteraciones),

$t_{comp}(iter)$ el tiempo de cómputo de una iteración y $t_{cond}(iter)$ el tiempo de actualización y evaluación del criterio de convergencia en cada iteración $iter$. El costo computacional de un algoritmo se puede representar por la ecuación 5.1.

$$T_{ejecución} = t_{Iterativo} + t_{NoIterativo} \tag{5.1}$$

donde

$$t_{Iterativo} = \sum_{iter=1}^{No_{iter}} (t_{comp}(iter) + t_{cond}(iter)) \tag{5.2}$$

Considere que si los costos en las distintas iteraciones coinciden, entonces

$$t_{Iterativo} = No_{iter} \cdot (t_{comp}(iter) + t_{cond}(iter)). \tag{5.3}$$

En algunos casos, el costo de la actualización y evaluación de la condición es despreciable respecto al costo de la computación. Por ejemplo, aquellos casos en los que no se realiza la evaluación debido a que existe un número fijo de iteraciones, o bien, el número de iteraciones en que se realiza la evaluación es relativamente pequeño. En estos casos no es necesario incluir el término t_{cond} y la ecuación 5.2 se reescribe tal como se expresa en las ecuaciones 5.4.

$$t_{Iterativo} = \sum_{iter=1}^{No_{iter}} (t_{comp}(iter)) = No_{iter} \cdot (t_{comp}(iter)) \tag{5.4}$$

Este esquema lo siguen los algoritmos de agrupamiento. En algunos casos, el costo de la evaluación del criterio de convergencia es pequeño con relación al tiempo dedicado a la computación, cuyo costo se eleva debido a la evaluación de las funciones $u(m_k|e_p)$ y μ_i en cada iteración (ver página 34 de la sección 3.2). O bien, en otros casos, el criterio de convergencia está determinado por la comparación de un umbral con la norma de un vector de diferencias de orden $O(n)$, el cual es despreciable cuando el costo de la computación es de orden superior ($O(n^2)$ por ejemplo).

La situación no siempre es simple. En cada iteración del algoritmo KM, descrito en la sección 3.3.1, se evalúa tanto la función de pertenencia (ecuación 3.7) como la función que recalcula los centroides (ecuación 3.8). El número de veces que estas funciones se evalúan coincide con el número de patrones en E, sin embargo el costo de cada evaluación puede variar en función del grupo sobre el cual se este operando, por lo que en la ecuación 5.2, el valor que puede tomar $t_{comp}(iter)$ es distinto para cada iteración.

En algunas ocasiones, el costo de cada iteración depende de la forma de los parámetros de entrada y de los mecanismos utilizados para la selección del conjunto de valores que minimizan la función objetivo.

Cabe mencionar que en los esquemas paralelos se hace énfasis en esquemas de «grano grueso» evitando que el proceso de comunicación o sincronización represente un porcentaje alto en el tiempo total de ejecución. Debido a esta situación, se introduce la variable «granularidad» que se utiliza para definir un mayor o menor peso en la computación con relación a las comunicaciones dentro de la ejecución del algoritmo.

Este trabajo se centra en el comportamiento de esquemas paralelos iterativos. Da-

Algoritmo 4 Esquema iterativo paralelo

Entrada: Parámetros de entrada
Salida: Parámetros de salida

```
1: procedure ParallelIterativeSchema
2:     Conjunto de instrucciones
3:     for P_i,  i = {1, ..., p} do
4:         while ConditionEvaluation(Parámetros) do
5:             Conjunto de instrucciones
6:             Instrucciones de comunicación o sincronización entre unidades de procesamiento
7:             Actualización de parámetros y condiciones
8:         End while
9:         Conjunto de instrucciones
10:    End for
11: End procedure
```

do que existen distintas posibilidades para estos esquemas, su comportamiento y la metodología de paralelización puede variar dependiendo de la arquitectura paralela al que están destinados. En este trabajo se consideran dos tipos de arquitecturas, las homogéneas y las heterogéneas (cuyas características han sido explicadas en la página 75 de la sección 4.1). En la sección 5.3 se define la idea general de los esquemas iterativos paralelos orientados a arquitecturas homogéneas, y en lo subsecuente, esquemas heterogéneos.

5.3. Esquema paralelo homogéneo

Un esquema iterativo paralelo se obtiene añadiendo al esquema secuencial (ver algoritmo 3) una parte de comunicación o sincronización después de haber realizado cada iteración, tal como se describe en el esquema algorítmico paralelo representado por el pseudocódigo 4.

En programación paralela de memoria compartida se define una operación de sincronización, y mediante programación paralela de memoria distribuida con paso de mensajes se tendrán operaciones de comunicación que además, aseguran la sincronización.

De esta forma, la ecuación 5.2 se reescribe como

$$t_{Iterativo} = \sum_{iter=1}^{No_{iter}} (t_{comp}(iter) + t_{comm}(iter) + t_{cond}(iter)), \tag{5.5}$$

donde $t_{comm}(iter)$ representa el tiempo de comunicación o sincronización después de la computación de un conjunto de instrucciones. Si el costo de las distintas iteraciones

coincide, entonces

$$t_{Iterativo} = No_{iter} \cdot (t_{comp}(iter) + t_{comm}(iter) + t_{cond}(iter)). \tag{5.6}$$

La evaluación del criterio de convergencia incluye un costo derivado del tiempo de cómputo (t_{comp}) seguido de uno de comunicación (t_{comm}) debido a que el criterio de convergencia será global en todas las unidades de procesamiento. De esta manera, el costo por iteración del algoritmo se descompone en dos procesos de computación intercalados por dos procesos de comunicación.

En algunos casos el costo de la actualización y evaluación de la condición de convergencia es despreciable respecto al costo de computación y comunicación. Por ejemplo, cuando se dispone de un número específico de iteraciones. En estos casos no se incluye el término t_{cond} en la ecuación 5.6 y si, además, el costo de cómputo y de comunicación es similar en todas las iteraciones, el costo total se define con la expresión 5.7.

$$No_{iter} \cdot (t_{comp} + t_{comm}). \tag{5.7}$$

Además del número máximo de iteraciones, en los algoritmos de agrupamiento, otro criterio criterio de convergencia que normalmente se utiliza es la minimización de una función objetivo (ver ecuación 3.6 para KM, ecuación 3.9 para FCM y ecuación 3.12 para EM). En este criterio se establece un umbral o valor tolerancia lo suficientemente pequeño que establezca que no ha sido posible mejorar la solución en un determinado número de iteraciones.

Se describe el costo computacional ($t_{Iterativo}$) del esquema iterativo descrito en el pseudocódigo 1 que define las técnicas de agrupamiento particional. Se considera que esta función es común para los algoritmos descritos en el capítulo 3. Sin embargo, si se asume que se conoce el número de iteraciones No_{iter} *a priori* debido a que el criterio de convergencia no ha sido definido a través de un umbral, ρ que diferencía a la solución obtenida en la iteración anterior $(iter - 1)$ de la iteración actual $(iter)$. Entonces, el costo se puede modelar como

$$t_{Iterativo} = No_{iter} \cdot (t_{u(m_k|e_p)} + t_{w(e_p)} + t_{m_k} + t_{cond}) \tag{5.8}$$

donde se supone que el costo del subsistema computacional en cada iteración viene dado por el costo de evaluación de la función de pertenencia ($t_{u(m_k|e_p)}$) y el costo de la evaluación de la función de peso ($t_{w(e_p)}$). Además, se define a t_{m_k} como el costo de recalcular los K centroides tal que represente el costo del subsistema de comunicación, es decir, el costo computacional de las comunicaciones que se realizan en el algoritmo distribuido; en este sentido, es necesario considerar que si se comparte un gran número de datos, en este caso patrones entre los nodos y se asignan a todas las agrupaciones, el costo de comunicación será muy alto, sin embargo, puede que la convergencia sea rápida al compartir más información entre cada agrupación; por otro lado si se comparten pocos patrones, los procesos de comunicación serán menos costosos pero la convergencia más lenta. El término t_{cond} representa la comprobación de la condición de convergencia que se puede realizar cada vez que se termina una

iteración.

Como ya se había mencionado en la sección 3.2, cada algoritmo de agrupamiento se compone de operaciones aritméticas básicas que conforman a las tareas de inicialización aleatoria de los centroides, cálculo de la función de pertenencia, evaluación de la función de peso, actualización de los centroides, comprobación de la condición de convergencia, entre otras. De esta forma se identifican parámetros aritméticos distintos para cada una de las operaciones, y a su vez dependen también del algoritmo de agrupamiento concreto y del problema al que se este aplicando. Por lo tanto, el costo de las operaciones elementales que se realizan durante alguna fase del algoritmo de agrupamiento se definen como el producto del costo de una operación elemental y el número de veces que esta se realiza. Por ejemplo,

$$t_{u(m_k|e_p)} = \delta_{i,j} \cdot n_{u(m_k|e_p)} \tag{5.9}$$

donde $\delta_{i,j}$ representa el costo de evaluar la función de pertenencia de cada uno de los patrones a cada uno de los centroides y $n_{u(m_k|e_p)}$ el número de operaciones elementales que se realizan en esta etapa. Sin embargo, y por facilidad se ejecutó secuencialmente una iteración del algoritmo y se identificó el parámetro que afecta a la computación: $(n_{u(m_k|e_p)} + n_{w(e_p)}) \cdot \delta_{i,j}(iter)$. Lo mismo aplica para la condición de la evaluación del criterio de convergencia $n_{cond} \cdot \delta_{i,j}(cond)$.

Respecto a la estimación de parámetros relacionados con el subsistema de comunicación, se utilizó la técnica de *ping-pong* para proponer un modelo líneal clásico $t_{inicio}+m \cdot t_{envío}$ que estime los valores de los parámetros t_{inicio} y $t_{envío}$. Sin embargo, no se consideró el tipo de comunicación que realiza el algoritmo. El modelo matemático que representa el tiempo de ejecución total para el caso secuencial es más complicado, debido a que cada iteración puede ser diferente entre sí, y en consecuencia cada una de ellas tiene un costo computacional distinto.

Lo anterior, hace que no sea posible la estimación de parámetros con la ejecución simple de una iteración ni de un proceso de comunicación. Por lo tanto, se deben de identificar cada una de las operaciones que componen a cada iteración. Enseguida, estas operaciones se deben ejecutar durante la fase de instalación para conocer su costo computacional. De esta forma, si una operación se repite dentro de un bloque de instrucciones más de una vez, y aunque su costo computacional sea distinto, solo será necesaria la ejecución de una iteración para identificar los parámetros a pesar de que la ecuación de costo total dependa del número de iteraciones a realizar por parte del algoritmo.

Puede ocurrir que el costo de la computación o de las comunicaciones sea distinto en unidades de procesamiento diferentes, y que además, difiere con base en el número de iteración. Si así fuera, el costo computacional de las operaciones en la ecuación 5.5 se definirá con base en el tiempo computacional que requiere el procesador que tarda más en terminar. De esta forma, el tiempo computacional se representa como

$$t_{paralelo} = \sum_{iter=1}^{No_{iter}} \left(\max_{i_0=1,\ldots,p} \{t_{comp}(iter, i_0)\} + \max_{i=1_1,\ldots,p} \{t_{comm}(iter, i_1)\} + \max_{i_2=1,\ldots,p} \{t_{cond}(iter, i_2)\} \right), \tag{5.10}$$

donde por cada operación e iteración se evalúa el tiempo máximo que requiere cada una de las unidades de procesamiento que conforman a la arquitectura.

A pesar de que puede existir solapamiento entre las tareas de computación y comunicación, no han sido considerados dentro de la ecuación 5.10. El problema de optimización a resolver será obtener el número de procesadores p_s de todos los disponibles en el sistema con el que el valor de la ecuación 5.9 sea mínimo.

5.4. Esquemas paralelos heterogéneos

Para esta sección se consideran las arquitecturas heterogéneas y los esquemas iterativos. El esquema básico de un algoritmo iterativo para un algoritmo heterogéneo coincide con el algoritmo homogéneo (ver algoritmo 4). En este sentido, se considera un conjunto de iteraciones sucesivas que se ejecutan hasta que se cumple algún criterio de convergencia. En cada una de las iteraciones se define una tarea de cómputo seguida por una tarea de comunicación y/o sincronización.

Un aspecto que debe tomarse en cuenta durante la optimización del tiempo de ejecución de un algoritmo paralelo en una arquitectura heterogénea es que la velocidad de cómputo y de comunicación varía en al menos una unidad de procesamiento.

El modelo del tiempo de ejecución para arquitecturas heterogéneas es similar al modelo planteado para sistemas homogéneos. Considere la ecuación 5.5, la cual supone que el costo en cada iteración es el mismo. Entonces, el costo de una iteración en una arquitectura heterogénea incluye el tiempo de cómputo y comunicación sin considerar la evaluación del criterio de convergencia. A pesar de que el costo en cada unidad de procesamiento varía incluso, en los casos donde se asigna el mismo volumen de datos a cada procesador, el tiempo de ejecución se puede estimar mediante el producto del número de operaciones elementales y su costo computacional en cada unidad de procesamiento. Esto permite modelar el costo computacional en la i-ésima unidad de procesamiento. De esta forma, el costo computacional de una iteración será el máximo de los costos en todas las unidades de procesamiento:

$$t_{Iterativo} = \max_{i=1,\ldots,p} \{n_{comp}(i)\delta_{i,j}\} \tag{5.11}$$

Para introducir el tiempo de comunicación en el modelo matemático se procede de forma similar. De esta manera, se considera la cantidad de operaciones relacionadas al inicio de las comunicaciones ($n_{inicom}(i,j)$), el número de datos transferidos entre dos procesos ($n_{dattra}(i,j)$), y el costo de las comunicaciones representado por el máximo de los inicios de comunicación y de los datos transferidos tal que,

$$\max_{i=1,\ldots,p;j=1,\ldots,p} \{n_{inicom}(i,j)t_{inicio_{i,j}}\} \tag{5.12}$$

$$\max_{i=1,\ldots,p;j=1,\ldots,p} \{n_{dattra}(i,j)t_{envío_{i,j}}\} \tag{5.13}$$

y donde, para cada par de procesos se considera el producto entre el número de operaciones relacionadas al inicio de comunicaciones entre dos procesos y el costo de

inicio de una comunicación. De esta forma se considera que el costo total de inicio de comunicación en el sistema paralelo es el máximo entre cualquier pares de procesos. Lo mismo aplica para la transferencia de datos.

El modelo del tiempo puede ser expresado por la siguiente expresión

$$\begin{aligned} t_{Iterativo} = \sum_{iter=1}^{No_{iter}} (& \max_{i_0=1,\ldots,p} \{n_{comp}(i_0)\delta_{i_0,j}\} \\ & + \max_{i_1=1,\ldots,p;j_1=1,\ldots,p} \{n_{inicom}(i_1,j_1)t_{inicio_{i_1,j_1}}\} \\ & + \max_{i_2=1,\ldots,p;j_2=1,\ldots,p} \{n_{dattra}(i_2,j_2)t_{envío_{i_2,j_2}}\}). \end{aligned} \tag{5.14}$$

Con la distribución de los datos en el sistema se intenta que el trabajo esté equilibrado, por lo que los valores de los que se tomen esos máximos deberían relativamente coincidir entre los distintos procesadores. Puede que el balance del tiempo de cómputo afecte al balance en el tiempo de comunicación y/o sincronización. De ahí que, para el modelo propuesto no es suficiente realizar la distribución del trabajo de forma tal que

$$n_{comp}(i)\delta_{i,j} = n_{comp}(j)\delta_{i,j}. \tag{5.15}$$

Cuando el costo de comunicación es mucho menor que el costo de cómputo, sólo se toma en cuenta el costo de cómputo y se opta por una distribución que satisfaga la ecuación 5.15.

De forma general, se describe un problema de optimización que toma en cuenta todo el espacio de soluciones y selecciona aquella solución que minimice o maximice el valor final de una función objetivo. Para la función descrita en la ecuación 5.9 el problema de optimización es:

$$\begin{aligned} \min_{\sigma \in Asignaciones} \{ & \sum_{iter=1}^{No_{iter}} (\max_{i_0=1,\ldots,p} \{n_{comp}(i_0,\sigma)\delta_{i_0,j}\} \\ & + \max_{i_1=1,\ldots,p} \{\sum_{j=1}^{p} (n_{inicom}(i_1,j,\sigma)t_{inicio_{i!j}} + n_{ndattra}(i_1,j,\sigma)t_{envío_{i_1,j}})\})\} \end{aligned} \tag{5.16}$$

Se discute brevemente cómo se realiza la estimación de los parámetros del sistema, $\mathcal{S}$, en un entorno heterogéneo. En el análisis de este trabajo se considera un sistema estáticos cuya capacidad de cómputo y comunicación de cada uno de los elementos que lo integran son distintos, sin embargo, no varían con el tiempo. Para este caso particular, los parámetros del sistema se estiman en la instalación de la rutina. Los parámetros de computación (el vector $\delta_{i,j}$) se estiman a partir de la solución secuencial de una versión reducida del problema en cada uno de los procesadores.

Es probable que para realizar la estimación de la parte de comunicaciones se deben utilizar diferentes posibilidades. En este sentido, se utilizan los valores de t_{inicio} y $t_{envío}$ entre cada par de procesadores mediante la ejecución de una versión simplificada de la operación de comunicación que se este utilizando durante la ejecución del algoritmo.

Si bien, esta propuesta simplifica el problema, la operación de comunicación puede ejecutarse de diversas formas debido a que el costo de la operación se define mediante la asignación de datos a unidades de procesamiento.

Lo anterior puede generar condiciones distintas. En este sentido se propuso la realización de experimentos con configuraciones distintas y después tomar la media de los parámetros o almacenar los parámetros de comunicación en una base de datos en función de algunas configuraciones básicas. Sin embargo, en entornos heterogéneos se pudiese tratar como un caso simplificado que no incluye comunicaciones, y que considera que el costo computacional de las operaciones es considerablemente mayor, situación que ocurre en muchas de las aplicaciones paralelas.

También se considera que en arquitecturas heterogéneas se realiza la asignación de un proceso con distinto volumen de datos por procesador. De esta forma el parámetro algorítmico a optimizar es el tamaño del bloque de datos que se debe ser asignado: $B = (b_1, \ldots, b_p)$. En consecuencia, se tendrán tantos parámetros como procesadores. Para obtener el tamaño de $b_i \in B$ que minimiza el valor del tiempo de cómputo teórico en la fase de ejecución se evalúa el modelo matemático que representa el tiempo de cómputo necesario durante la solución de un problema. A tal efecto se consideran los parámetros del sistema obtenidos en la instalación y los parámetros representativos del tamaño del problema que se está resolviendo. Finalmente, se resuelve el problema considerando los valores prácticos obtenidos.

Este problema es clasificado dentro de los problemas NP-completo ya que la cantidad de parámetros a optimizar depende de la características del sistema, y del dominio de solución. Para este tipo de problemas no existe un algoritmo que encuentre una solución en tiempo polinomial. Una alternativa que se usa normalmente y que se utilizó en este trabajo consiste en usar heurísticas con el objetivo de encontrar una buena solución cercana a la mejor solución. Esto hace posible la ejecución del algoritmo en un tiempo de cómputo reducido y sin añadir una sobrecarga durante la toma de decisiones.

Cabe señalar que con esta optimización, solo se ha considerado el tamaño del bloque de datos que debe ser enviado a cada unidad de procesamiento. En la sección 5.5 sse describirá la metodología utilizada para definir el tamaño de bloque, así como el conjunto de banderas del compilador que controlan el flujo de instrucciones en tiempo de compilación. Siendo esta última, la principal aportación de este trabajo.

5.5. Auto-adaptación de código paralelo

El enfoque de auto-adaptación mejora la optimización tradicional guiada por un experto en HPC al descargar parte o toda la búsqueda de una implementación óptima del algoritmo a una técnica de paralelización automatizada. Sea una configuración, el conjunto de parámetros definidos por: (1) El tamaño de bloque que debe ser resuelto por cada unidad de procesamiento; y (2) El conjunto de banderas del compilador que optimizan el flujo de instrucciones en tiempo de compilación. Se propone un «marco de trabajo de auto-adaptación» que es capaz de seleccionar de entre todas las posibles configuraciones que definen el espacio de búsqueda, aquella que defina

la mejor implementación. Su aplicación mejora la optimización tradicional de un programa en tiempo de compilación.

Considere que para la representación de una configuración se debe tomar en cuenta: (1) La correcta representación de la configuración para un algoritmo específico; y (2) La definición del tamaño del espacio de búsqueda o dominio, el cual garantice una configuración óptima. En el primer aspecto se señala que una configuración puede estar representada por un entero que define un tamaño de bloque, así como un dato complejo representado por un árbol sintáctico de expresiones. Para este caso, se han considerado estructuras de datos con restricciones de dominio, las cuales especifican los valores permitidos para un atributo determinado. Para el segundo aspecto, la búsqueda exhaustiva no se contempla como una solución, ya que se sabe que el tamaño del espacio de búsqueda puede ser lo suficiente grande, hasta 10^{806} configuraciones posibles para una arquitectura de referencia en este estudio que el análisis no se completaría en lo que resta de la vida del Universo; por lo tanto es necesaria la aplicación de técnicas heurísticas que obtengan la configuración óptima. Se ha considerado la heurística de un enjambre de partículas (*Particle Swarm Optimization* en inglés).

5.5.1. Descripción del marco de trabajo

En el estado del arte, el problema de auto-adaptación se ha abordado desde un punto de vista unificado tomando en cuenta un modelo analítico que representa el tiempo de ejecución teórico en tiempo de compilación. El diseño de esta propuesta se basa en la metodología `SANS` (*Self-adapting Numerical Software*) [58], la cual se describe como un marco de referencia para cualquier metodología de auto-adaptación.

De manera general `SANS` se integra de seis componentes básicos:

- Un componente inteligente que estudia la estructura de los datos que recibe el algoritmo.
- Un componente de sistema capaz de recopilar información de los recursos de la arquitectura.
- Un componente histórico que almacena una serie de datos que representan las características de *hardware* y *software* que describen un problema resuelto con anterioridad.
- Un lenguaje que permite la comunicación entre el usuario y el sistema.
- Un diccionario de los datos que describe los datos que se almacenan dentro del componente histórico.
- Un conjunto de bibliotecas compiladas de manera óptima en diferentes arquitecturas.

A pesar de que existe un amplio conjunto de trabajos que plantean metodologías para el diseño y programación de código auto-adaptable, la relevancia en la mayoría de ellos radica en el Álgebra líneal [24, 109, 176, 204, 216, 223]. En la tabla 5.1 se

Tabla 5.1: Resumen de proyectos que utilizan la adaptación automática en arquitecturas paralelas. Se describe el dominio o área de interés y la heurística utilizada para obtener una configuración óptima. Las bibliotecas `PetaBricks` y `HPL` (marcadas con color azul) serán tomadas como referencia para validar los resultados obtenidos por la propuesta de este trabajo. En el estado del arte, `HPL` se considera como un criterio de evaluación del TOP500 de las supercomputadoras, el cual sugiere la adaptación de al menos 15 parámetros para obtener el máximo rendimiento de un algoritmo (factorización LU) en una arquitectura paralela.

Proyecto	Espacio de búsqueda	Heurística
`Active harmony` [208]	Sistemas de ecuaciones	Nelder-Mead
`ATLAS` [223]	Álgebra líneal	Búsqueda exhaustiva
`FFTW` [72]	Transformada rápida de fourier	Programación dinámica
`Insieme` [107]	Banderas de compilación	Evolución diferencial
`OSKI` [220]	Álgebra líneal	Búsqueda exhaustiva
`PetaBricks` [7]	Lenguaje de programación	Algoritmos evolutivos
`HPL` [198]	Lenguaje de programación	Búsqueda exhaustiva
`Sepya` [165]	Banderas de compilación	Método del descenso del gradiente
`SPIRAL` [176]	Procesamiento digital de señales	Métodos basados en Pareto

describe el dominio y las técnicas heurísticas de los nueve proyectos más relevantes en esta área. Cabe mencionar que las bibliotecas `PetaBricks` y `HPL` fueron tomadas como referencia para validar los resultados obtenidos por la propuesta descrita en este trabajo. En el estado del arte, `HPL` se considera como un criterio de evaluación del TOP500 de las supercomputadoras, el cual dada una arquitectura paralela mide el máximo rendimiento de un algoritmo, específicamente la factorización LU, a través de la adaptación de al menos 15 parámetros algorítmicos.

Enfoque de auto-adaptación propuesto

El problema de auto-adaptación en este trabajo es tratado como un problema de optimización, donde el espacio de búsqueda se integra por la amplia variedad de configuraciones. Se define el término «evaluación del espacio de búsqueda» como la evaluación de una función objetivo que optimice el tiempo de ejecución de un algoritmo, la precisión del resultado o la cuantificación de cualquier otra métrica en tiempo de compilación. El resultado de esta evaluación es la configuración óptima.

En la figura 5-1 se describe el marco de trabajo incluyendo cada uno de los componentes principales. El componente de «`configuración`» define las técnicas heurísticas que el auto-adaptador utiliza. El auto-adaptador es definido por el usuario para leer y escribir posibles configuraciones. El componente de «`evaluación`» se encarga de estimar el valor de la función objetivo considerando las configuraciones candidatas. En este caso, la función objetivo evalúa el tiempo de ejecución de un algoritmo. Estos dos componentes realizan múltiples estimaciones de la función objetivo simultáneamente. Ambos componentes se comunican exclusivamente a través del componente denominado «`base de datos`», el cual registra todos los resultados recopilados durante el proceso de instalación y ejecución. Específicamente, durante la instalación se recaban las características de la arquitectura; en la ejecución, se optimiza la función objetivo considerando el tamaño del problema y el conjunto de banderas propias del compilador. Para esta última etapa, el usuario debe definir un dominio de búsqueda que incluya el conjunto de parámetros a optimizar.

Esta propuesta está motivada por tres factores:

- Permitir el paralelismo entre múltiples heurísticas. El paralelismo es más importante en los procesos de medición ya que en la mayoría de los marcos de trabajo de auto-adaptación dominan los costos de evaluación de la función objetivo. Para solventar este problema, el componente de `configuración` realiza múltiples solicitudes de resultados deseados al componente de `evaluación` sin esperar a que se cumpla cada solicitud. Si una heurística específica se bloquea en espera del resultado, se descarta y se descarta la posible configuración. De esta forma se evita el tiempo de inactividad del componente de `evaluación`.

- El componente de `evaluación` apoya al entrenamiento y posterior toma de decisiones. En esta propuesta, la auto-adaptación no solo se realiza antes de la implementación de un algoritmo, sino que existe una optimización previa durante la instalación del marco de trabajo. El componente `evaluación` puede ser reemplazado por un componente de aprendizaje específico del dominio que examine periódicamente la base de datos para decidir qué configuración usar y registrar el rendimiento, por ejemplo una red neuronal.

- El componente de `evaluación` puede ser modificado para su adaptación en otros lenguajes de programación sin necesidad de modificar la mayor parte del marco de trabajo. Lo anterior se describe tomando ventaja de la POO.

5.5.2. Heurística como componente inteligente

El marco de trabajo requiere la implementación de una heurística que permita la optimización de los parámetros del modelo. En particular, la heurística empleada ha sido PSO [145] y fue implementada en `python`. Sin embargo, dada la flexibilidad que propone este lenguaje de programación, podría optarse por alguna heurística de un amplio catálogo a conveniencia del usuario.

Etapas de auto-adaptación:

(1) Instalación

(2) Ejecución

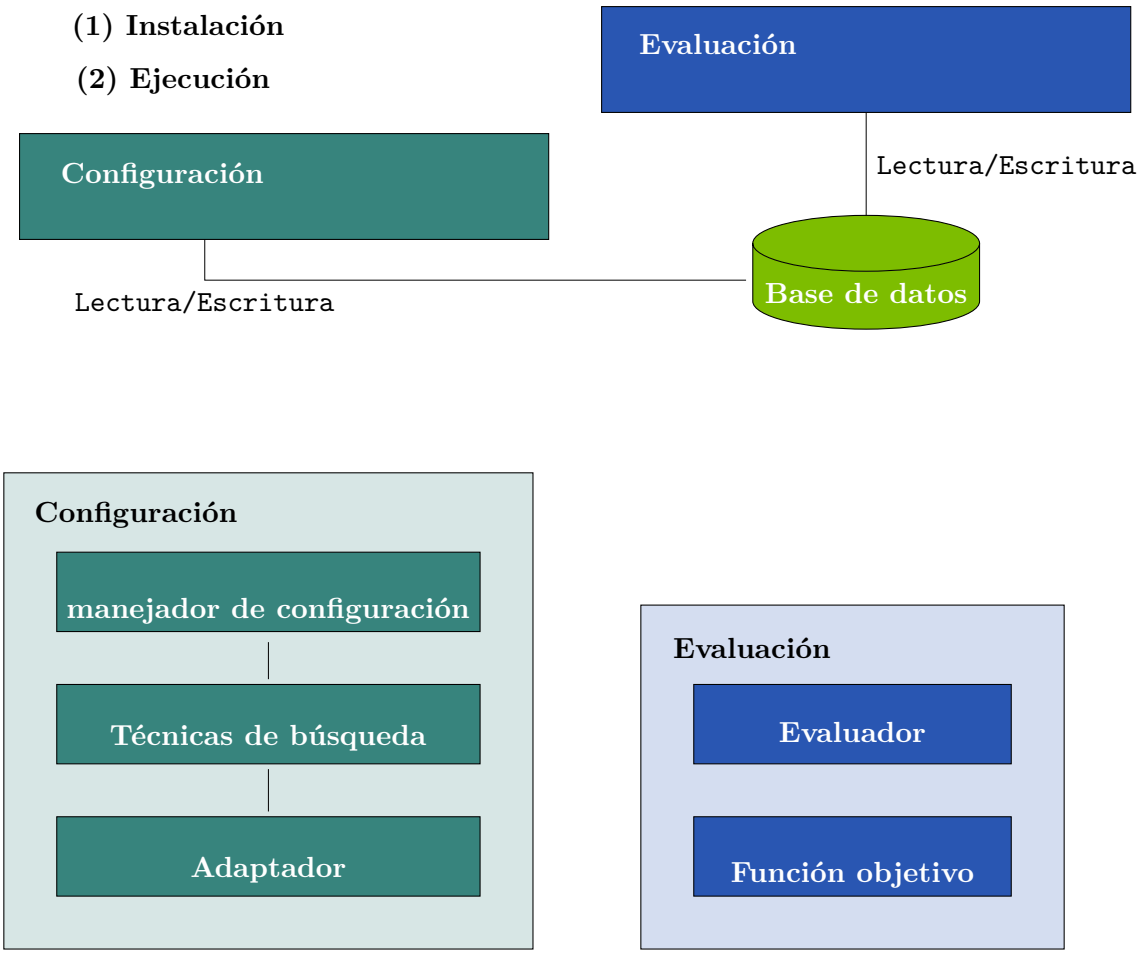


Figura 5-1: Descripción general de los componentes principales del marco de trabajo. El componente de `configuración` define las diferentes técnicas heurísticas que servirán para optimizar la función objetivo. Se añade el manejador de configuración que se encarga de seleccionar las técnicas heurísticas que se utilizarán para optimizar el código fuente y los parámetros descritos en el adaptador. Este último es definido por el usuario. Respecto al componente `evaluación` se define el Evaluador, el cual se encargará de realizar las diversas ejecuciones utilizando las técnicas heurísticas seleccionadas por el manejador de configuración. Finalmente, se encuentra el módulo que almacena la función objetivo. Está función objetivo evalúa el tiempo de ejecución de un algoritmo, sin embargo puede ser adaptada para evaluar el consumo de energía. Finalmente, la comunicación entre ambos componentes solo se realiza con llamadas a procesos de lectura y escritura a una base de datos.

PSO hace referencia al comportamiento de las partículas en la naturaleza. El algoritmo analiza una población (enjambre) de soluciones candidatas (partículas) con el objetivo de minimizar la función de costo f, en este caso, el tiempo de ejecución.

Durante su ejecución, se almacenan los resultados obtenidos en una base de datos de modo que, los resultados obtenidos durante un experimento, puedan beneficiar a otros experimentos. Este intercambio ocurre de forma específica; por ejemplo, los mejores resultados obtenidos en un experimento contribuyen a la selección de soluciones candidatas en futuros experimentos, ya que se consideran como la mejor posición en la que ha estado la partícula. Por otro lado, los vectores de velocidad y posición de cada partícula son calculados mediante combinaciones de configuraciones generadas durante el proceso de búsqueda de cada experimento.

5.5.3. El adaptador como lenguaje de comunicación

El «`adaptador`» es elemento que forma parte del componente configuración (ver figura 5-1) y se describe como la interfaz que permite la comunicación entre la heurística y el dominio de búsqueda definido por el usuario. Su implementación cuenta con valores por defecto que incluye el conjunto de todas las banderas del compilador que se utilice. Este conjunto se almacena a través de un diccionario de datos que incluye el nombre del parámetro y su tipo de dato. Sin embargo, el usuario puede ampliar el conjunto de parámetros a optimizar por el adaptador para cambiar la estructura de datos que se utiliza para las configuraciones o para admitir un conjunto dinámico de parámetros que dependen de la instancia inicial de configuración.

Los formatos de datos que soportan el `adaptador` son `json` y `pickle`. El primero de ellos fue considerado por ser un formato de texto independiente del lenguaje debido a su amplia adopción como alternativa a `XML`; por otro lado, `pickle` representa un objeto de `python` como una cadena de bytes para ser utilizado de manera transparente por otro programa. Además, en este modulo se ejecutan operaciones de serialización y deserialización que determinan que tan rápido se ejecuta un programa que tan seguro es y cuánta libertad se tiene para garantizar su integridad.

5.5.4. Diccionario de datos

El diccionario de datos se describe como una jerarquía de parámetros (ver figura 5-2) y se integra dentro del «`manejador de configuración`». Con base en la ámplia variedad que existe para representar una configuración, se identifican dos tipos de parámetros: (1) «parámetros primitivos» que describen los valores caracterizados por un valor numérico contenidos entre una cota superior e inferior y (2) «parámetros complejos» que presentan una visión global de las configuraciones; a diferencia de los parámetros primitivos, los parámetros complejos tienen un conjunto variable de operadores auto-adaptables que realizan cambios estocásticos en la bandera del compilador que se quiere optimizar. De esta forma este tipo de parámetros se utilizan para agregar estructuras específicas de configuraciones.

Específicamente, los parámetros primitivos: `Float`, `logFloat`, `Integer` y `logInteger`, tienen representaciones idénticas dentro de la configuración del adaptador, pero pre-

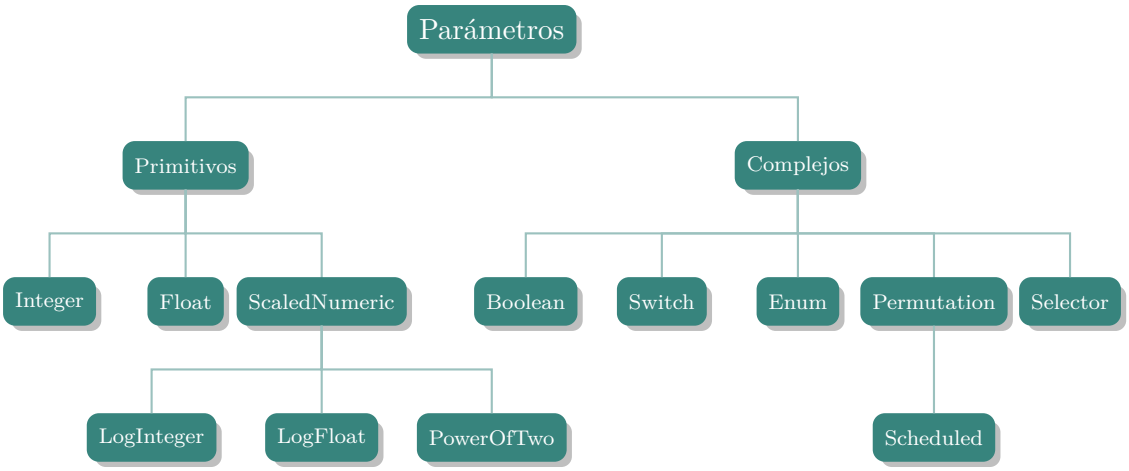


Figura 5-2: Jerarquía de parámetros integrados definidos por los tipos de parámetros. Los parámetros se dividen en dos clases: (1) Primitivos y (2) Complejos. Los primeros se refieren a los parámetros numéricos descritos entre una cota superior e inferior determinada para una configuración determinada. Por otro lado, los segundos se refieren al conjunto de parámetros de operadores que realizan procesos estocásticos para las heurísticas implementadas.

sentan una vista diferente del valor subyacente que trata de optimizar las heurísticas. En este sentido, `Float` se utiliza directamente en las heurísticas, mientras que `logFloat` describe una vista a escalada del parámetro subyacente. Las variantes de parámetros a escala logarítmica son mejores en parámetros asociados a los tamaños de bloque, donde los cambios fijos en los valores tienen efectos decrecientes cuanto mayor es el parámetro. `PowerOfTwo` es un caso especial de uso común, similar a `LogInteger`, debido a que los valores que pueden adquirir el parámetro están restringidos a potencias de dos.

Los tipos de parámetros `Boolean`, `Switch` y `Enum` teóricamente también se pueden representar como parámetros primitivos debido a que cada uno de ellos es utilizado directamente a una representación de un tipo de dato `Integer`. Por otro lado, el tipo de parámetro `Permutation` asigna un orden a un conjunto dado de valores y tiene un adaptador específico que realiza varios tipos de cambios aleatorios. Finalmente, un parámetro `Selector` es un tipo especial de árbol que se usa para definir un mapeo de un `Integer` a uno de tipo `Enum`.

Cabe mencionar que cada parámetro es responsable tanto de la interfaz para la representación de un parámetro dentro de la configuración, como para definir un estándar de un parámetro en la heurística. De esta forma, un parámetro puede extenderse para cambiar la representación subyacente así como, para cambiar la abstracción proporcionada a las heurísticas.

5.5.5. Base de datos descrita como un componente histórico

Se propone un componente histórico definido por una base de datos que puede ser configurada con cualquier manejador de base de datos (DBM) basado en el estándar `SQL`, sin embargo, el estándar del marco de trabajo es `SQLite` siempre y cuando el

usuario no haya realizado la configuración de un tipo de base de datos específica. En este sentido, se permite que el usuario administre las sentencias de consulta para poder compartir los resultados obtenidos anteriormente sin añadir procesos de comunicación con el sistema.

Este componente se integra por trece tablas (ver figura 5-3). Se describe el núcleo del auto-adaptador que almacena los resultados obtenidos de los experimentos realizados para la auto-adaptación de un programa específico (Tabla `resultado`). Además, se almacenan las características de la arquitectura considerando el tipo de `CPU` o `GPU` que contiene el equipo, el número de núcleos y la memoria disponible. Además, se almacena el resultado «óptimo» en una tabla específica, así como los resultados «subóptimos» con el objetivo de que esta información sirva como datos de entrenamiento para futuros experimentos.

Se proponen dos tablas adicionales para almacenar las técnicas heurísticas implementadas en el código fuente del auto-adaptador y el conjunto de heurísticas que fueron utilizadas en las ejecuciones de un experimento, y finalmente, poder asignar una ponderación con base en la aportación que tuvo la técnica para obtener un resultado óptimo. En la figura 5-4 se describen ambas tablas. La ponderación se realiza a través del promedio que veces que una técnica obtuvo un mejor valor a comparación de las otras técnicas utilziadas para minimizar la función objetivo. Además, tiene por objetivo retroalimentar futuros experimentos (Tabla `tecnicaresultado`).

Para saber la técnica que aportó mejores resultados para minimizar la función objetivo en un proyecto específico, basta con hacer una sentencia `SQL`, tal como se observa en el código 5.1.

Código 5.1: Sentencia SQL que permite conocer la técnica que tuvo mayor ponderación durante la optimización de una función objetivo en un proyecto específico.

```
1 SELECT max(tr.peso), tr.idTecnica FROM tecnicaresultado as tr
2 left join programa as p on p.idPrograma = tr.idPrograma
3 left join proyecto as pr on pr.idProyecto = p.idPrograma
4 where pr.nombre="K-means v1.0";
```

5.5.6. Características de implementación

El marco de trabajo ha sido implementado en lenguaje de programación `python` debido a que simplifica la programación y es multiplataforma. Para utilizarlo solo requiere de la importación de bibliotecas a través de la palabra reservada `import`. El código 5.2 describe un ejemplo de su implementación. En este caso, el usuario ha definido el adaptador para optimizar el tiempo de ejecución del algoritmo KM secuencial a través de las banderas del compilador `GCC`. El principal objetivo es minimizar el tiempo de ejecución. En la sección 5.5.7 se describen los resultados de una versión ampliada de este código que obtiene una aceleración de hasta `3.4x` sobre `-O3`. En este ejemplo se ajustan 181 banderas de `GCC` sobre un código secuencial y 147 parámetros `GCC` incluyendo un rango de valores posibles, los cuales pueden o no ser definidos por el usuario, sin embargo su definición acelera la convergencia del marco de trabajo. Se seleccionan de entre los cuatro niveles de optimización `-O0`,`-O1`, `-O2`, `-O3`. Finalmente,

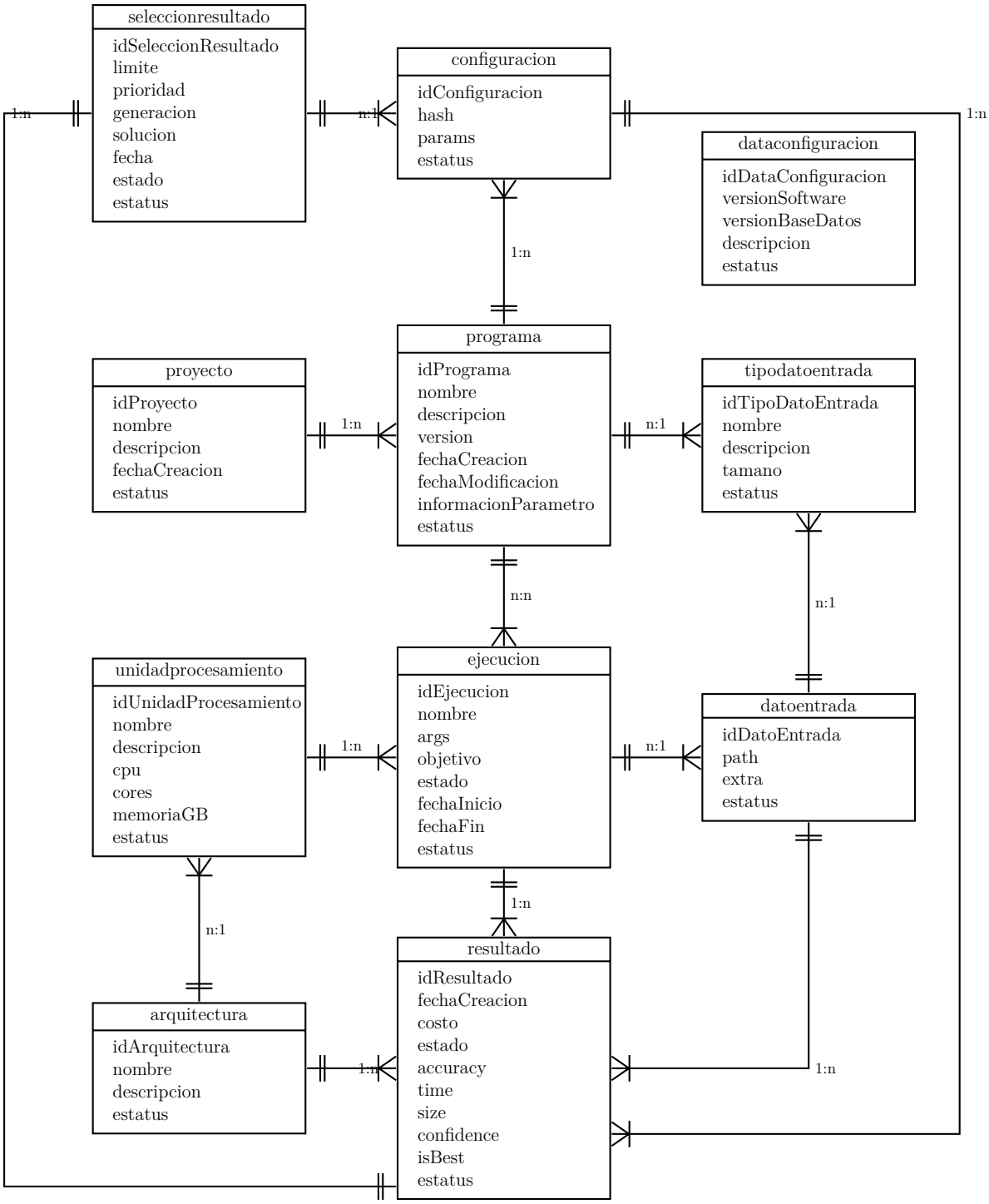


Figura 5-3: Base de datos que sirve como canal de comunicación entre el componente configuración y evaluación: Almacen de los mejores resultados considerando las características de la arquitectura destino. La base de datos se conforma de trece tablas, sin embargo, en este caso solo se describen once de ellas. Estas tablas almacenan los resultados temporales de la auto-adaptación de un programa específico, considerando la configuración utilizada, los valores subóptimos encontrados en cada ejección y considerando las características de la arquitectura. Finalmente, el resultado óptimo se almacena en una tabla `seleeccionresultado` para futuros experimentos.

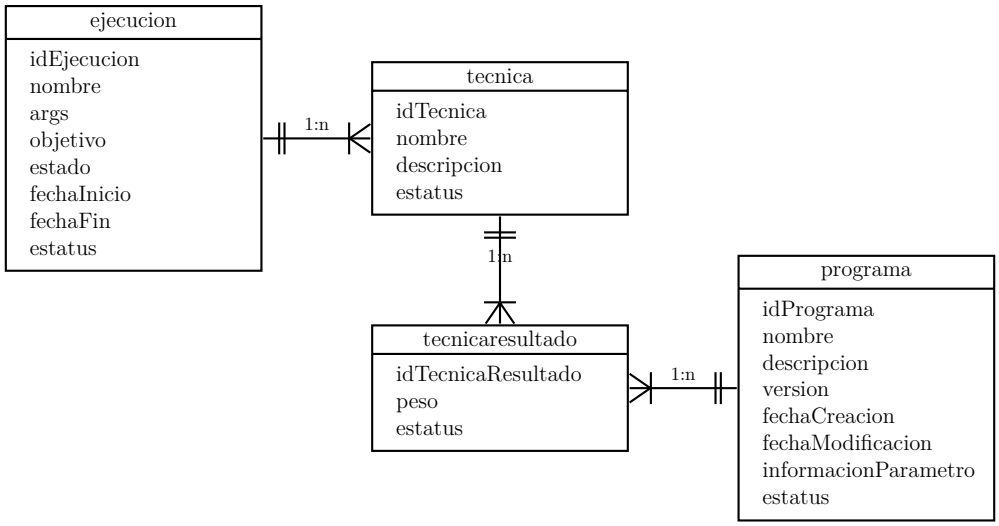


Figura 5-4: Base de datos que sirve como canal de comunicación entre el componente configuración y evaluación: Ponderación de las técnicas con mejores resultados en una ejecución durante la auto-adaptación de un programa. Esta parte de la base de datos está conformada por dos tablas, las cuales tienen el principal objetivo de almacenar las técnicas heurísticas que fueron utilizadas para minimizar la función objetivo. Cada una de estás técnicas recibe una ponderación con base en su aportación para resolver el problema.

la función `exec` descrita en la línea 37 del código 5.2 compila y ejecuta el algoritmo hasta obtener la configuración óptima deseada.

Cabe mencionar que el método `SelfAdapt` de la clase `gccFlagsSelfAdapt` (línea 23 del código 5.2) se invoca una vez en cada auto-adaptación para crear un objeto de configuración `configurationSelfAdapt` que define el dominio para todas las banderas `GCC`. Todas las llamadas a las configuraciones a través de las heurística se realizan a través de la clase `selfAdapt`. Para el nivel de optimización, se instancia el constructor de la clase `integerParameter` con parámetro entre 0 y 4. Para cada bandera, se crea un objeto de tipo `enumParameter` que puede activar, desactivar y predeterminar los valores. Para el resto de parámetros `GCC` acotados, se crea un objeto del tipo `integerParameter` con el rango apropiado (línea 34 del código 5.2).

La instancia del método `exec` (línea 37 del código 5.2) se encarga de realizar la comparación entre las distintas configuraciones. La configuración se realiza de manera similar a una línea de comandos específica para `C++`. La ejecución de esta línea de comandos genera un archivo binario (`tmp.bin`), el cual será ejecutado usando el método `callProgram`. `callProgram` es un método de conveniencia que mide el tiempo de ejecución de un programa, a través de las ecuaciones definidas en las secciones 5.3 y 5.4. Al terminar las ejecuciones se devuelve un resultado que también es almacenado en la base de datos. La base de datos está desarrollada en `SQLite`. El resultado representa un registro del tiempo computacional que requiere el algoritmo para ejecutarse en una arquitectura específica. Cabe mencionar que además del tiempo de ejecución, se puede incluir en trabajo a futuro el consumo de energía de un algoritmo.

Código 5.2: Banderas `GCC/G++` utilizadas en el marco de trabajo

```
import parallelSelfAdapt
from parallelSelfAdapt import configurationSelfAdapter
from parallelSelfAdapt import enumParameter
from parallelSelfAdapt import integerParameter
from parallelSelfAdapt import selfAdapt
from parallelSelfAdapt import Result

GCC_FLAGS = [
  'alignFunctions', 'align-jumps', 'align-labels',
  'branchCountReg', 'branchProbabilites',
  # ... (176 total)
]

# (name, min, max)
GCC_PARAMS = [
  ('earlyInliningInsns', 0, 1000),
  ('gcseCostDistanceRatio', 0, 100),
  # ... (145 total)
]

class gccFlagsSelfAdapt(SelfAdaptInterface):
  def SelfAdapt(self).
    """ Define the search space by creating
          the configuration of the Self-Adapter framework"""

    SelfAdapt = configurationSelfAdapt()
    SelfAdapt.addParameter(
      integerParameter('optLevel', 0, 3))
    for flag in GCC_FLAGS:
       SelfAdapt.addParameter(
         enumParameter(flag,['on','off','default']))
    for param, min, max in GCC_PARAMS:
      SelfAdapt.addParameter(
        integerParameter(param, min, max))
    return SelfAdapt;

  def exec(self, desiredResult, input, limit):
    """Compile and run a given configuration then
       return performance"""
    cfg = desiredResult.configuration.data
    gccCMD = 'g++ kmeans.cpp -o ./tmp.bin'
    gccCMD += '-O{0}'.format(cfg['optLevel'])
    for flag in GCC_FLAGS:
      if cfg[flag] == 'on':
        gccCMD += ' -f{0}'.format(flag)
      elif cfg[flag] == 'off':
        gccCMD += ' -fno-{0}'.format(flag)
    for param, min, max in GCC_PARAMS:
      gccCMD += ' --param {0}={1}'.format(param, cfg[param])

    compileResult = self.callProgram(gccCMD)
    assert compileResult['returnCode'] == 0
    runResult = self.callProgram('./tmp.bin')
```


Tabla 5.2: Definición del tamaño del dominio que considera el número de configuraciones posibles con representación en la biblioteca propuesta. Para el primer experimento, se realiza una exploración de las banderas del compilador `GCC/G++` que mejoran el rendimiento. Se consideran los tres niveles de optimización `-O1`, `-O2` y `-O3`, sin embargo, cabe mencionar que el último nivel es puramente experimental. Para los otros dos experimentos, se plantea la paralelización del algoritmo de factorización LU tridiagonal y la multiplicación de matrices utilizando el algoritmo de Strassen. Finalmente, considere que el número de posibles configuraciones obtenidas por el marco de trabajo propuesto pueden ser semánticamente equivalentes.

Proyecto	Punto de referencia	Número de configuraciones
`GCC/G++ Flags`	Es válido para cualquier algoritmo	10^{748}
`HPL`	n/a	$10^{9,9}$
`PetaBricks`	Algoritmo de Strassen	10^{153}

```
54      assert runResult['returnCode'] == 0
55      return Result(time=runResult['time'])
56
57  if __name__ = '__main__':
58      argparser = SelfAdapt.defaultArgparser()
59      gccFlagsSelfAdapt.main(argparser.parserArgs())
```

5.5.7. Resultados experimentales

El marco de trabajo fue validado a través de la implementación de tres adaptadores para tres experimentos respectivamente. Se describen cada uno de los experimentos, los adaptadores implementados, y finalmente se presentan los resultados mediante una comparación entre cuatro algoritmos. La tabla 5.2 describe el tipo de dominio en que el conjunto de datos ha sido aplicado y el número de posibles configuraciones que se pueden generar para cada uno de los tres experimentos con base en el marco de trabajo. Se debe considerar que no todas las configuraciones son diferentes y que algunas de ellas son semánticamente equivalentes. Por lo tanto, el espacio de búsqueda depende de la representación seleccionada por la metodología propuesta.

`GCC/G++ Flags`

La sección 5.5.3 describe a detalle la implementación del marco de trabajo orientado a la auto-adaptación de parámetros relacionados al proyecto `GCC/G++ Flags`, el cual busca encontrar un conjunto de banderas que mejoran el rendimiento del algoritmo KM. La lista de banderas ha sido omitidas en el código 5.2, sin embargo se describen en la implementación a través de una biblioteca de banderas. Se han considerado 181 banderas del total de las banderas implementadas en GCC. Se utilizó la

herramienta `Valgrind` [3] para minimizar los problemas de memoria y rendimiento que surgieron durante la ejecución de pruebas, los cuales en la mayoría de las ocasiones, estaban relacionados con errores internos del compilador.

El experimento considera un conjunto de banderas y parámetros definidos en el adaptador definido por el usuario. Enseguida, el marco de trabajo se enfoca en extraer los parámetros y rangos de la biblioteca `params.def` en `GCC` de manera automática. Lo anterior se logra a través de la bandera `g++ -help=optimizers`. Se consideró: (1) Límite de tiempo para abortar una prueba lenta que no será óptima; (2) Uso de tipo de parámetros `LogInteger`; y (3) Método `saveFinalConfig` para definir las banderas que podrán ser utilizadas durante la aplicación del marco de trabajo en la ejecución final. La ejecución final será aquella que ejecute el código con las banderas que mejoran su rendimiento.

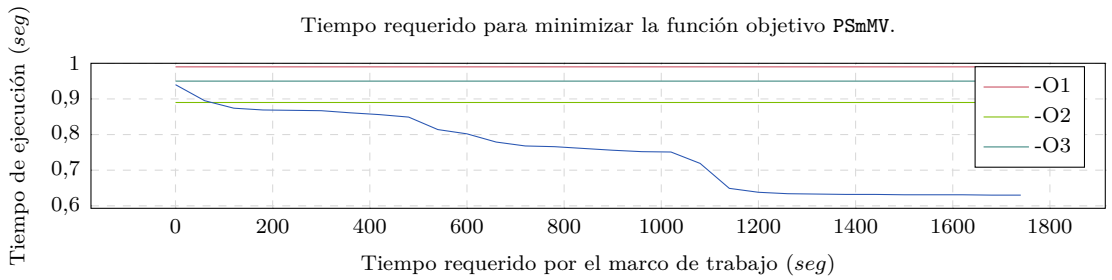
Los experimentos fueron probados en una arquitectura `gcc 7,5,0-3ubuntu1`, en una arquitectura con un procesador Intel Xeon E5-1603 v3 con 5 núcleos y sistema operativo Ubuntu 18.04 a 2.80 GHz. Se permitió el uso de la bandera `-ffast-math`, la cual permite cambiar el comportamiento de los tipos de datos `NAN` y tener un pequeño impacto en los resultados del algoritmo a ejecutar. Incluso así se observa un alto rendimiento si no se considera el uso de este tipo de banderas.

Se eligieron cuatro algoritmos destino a auto-adaptar a través de este experimento, los cuales fueron: (1) Producto matriz-vector en `C` (`PSpMV.c`) propuesto en la sección 3.5; (2) Producto matricial implementado en `C++`, (`matrixMultiply.cpp`) y propuesto en [67]; (3) algoritmo KM cuya implementación está hecha en `C` y `C++` y ha sido descrito en la sección 3.3.1; (4) algoritmo FCM implementado en lenguaje `C` y `C++` y ha sido descrito en la sección 3.3.2. Los últimos algoritmos han sido modificados para que se ejecuten de manera determinista y poder realizar una comparación más justa. El objetivo de este experimento es probar el marco de trabajo en un amplio rango de códigos optimizados.

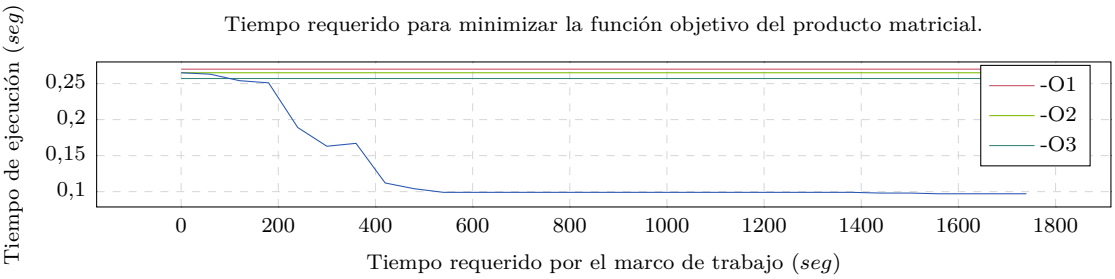
La figura 5-5 muestra la relación que existe entre el tiempo de ejecución en función del tiempo necesario por el marco de trabajo para la auto-adaptación de parámetros de los algoritmos descritos anteriormente (`PSpMV`, producto matricial, KM y FCM). Se añade el tiempo computacional utilizando la biblioteca de operaciones elementales descrita en la sección 3.5 sin considerar una auto-adaptación. Los resultados demuestran una variación del rendimiento computacional a razón de 1.16x para `PSpMV`, de 1.23x para el Producto Matricial, 1.14x para algoritmo KM y 1.16x para algoritmo FCM. Los resultados demuestran que para las figuras 5-5b y 5-5c, el uso de las banderas `-O3`, `-O2` y `-O1` no representan una mejora dentro de la ejecución del algoritmo. Se aprecia que la bandera `-O2` en el 75 % de las ocasiones está en la media de las banderas `-O3` y `-O1`.

Por otro lado, y para conocer cuanto contribuye cada una de las banderas de `GCC` en la mejora del rendimiento durante la ejecución de un algoritmo específico, se cuantificó la desaceleración resultante después de que una bandera no ha sido considerada en la línea de comandos GCC. En las tablas 5.3, 5.4, 5.5, y 5.6 se describen las primeras

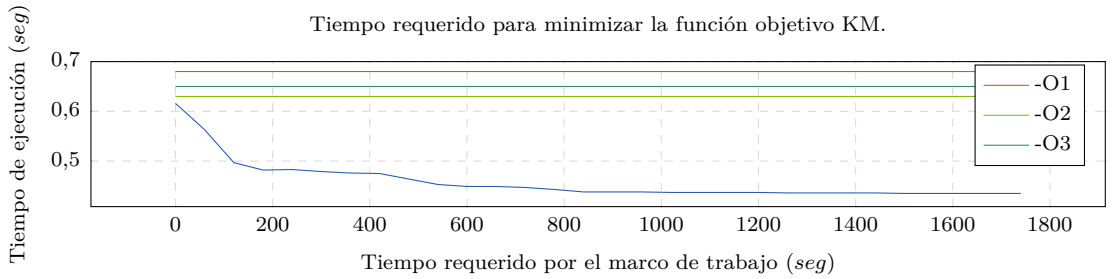
[3] `Valgrind` es un conjunto de herramientas libres que colaboran en la depuración de problemas relacionados al uso de la memoria y el rendimiento de programas computacionales.



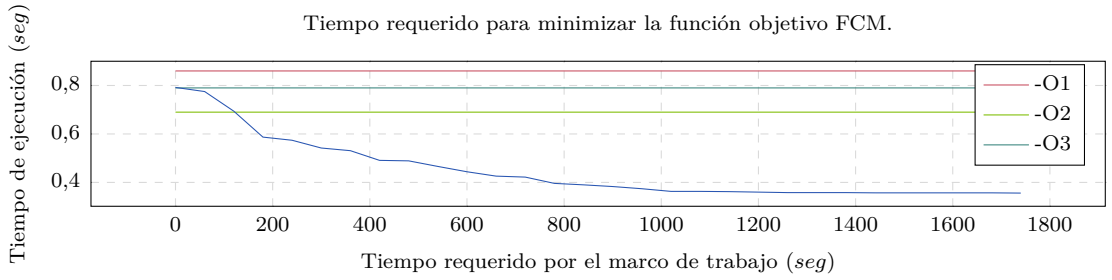
(a) Análisis de rendimiento `PSpMV`



(b) Análisis de rendimiento para el producto matricial



(c) Análisis de rendimiento para algoritmo KM



(d) Análisis de rendimiento para algoritmo FCM

Figura 5-5: Análisis del tiempo de ejecución en función del tiempo necesario por el marco de trabajo. Entre más pequeño el tiempo de ejecución sea, mejor será la aproximación obtenida. Se realizaron 30 ejecuciones y se obtuvo el tiempo de ejecución promedio en cada caso. Las líneas horizontales representan el tiempo de ejecución del algoritmo sin auto-adaptación. La línea roja representa el uso de `-O1`, la línea azul `-O3` y la línea verde a `-O2`. Los resultados demuestran que para las figuras 5-5b y 5-5c el tiempo de ejecución optimizado por las banderas no tienen ningún valor agregado dentro del proceso. El tiempo de ejecución obtenido por las banderas `-O2` en la mayoría de las ocasiones se mantiene en la media.

Tabla 5.3: Las primeras 10 banderas de `GCC` que más contribuyen en la mejora del rendimiento durante la ejecución del algoritmo `PSpMV`. La importancia se define como la ralentización cuando se elimina la bandera de la configuración final en la consola GCC. La importancia se normaliza para sumar 100 %.

Bandera	Importancia (%)
`-fno-tree-vectorize`	25.0 %
`-funroll-loops`	6.7 %
`-fno-jump-tables`	5.4 %
`-fno-inline`	5.0 %
`-fno-ipa-pure-const`	4.8 %
`-fno-wrapv`	4.7 %
`-fno-rerun-cse-after-loop`	4.5 %
`-param=max-sched-extend-regions-iters=4`	4.0 %
`-fno-tree-tail-merge`	4.0 %
`-fno-cprop-registers`	3.2 %
`264 otras banderas`	32.7 %

10 banderas que contribuyeron más a las aceleraciones obtenidas. Se normalizaron todas las ralentizaciones para sumar el 100 %. Este experimento se realizó en una sola ejecución del adaptador. Sin embargo, este experimento no se considera esencial dentro del proceso de comparación, debido a que las banderas pueden interactuar entre ellas de manera no independiente. Se observa que la desaceleración total al eliminar cada bandera de forma independiente es mayor que la desaceleración al eliminar todas las banderas.

Los resultados obtenidos en este experimento muestran un conjunto de banderas que optimizan el rendimiento del algoritmo. Para `PSpMV` muestra que el algoritmo funciona mejor para `-O2` que para `-O3` y que la estrategia en este caso es deshabilitar banderas en lugar de habilitarlas, pues las banderas tienden a interactuar pobremente con el compilador; para el Producto Matricial se alcanza un rendimiento óptimo con solo tres banderas: `-fno-exceptions`, `-fwrapv`, y `-funsafe-math-optimizations`; para FCM se observó un mejor rendimiento para `-O2` que para `-O3`. Esto es congruente con lo que se sabe de que la bandera `-O3` optimiza la velocidad, mientras `-O2` optimiza el espacio. Esto significa que `-O3` generará un ejecutable rápido, pero bastante grande, mientras que `-O2` generará un ejecutable más pequeña que podría ser más lento.

En este sentido, se hace mención que la eficiencia del espacio y el tiempo suele ser una compensación. Los algoritmos más rápidos tienden a ocupar más espacio, donde los algoritmos *in situ* [4] tienden a ser menos eficientes. Por lo general, las unidades de procesamiento modernas tienen mucho espacio de memoria, por lo que generalmente es preferible `-O3`. Sin embargo, si está programando para algo con poca memoria RAM (como un dispositivo pequeño), es posible que prefiera `-O2`.

[4]Un algoritmo *in situ* se define como un algoritmo que no aumentan el uso del espacio

Tabla 5.4: Las primeras 10 banderas de `GCC` que más contribuyen en la mejora del rendimiento durante la ejecución del algoritmo relacional al producto matricial. La importancia se define como la ralentización cuando se elimina la bandera de la configuración final en la consola GCC. La importancia se normaliza para sumar 100 %.

Bandera	Importancia (%)
`-fno-exceptions`	28.5 %
`-fwrapv`	27.8 %
`-funsafe-math-optimizations`	27.0 %
`-param=large-stack-frame=65`	1.1 %
`-fschedule-insns2`	1.0 %
`-funroll-loops`	0.9 %
`-fno-ivopts`	0.8 %
`-param=sccvn-max-scc-size=2995`	0.7 %
`-param=max-sched-extend-regions-iters=2`	0.6 %
`-param=slp-max-insns-in-bb=1786`	0.6 %
`272 otras banderas`	10.9 %

Tabla 5.5: Las primeras 10 banderas de `GCC` que más contribuyen en la mejora del rendimiento durante la ejecución del algoritmo KM. La importancia se define como la ralentización cuando se elimina la bandera de la configuración final en la consola GCC. La importancia se normaliza para sumar 100 %.

Bandera	Importancia (%)
`-freirder.blocks.and-partitions`	21.1 %
`-fwrapv`	9.4 %
`-funsafe-math-optimizations`	3.9 %
`-funroll-all-loops`	2.8 %
`-frename-registers`	2.4 %
`-fwhole-program`	2.3 %
`-param=selsched-insns-to-rename=3`	2.2 %
`-fno-tree-dominator-opts`	2.1 %
`-param=min-crossjump-insns=17`	2.0 %
`-param=max-sched-extend-regions-iters=2`	1.8 %
`265 otras banderas`	49.9 %

Tabla 5.6: Las primeras 10 banderas de `GCC` que más contribuyen en la mejora del rendimiento durante la ejecución del algoritmo FCM. La importancia se define como la ralentización cuando se elimina la bandera de la configuración final en la consola GCC. La importancia se normaliza para sumar 100 %.

Bandera	Importancia (%)
`-fno-exceptions`	25.8 %
`-fwrapv`	24.9 %
`-funsafe-math-optimizations`	16.7 %
`-funroll-all-loops`	12.4 %
`-fno-rerun-cse-after-loop`	3.5 %
`-fstrict-overflow`	0.8 %
`-fno-ivopts`	0.8 %
`-fschedule-insns2`	0.7 %
`-param=max-sched-extend-regions-iters=2`	0.3 %
`-param=selsched-insns-to-rename=2`	0.2 %
`272 otras banderas`	13.9 %

Se observa que en tres de los experimentos (ver tablas 5.4, 5.5, y 5.6) tiene relevancia `-fwrapv` ocupando el segundo lugar en todos los casos. Esta bandera asume que el desbordamiento aritmético con signo de operaciones como la suma, resta y multiplicación se ajusta utilizando la representación en complemento a dos.

Otra bandera es `-funroll-all-loops`, la cual se enfoca en el desenrollado de un ciclo y hace que el compilador prediga el número exacto de iteraciones que este ciclo será ejecutado en tiempo de compilación (o al menos establezca un límite superior). Si la predicción es correcta, entonces la ejecución se optimiza. Sin embargo, cuando el tamaño de los datos es variable en cada iteración, la bandera no tiene ningún efecto en el rendimiento del algoritmo. Por lo tanto, en este experimento se observó que el uso de esta bandera tiene más beneficios para ciclos con pocas instrucciones que no necesitan repetirse un elevado número de veces. Sin embargo, su uso depende de la comparación entre los tiempos de ejecución.

High Performance Linpack `HPL`

High Performance Linpack (`HPL`) representa un criterio estándar de evaluación para el *TOP500 supercomputer sites* [61]. Su objetivo es evaluar el rendimiento que tienen las operaciones de punto flotante en arquitecturas que van desde pequeños multiprocesadores hasta arquitecturas a gran escala. Para lograrlo, `HPL` evalúa la velocidad de resolución de un sistema de ecuaciones a través del algoritmo de factorización LU mediante la búsqueda exhaustiva de un conjunto de parámetros proporcionados por el usuario. Para alcanzar el rendimiento óptimo se deben ajustar al menos quince parámetros que incluyen tanto los tamaños de los bloques de las matrices a analizar, como los parámetros algorítmicos.

En este experimento se describe la comparación del rendimiento de la ejecución de `HPL` en conjunto con la biblioteca `Intel Math Kernel Library` (MKL v11.0 [5]) contra el marco de trabajo propuesto en conjunto con la biblioteca de operaciones elementales descrita en la sección 3.5. Cabe señalar que para este experimento se ha considerado una arquitectura basada en un procesador Xeon E5-1603 v3 con 4 núcleos, compilado con `gcc 7.5.0-3ubuntu1` y un sistema operativo Ubuntu 18.04 a 2.08 GHz. Se ha seleccionado un conjunto de parámetros de ajuste de entrada para `HPL`, sin considerar ningún conocimiento específico de la computadora donde se realizará el experimento. Para la mayoría de los parámetros, se utilizaron los parámetros `Enum` y `Switch`, ya que generalmente representan opciones discretas en el algoritmo utilizado. El parámetro principal que controla el rendimiento es el tamaño de bloque de la matriz; esto se representó como un parámetro de tipo `Integer` para dar la mayor libertad posible al adaptador.

La figura 5-6 muestra la relación que existe entre el tiempo de ejecución en función del tiempo necesario por el marco de trabajo para la auto-adaptación de parámetros. Se añade el tiempo computacional utilizado por `HPL` optimizado con `KML`. Los resultados muestran que después de 1400 segundos ($\approx$ 30 minutos) de ejecutar el marco de trabajo de auto-adaptación se puede mejorar el rendimiento de los parámetros optimizados por `HPL`.

Para este caso, el marco de trabajo propuesto es capaz de obtener un rendimiento del 82.5 % del rendimiento máximo teórico en esta computadora, mientras se explora un subconjunto pequeño del dominio. El tamaño de bloque óptimo no es una potencia de dos, y generalmente es un valor poco probable que se adivine para su uso en la sintonización manual.

Finalmente, en la tabla 5.7 se describe el conjunto de banderas que mejoran el tiempo de ejecución. Se observa que tiene mayor aportación las banderas basadas en `-funsafe-math`. La primera de ellas permite realizar optimizaciones para aritmética de punto flotante que asume que los argumentos y resultados son distintos de valores tipo `NAN`. Por otro lado, la bandera `-fwrapv` asume que el desbordamiento aritmético con signo de suma, resta y multiplicación se ajusta utilizando la representación complemento a 2.

`PetaBricks`

`PetaBricks` es un compilador de lenguaje paralelo que proporciona un marco de trabajo para que el programador describa múltiples formas de resolver un problema mientras permite al adaptador determinar qué combinación de parámetros es la mejor para cumplir el objetivo del usuario [7]. `PetaBricks` es considerado como un optimizador de bajo nivel y se compone de un conjunto de selectores algorítmicos orientados a generar instancias de poli-algoritmos a partir de elecciones definidas. Cada selector se utiliza para asignar tamaños de entrada a cada una de las opciones algorítmicas a través de un conjunto de parámetros. Por ejemplo, el selector `[InsertionSort;`

[5] MKL v11.0 es una libreria de optimización de operaciones matemáticas y se encuentra disponible en `http://software.intel.com/en-us/articles/intel-math-kernel-linpack-download`

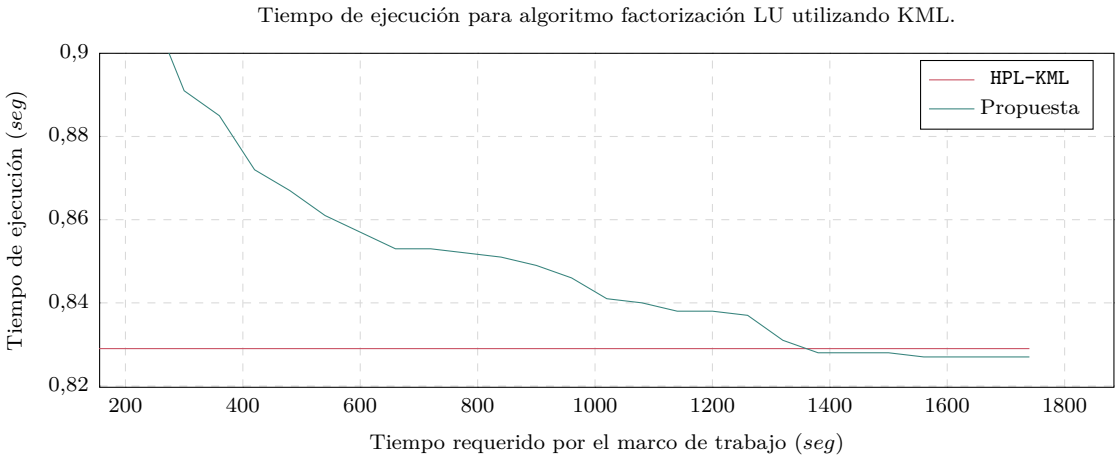


Figura 5-6: *High Performance Linpack* Análisis del tiempo de ejecución en función del tiempo necesario por el marco de trabajo para **factorización LU** en una arquitectura multicore. La línea roja representa el tiempo de ejecución óptimo con los parámetros recomendados por Intel. Para este experimento, se han realizado 30 ejecuciones y se ha obtenido el tiempo de ejecución promedio en cada caso. **Para las pruebas se utilizó un procesador Xeon E5-1603 v3 con 4 núcleos en Ubuntu 18.04 a 2.80 GHz y versión GCC 7.5.0-3ubuntu1**.

Tabla 5.7: Las primeras 10 banderas de `GCC` que más contribuyen en la mejora del rendimiento durante la ejecución del algoritmo de factorización LU con HPL y KML. La importancia se define como la ralentización cuando se elimina la bandera de la configuración final en la consola GCC. La importancia se normaliza para sumar 100 %.

Bandera	Importancia (%)
`-funsafe-math.optimizations` [6]	21.1 %
`-ffinite-math-only`	19.7 %
`-fwrapv`	9.4 %
`-frename-registers`	7.6 %
`-ffinite-loops`	3.9 %
`-funroll-all-loops=6`	2.8 %
`-param=selsched-insns-to-rename=2`	2.4 %
`-freorder-blocks-and-partition=4`	2.3 %
`-param=max-sched-extend-regions-iters=2`	2.1 %
`-param=slp-max-insns-in-bb=1249`	0.2 %
`-param=sms-loop-average-count-threshold=2`	0.1 %
`223 otras banderas`	6.6 %

500; ordenación rápida; 1000; MergeSort] que correspondería a sintetizar el código 5.3.

Código 5.3: Función para sintetizar por el autoadaptador.

```
void Sort(List& list){
  if(list.length < 500)
    InsertionSort(list);
  else if(list.lenght < 1000)
    QuickSort(list);
  else
    MergeSort(list);
}
```

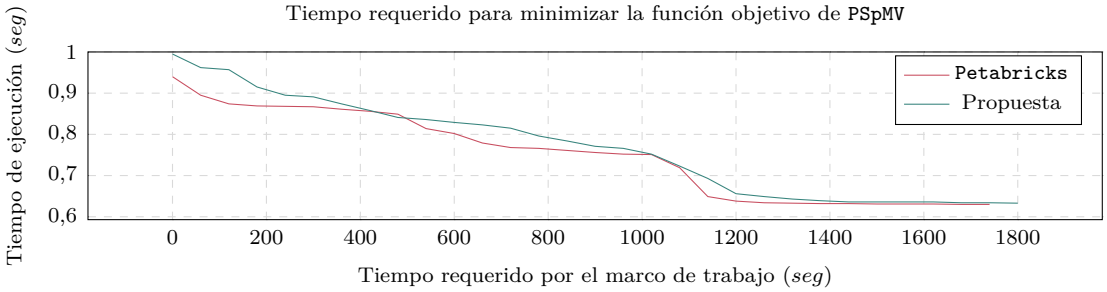
El código 5.3 realiza la instancia de los algoritmos `QuickSort` y `MergeSort` a la función `Sort` con el objetivo de que el compilador intercambie dinámicamente entre ambos métodos a medida que se realizan llamadas recursivas a objetos tipo `list` cada vez más pequeños. Este proceso se realiza de manera recursiva. Los programas de `PetaBrick` contienen muchos selectores algorítmicos y otros tipos de parámetros, como tamaños de bloque, recuentos de hilos, recuentos de iteraciones y parámetros específicos del programa.

En la figura 5-7 se describen los resultados obtenidos con el marco de trabajo en comparación con la auto-adaptación que realiza `PetaBrick`. Cabe mencionar que `PetaBricks` utiliza una estrategia diferente que comienza con pruebas en entradas de problemas muy pequeños y trabaja incrementalmente hasta entradas de tamaño completo [8]. En todos los casos, ambos adaptadores llegan a soluciones similares, y para `PSpMV`, tanto `PetaBricks` como el marco de trabajo aquí propuesto obtienen la misma solución. Para KM y FCM, el marco de trabajo propuesto supera a `PetaBricks`. Se observa que en la mayoría de los casos hay un intervalo en el que `PetaBricks` obtiene mejores resultados que el marco de trabajo, sin embargo, el marco de trabajo tiene una convergencia más rápida a un valor óptimo.

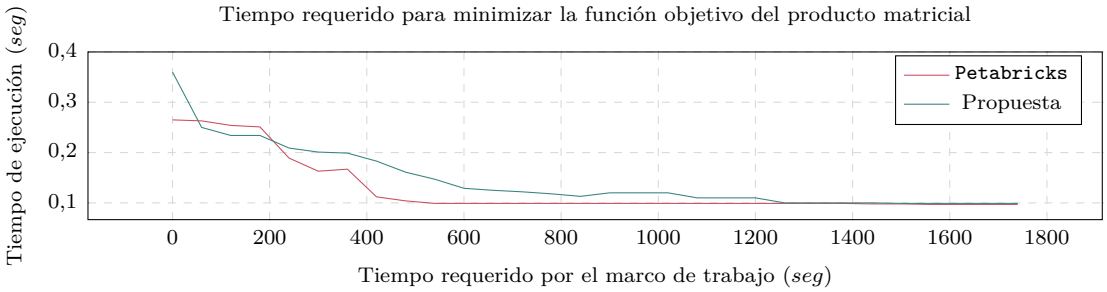
En esta sección se ha analizado el rendimiento del marco de trabajo a través de la comparación del tiempo de ejecución que requieren tres enfoques diferentes (`GCC/G++ flags`, `HPL` y `PetaBricks`). Los resultados demuestran que el enfoque puede llegar a obtener parámetros competitivos con los marcos de trabajo ya propuestos. En la sección 5.6 se analiza la eficiencia del marco de trabajo al aplicarlo en el análisis de datos bioinformáticos, específicamente en el análisis de expresión genética.

5.6. Análisis bioinformático utilizando técnicas de auto-adaptación

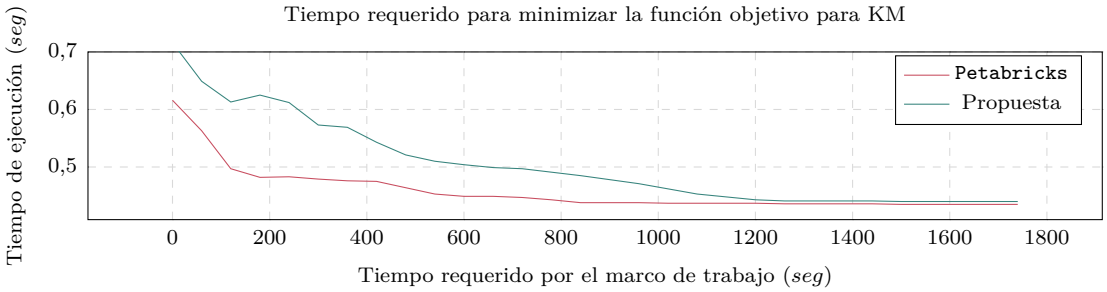
En esta sección, se describe la metodología utilizada durante la evaluación de la eficiencia del marco de trabajo aplicado en el algoritmo de consenso distribuido descrito en la sección 3.4. El objetivo está orientado al descubrimiento de agrupaciones biológicamente significativas a partir de matrices de expresión genética. A pesar de que este trabajo se enfoca en algoritmos como KM, FCM y EM, la metodología puede



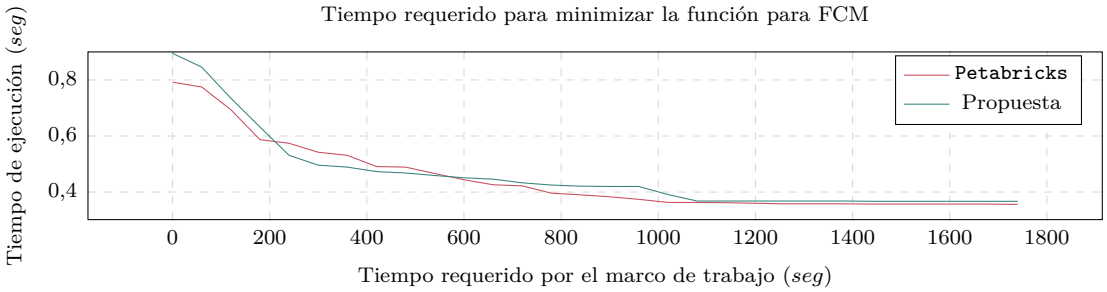
(a) Análisis de rendimiento PSpMV



(b) Análisis de rendimiento para el producto matricial



(c) Análisis de rendimiento KM



(d) Análisis de rendimiento FCM

Figura 5-7: `PetaBricks` Análisis del tiempo de ejecución en función del tiempo necesario por el marco de trabajo. Entre más pequeño el tiempo de ejecución sea, mejor será la aproximación obtenida por el marco de trabajo. Para este experimento, se han realizado 30 ejecuciones y se ha obtenido el tiempo de ejecución promedio en cada caso. Se observa de que a pesar la propuesta de este trabajo en la primera evaluación comienza con un valor relativamente más grande en comparación con `Petabricks`, en al menos 1200*seg*, la propuesta de este trabajo logra igualar o mejorar el valor que obtiene `Petabricks`.

ser considerada como un enfoque general para cualquier algoritmo de agrupamiento particional descrito en el estado del arte.

Durante este análisis de expresión genética se deben considerar dos cuestiones para cuantificar la eficacia del algoritmo: (1) La forma en la que el usuario determina el número de grupos; y (2) El nivel de confianza en la que el usuario evalúa la asignación de patrones a cada uno de los grupos. Lo anterior se debe a que la solución de un problema relativamente pequeño (≈ 40 genes) se ve afectada por la alta dimensionalidad de los datos, la sensibilidad al ruido y un posible ajuste excesivo.

Se evalúa el consenso de múltiples ejecuciones de uno o más algoritmos de agrupamiento particional en diferentes unidades de procesamiento sobre un subconjunto de datos. Se describen los resultados obtenidos de los experimentos considerando la sensibilidad del marco de trabajo a las condiciones iniciales definidas por la elección de los centroides $m_k \forall k = 1, \ldots, K$. Para evaluar la efectividad del marco de trabajo se abordan conjuntos de datos de expresión genética construidos *in silico*. Finalmente, se analizan conjuntos de datos propuestos en la literatura así como también se analiza un conjunto de datos relacionado al desarrollo de hepatocarcinomas en ratones. Estos datos han sido obtenidos a partir del análisis de microarreglos (ver sección 2.6).

5.6.1. Metodología

La motivación de la metodología propuesta es la necesidad de evaluar: (1) La «estabilidad» del conjunto de grupos; y (2) El «rendimiento» del marco de trabajo propuesto en la ejecución de los algoritmos en una arquitectura. La primera se refiere al supuesto de que el método es intuitivamente simple, es decir, si los datos representan una muestra de elementos extraídos de distintas subpoblaciones, y si se tuviera que observar una muestra diferente extraída de las mismas subpoblaciones, la composición y el número de subgrupos inducidos no deberían ser radicalmente diferentes. Cuanto más robustos sean los grupos a la variabilidad del muestreo, se puede asegurar que los resultados representan una estructura real. El segundo está orientado a la evaluación del tiempo de cómputo necesario en la ejecución de estos algoritmos y verificar si el marco de trabajo de auto-adaptación es capaz de optimizar algunos parámetros para mejorar el rendimiento sin perder eficacia en los resultados.

Se realizaron permutaciones en los datos originales, las cuales se simularon mediante técnicas de remuestreo. Cuando se utiliza el algoritmo distribuido de agrupamiento particional, se distribuye al menos el 70 % de los patrones en un número específico de nodos. Enseguida se aplica el algoritmo de agrupamiento. De esta forma, cada nodo evalúa la función objetivo para llegar a un resultado, el cual será verificado por un nodo líder encargado de llegar a un consenso. Sin embargo, si no se tiene una arquitectura distribuida, se procede de la siguiente manera: se realizan diferentes ejecuciones y en cada una de ellas se almacena el resultado «temporal». Al finalizar se llega a un consenso sobre todos los resultados temporales de cada ejecución.

5.6.2. Evaluación experimental

La metodología propuesta en la sección 5.6.1 ha sido utilizada en cuatro conjuntos de datos reales construidos *in-vitro* y seis conjuntos simulados *in-silico*. En esta sección se resume la metodología de evaluación y las métricas utilizadas. También se describe el conjunto de datos reales y simulados. Finalmente, se reportan y discuten los resultados obtenidos.

Métrica de evaluación: Indice Rand

Para evaluar la composición de cada grupo en el análisis de expresión genética de microarreglos se propone la cuantificación de similitud entre dos grupos con base en el fenotipo conocido. En este caso, considere que si el número de grupos (K) es correcto en todos los casos, es decir, igual al número conocido de clases, entonces se define la «tasa de error de clasificación» como una métrica de evaluación. Por lo tanto, se requiere establecer un mapeo entre las etiquetas de clase definidas por el algoritmo (etiqueta relacionada a cada grupo) y las etiquetas de clase ya conocidas. Enseguida, se interpreta la asignación de grupo como una tarea de clasificación convencional para finalmente medir la tasa de error. Sin embargo, la tasa de error es difícil de interpretar cuando el valor de K se desconoce.

Se propone el uso del índice de Rand ($I_r(i,j)$) como métrica de similitud entre el i-ésimo y j-ésimo grupo, incluso cuando se consideran grupos con diferentes números de patrones. El índice Rand es independiente del modelo, lo que permite su adopción a los diferentes algoritmos de agrupamiento y ha sido ampliamente utilizado en [17,64,77], demostrando tener un rendimiento aceptable. Cabe mencionar que el dominio del índice oscila entre 0 y 1. Por lo tanto, considere dos grupos c_a y c_b. Si $I_r(c_a, c_b) = 1$ el valor de similitud es perfecto. En el estado del arte se ha demostrado que el índice Rand tiene un valor esperado de 0 para dos grupos aleatorios, bajo el supuesto de una distribución hipergeométrica para el modelo de aleatoriedad.

Diseño experimental

Los detalles de implementación del marco de trabajo se describen a continuación.

- Se aplicó la metodología paralela de agrupamiento descrita en el capítulo 3. El resultado de esta fase incluye una estimación del número de grupos K y, para un valor K dado, una asignación de patrones, los cuales son evaluados con el índice Rand. Cuando el número estimado de grupos K es diferente del número conocido K_{real}, se informa el valor del índice Rand. tanto para K como para K_{real}. Este último proporciona información adicional sobre qué tan bueno es el proceso de agrupación respetando los límites de clase conocidos

- Se evaluó la asignación de patrones a cada grupo con base en la aplicación de KM y FCM utilizando la métrica de distancia euclidiana.

- Para evaluar la partición inducida por el algoritmo de agrupamiento, se realizó la comparación con el índice Rand producido por un clasificador Naïve-Bayes [97]

entrenado por el mismo conjunto de datos mediante validación cruzada sin incluir un patrón. Es importante destacar que la formación de Naïve-Bayes se basa en las muestras supervisadas (es decir, las muestras que incluyen las etiquetas de clase). La tasa de error de Naïve-Bayes da una indicación de la precisión de agrupamiento

Configuración de las técnicas de agrupamiento

Se aplica el marco de trabajo a un conjunto de datos considerando el pseudocódigo 2 descrito en la sección 3.4. El marco de trabajo requiere que el usuario especifique el algoritmo de agrupación a utilizar, así como el diseño del adaptador de los parámetros a optimizar dentro de la función objetivo. Para ello, se exploró el uso de dos algoritmos: (1) KM; y (2) FCM. Para cada K se realizó la ejecución de $T = 200$ iteraciones y $t_{\text{máx}} = 200$ y $\epsilon = 0{,}0001$ para los algoritmos de consenso. En cada iteración, el conjunto de datos permutados se obtiene muestreando, sin reemplazo, el 70 % de los elementos de los datos originales. Los resultados producidos por el marco de trabajo incluyen: (1) La estimación del número de grupos; y (2) La asignación de los patrones a los grupos correspondientes. Ambos resultados son evaluados con el índice Rand descrito en la sección 5.6.2.

5.6.3. Conjunto de datos objetivo

En esta sección, se describen los conjuntos de datos utilizados durante este análisis (Ver tabla 5.8), junto con algunas de sus características relevantes, como el número de clases, el número de genes y el número de condiciones experimentales. Los primeros seis conjuntos de datos representan datos simulados, mientras que los últimos cuatro representan datos de microarreglos de expresión genética. Cabe mencionar que el conjunto de datos `Leukemia`, `CNS tumors` y `Normal tissues` ya han sido utilizados en estudios anteriores dentro del estado del arte. Sin embargo, el conjunto de datos descrito por la línea azul en la tabla 5.8 representa el conjunto de datos relacionados al hepatocarcinoma en ratones.

En primera instancia se analiza el conjunto de datos simulados, seguido por el conjunto de datos derivados de microarreglos. Finalmente, se dedica una sección al análisis exploratorio de hepatocarcinomas.

Datos simulados

Se generaron seis conjuntos de datos construidos *in-silico* para entender como funciona la metodología propuesta específicamente en: (1) Conjuntos de datos que carecen de clases definidas y cuya distribución de valores no varía; (2) Conjuntos de datos con clases bien definidas cuyos valores siguen la misma distribución; (3) Conjuntos de datos cuya distribución de valores con presencia de marcadores de características de diferente intensidad. Se describe cada uno de los conjuntos de datos.

`DUniforme` y `DGaussiana1` son dos conjuntos de datos que fueron utilizados para evaluar el comportamiento del marco de trabajo en tareas de agrupamiento cuando se

Tabla 5.8: Descripción de los conjuntos de datos construidos *in-silico* y *in-vitro* utilizado en la evaluación experimental. Los primeros seis conjuntos de datos representan datos simulados, mientras que los últimos cuatro representan datos de microarrays de expresión genética. Cabe mencionar que el conjunto de datos `Leukemia`, `CNS tumors` y `Normal tissues` ya han sido utilizados en estudios anteriores dentro del estado del arte. Sin embargo, hepatocarcinoma se describe un análisis estadístico de datos de manera exploratoria.

Conjunto de datos	No. de clases	No. de muestras	No. de características
`DUniforme`	1	60	600
`DGaussiana1`	1	60	600
`DGaussiana2`	3	60	600
`DGaussiana3`	4	400	2
`DGaussiana4`	5	500	2
`Simulados1`	6	60	600
`Leukemia`	3	38	999
`CNS tumors`	5	103	1000
`Normal tissues`	13	99	1000
`hepatocarcinoma`	•	•	•

aplica a datos que contienen subpoblaciones similares. Se consideraron estas distribuciones debido a que son totalmente diferentes. Estas matrices contienen 60 muestras y 600 características.

`DGaussiana2` es el tercer conjunto de datos que contiene 60 muestras y 3 grupos. Cada grupo contiene 200 características diferentes del conjunto de las 600 asignadas a cada agrupación. Los datos simulan un patrón mediante el cual un conjunto distinto de 200 genes se sobreexpresan en uno de los tres grupos y se inhiben en los dos grupos restantes.

`DGaussiana3` y `DGaussiana4` representan una mezcla de modelos Gaussianos. En particular, *DGaussiana3* representa la unión de observaciones de cuatro funciones gaussianas bivariadas con el mismo valor en la diagonal de la matriz de covarianza $\sigma = 0{,}25$ y centrada en las cuatro esquinas de un cuadrado con longitud de lado $\lambda = 2$. Se generaron 200 muestras de las cuales se asignaron 50 muestras a cada clase. De manera similar, `DGaussiana4` representa la unión de 5 funciones gaussianas bivariadas, 4 de los cuales están centrados en las esquinas del cuadrado de longitud $\lambda = 2$ con el mientras que la quinta función gaussiana está centrada en $(\frac{\lambda}{2}, \frac{\lambda}{2})$. Se generaron un total de 250 muestras, de las cuales 50 muestras distintas fueron asignadas a cada clase. Se utilizaron dos valores de $\lambda = \{2, 3\}$, para investigar diferentes niveles de superposición entre grupos.

Por último, se consideró un conjunto de datos con grupos de tamaño diferente y marcadores genéticos de diferente intensidad. `Simulados1` consta de un conjunto de datos de 600 genes y 60 muestras. Este conjunto de datos se divide en 6 clases con

Tabla 5.9: Descripción de los conjuntos de datos construidos *in-vitro*. Los primeros tres conjuntos ya han sido descritos en la literatura, sin embargo, del último conjunto de datos se describe un análisis exploratorio de datos. En los artículos de referencia se pueden encontrar más detalles biológicos.

Conjunto de datos	Descripción
`Leukemia`	Muestras de tejido relacionado a médula ósea obtenidas de pacientes con leucemia aguda en el momento del diagnóstico, de las cuales 11 muestras están relacionadas a Leucemia Mieloide Aguda (AML); 8 muestras a Leucemia Linfoblástica Aguda (LLA) de linaje T; y 19 muestras a leucemia de linaje tipo B (ALL)
`CNS tumors`	Muestras relacionadas a tumores embrionarios del Sistema Nervioso Central (SNC), de las cuales 10 están relacionadas a meduloblastomas (MD); 8 a tumores neuroectodérmicos primitivos (PNET); 10 tumores teratoides/rabdoides atípicos (Rhab); 10 a gliomas malignos (Glio); y 4 a cerebelos normales (Ncer).
`Normal tissues`	Muestras relacionadas a 13 tipos de tejidos distintos de los cuales, 5 están relacionados a mama, 9 a próstata, 7 a pulmón, 11 a colon, 6 a células del centro germinal, 7 a vejiga, 6 a útero, 5 a monocitos de sangre periférica, 12 a riñón, 10 a páncreas, 14 a ovario, 5 a cerebro completo y 3 a cerebelo.
`hepatocarcinoma`	Muestras relacionadas al desarrollo evolutivo del hepatocarcinoma en ratones de laboratorio.

8, 12, 10, 15, 5 y 10 muestras respectivamente. Cada clase está compuesta por 50 genes distintos sobreexpresados para esa clase. Se incluyen 300 genes de ruido. Los genes de los diferentes grupos tienen diferente nivel de expresión genética. Es decir, los primeros 50 genes que marcan la primera clase son las que se sobreexpresan, con mayor expresión diferencial y menor variación, seguidos de los 50 genes del segundo grupo, y así sucesivamente. Considere que λ representa la distancia máxima de un patrón al centro de la función Gaussiana.

Datos de microarreglos de expresión genética

Se ofrece una descripción de los conjuntos de datos de expresión genética construidos *in-vitro* en la tabla 5.9. La mayoría de los datos se procesaron en *Human Genome U95* `Affymetrix`© *microarrays*, sin embargo, el conjunto de datos `leukemia` ha sido procesado en *Human Genome HU6800* `Affymetrix`© *microarrays*. En los artículos de referencia se pueden encontrar más detalles biológicos sobre estos conjuntos de datos ([148], [221] y [202] respectivamente).

Para asegurar que el fenotipo conocido para un conjunto de datos sea el resultado dominante en los datos, se proyectó el conjunto de datos en el espacio de marcadores genéticos para el fenotipo. Lo anterior define a la información del fenotipo como el índice Rand, el cual ayudará al método de agrupamiento. Se utilizó una relación señal-ruido (SNR) para clasificar los genes [203]. El resultado se obtiene seleccionando los genes que tienen un nivel de expresión genética máximo en cada una de las clases, donde el número exacto depende del conjunto de datos. En particular, para un conjunto de datos con K grupos, se seleccionan los n genes sobreexpresados. Considere que el número n de marcadores genéticos depende de cuántos genes se expresan diferencialmente con un nivel de significancia suficientemente alto (0,05) según lo determinado por la prueba de permutación con base en lo descrito en [203].

5.6.4. Resultados

En esta sección se describen los resultados del análisis de eficacia del marco de trabajo. En primera instancia se describen los resultados relacionados a los seis conjuntos de datos construidos *in-silico*. Los resultados son prometedores con relación a los valores obtenidos por el índice Rand. Finalmente, se describen los resultados obtenidos en los datos construidos *in-vitro*. Se ha dedicado una sección específica para el análisis del conjunto de datos `hepatocarcinoma`.

Datos simulados

Los resultados demuestran que la metodología es invariante al tipo de datos que se estén utilizando. En general, se puede obtener el número correcto de grupos incluidos en cada experimento y clasificar al menos el 95 % de los patrones de forma correcta. Los elementos que no han sido clasificados con la etiqueta de clase predefinida durante la construcción *in silico* son aquellos que se encuentran en una región superpuesta entre grupos. Para estos casos, podría ayudar la aplicación de técnicas de biagrupamiento.

La tabla 5.10 describe el número de grupos obtenido para cada uno de los conjuntos de datos. Se aprecia que para el conjunto de datos `Gaussiana4` con $\lambda = 3$ la metodología recupera el número de grupos de manera correcta, sin embargo, cuando $\lambda = 2$ la metodología no funciona de manera correcta. En este caso, el 32 % de los patrones generados se encuentran más cerca del centro de la función Gaussiana, lo cual hace más dificil la detección de los 5 grupos.

En la tabla 5.11 se describe el índice Rand para cada uno de los conjuntos. Los resultados demuestran que para el conjunto de datos`DGaussiana2`, el índice Rand es ~ 1 en todos los casos. Esto es debido a que las clases están bien definidas y no contienen ningún tipo de variación ni error introducido de manera intencional. Una de estas clases se sobre-expresa, y otra se inhibe. Cabe mencionar que en la mayoría de los casos, la similitud entre grupos es perfecto, sin embargo, en el conjunto de datos `DGaussiana4` con $\lambda = 2$ representa un problema debido a que existe mayor sobrelape entre los patrones. Este resulta ser un problema inherente en las tareas de agrupamiento.

Tabla 5.10: Número estimado de conglomerados por el marco de trabajo y por el índice Rand, en combinación con FCM y KM. En esta tabla se describe su aplicación a datos simulados. De manera general, se puede obtener el número correcto de grupos incluidos en cada experimento.

Conjunto de datos	K_{real}	CC_{KM}	CC_{FCM}	$Rand_{KM}$	$Rand_{FCM}$
`DUniforme`	1	1	1	1	1
`DGaussiana1`	1	1	1	3	1
`DGaussiana2`	3	3	3	3	3
`DGaussiana3`	4	4	4	1	1
`DGaussiana4` ($\lambda = 3$)	5	5	5	1	1
`DGaussiana4` ($\lambda = 2$)	5	3	2	1	1
`Simulados1`	6	7	6	7	3

Tabla 5.11: Índice de Rand ajustado para NB, KM, agrupación de consenso con KM (CC_{KM}) y agrupación de consenso con FCM (CC_{FCM}). Se observa que el Índice Rand es menor para el conjunto de datos `DGaussiana4` con $\lambda = 2$.

Conjunto de datos	NB	KM	CC_{KM}	CC_{FCM}
`DUniforme`	•	1.000	1.000	1.000
`DGaussiana1`	1.000	1.000	0.849	1.000
`DGaussiana2`	1.000	1.000	1.000	1.000
`DGaussiana3`	0.896	0.813	0.917	0.898
`DGaussiana4` ($\lambda = 3$)	0.911	0.892	0.932	0.941
`DGaussiana4` ($\lambda = 2$)	0.567	0.422	0.589	0.592
`Simulados1`	0.906	0.986	0.986	0.986

Tabla 5.12: Número estimado de conglomerados por el marco de trabajo y por el índice Rand, en combinación con FCM y KM. En esta tabla se describe su aplicación a datos derivados de microarreglos. De manera general, se puede obtener un número similar de grupos incluidos en cada experimento entre los algoritmos utilizados.

Conjunto de datos	K_{real}	CC_{KM}	CC_{FCM}	$Rand_{KM}$	$Rand_{FCM}$
`Leukemia`	3	5	4	5	5
`CNS tumors`	5	5	5	6	4
`Normal tissues`	13	7	8	6	7

Datos de microarreglos de expresión genética

En general, cuando se analizan datos de expresión genética con el algoritmo FCM optimizado mediante el marco de trabajo se concluye que FCM supera a la implementación con KM con base en los resultados obtenidos. En los conjuntos de datos donde el número de clases es relativamente pequeño se pudo determinar el número correcto de agrupaciones (K) y establecer la función de pertenencia con un alto nivel de precisión, sin embargo, esto no sucede para conjuntos de datos con un número grande de clases debido a que la estimación del número de grupos y la asignación de patrones a cada centroide fue más difícil en el conjunto de datos `CNS tumors`. Esto también puede deberse a que el tamaño de muestra es relativamente pequeño para el número dado de clases.

El análisis del conjunto de datos `Leukemia` utilizando FCM selecciona 4 agrupaciones como su número óptimo, dos de las agrupaciones corresponden exactamente a las condiciones LMA y LLA, sin embargo, la clase ALL se subdivide en dos grupos más con algunos elementos de otras clases. Es interesante notar que el índice Rand también selecciona 5 como el número óptimo de grupos. Esto puede generarse por los sobrelapamientos que existen entre los datos. En el estado del arte, es ampliamente aceptado que la clase LLA (Leucemia Linfoblástica Aguda) pueda dividirse en subclases biológicamente significativas, aunque la composición y naturaleza de estas subclases no son tan bien aceptadas. Por lo tanto, es posible que la estructura de la subclase que se encontró dentro de ALL refleje una distinción biológicamente significativa.

La aplicación del marco de trabajo al conjunto de datos `CNS tumors` permite encontrar 6 grupos como número máximo. Cuatro de los seis grupos corresponden a Rhab, MD y PNET, SNC. Todos los errores en la asignación de grupos involucran muestras de SNC. La aplicación del algoritmo KM en el marco de trabajo también produce un número estimado de grupos de 5. Sin embargo, si se verifica la matriz de expresión genética, claramente con $K = 5$ o $K = 7$ el perfil se muestra limpio, con bloques diagonales claramente delimitados. Cuando $K = 7$ se identifican 4 grupos cuyos niveles de pertenencia de los elementos a su centroide es un 93 % correcta.

Por otro lado, la aplicación de todos los algoritmos en el análisis `Normal tissues`

Tabla 5.13: Índice de Rand ajustado para NB, KM, agrupación de consenso con KM (CC_{KM}) y agrupación de consenso con FCM (CC_{FCM}).

Conjunto de datos	NB	KM	CC_{KM}	CC_{FCM}
`Leukemia`	1.00	0.64(0.46)	0.64(1.0)	0.72(0.6)
`CNS tumors`	0.63	0.62	0.54	0.42
`Normal tissues`	0.65	0.45(0.57)	0.45(0.57)	0.21(0.48)
`Hepatocarcinomes`	0.94	0.37(0.28)	0.31(0.28)	0.23(0.22)

no devolvió el número correcto de agrupaciones. Sin embargo, debe quedar claro que, dado el número elevado de $K = 13$ y el pequeño número de muestras por grupo (90 muestras), la esperanza de recuperar una distinción de 13 clases es bastante escasa. Vale la pena señalar que la mayoría de los grupos descubiertos corresponden a uniones bastante limpias de tipos conocidos. La aplicación del algoritmo `FCM` estima un número menor de grupos. La inspección de la matriz de expresión genética sugiere no superar los 5 grupos. El valor pequeño para el índice de Rand confirma el hecho de que la mayoría de los grupos son una mezcla de varios tipos de tejidos distribuidos en más de un grupo.

5.7. Análisis bioinformático de hepatocarcinomas

El hepatocarcinoma (CHC) es la tercera causa de muerte relacionada con el cáncer. La supervivencia media y las tasas de supervivencia a 5 años de los pacientes con CHC siguen siendo bajas. Por tanto, existe una necesidad urgente de comprender los mecanismos de progresión del cáncer CHC e identificar biomarcadores útiles para predecir el pronóstico que pudiera tener un paciente [134].

En esta sección se describe el análisis de un conjunto de datos obtenidos a través del análisis de microarreglos de expresión genética asociados al desarrollo temporal de nódulos preneoplásicos [7] y los carcinomas hepatocelulares (CHC) para definir los genes implicados en la progresión del cáncer en un modelo de hepatocitos resistentes. El experimento se realizó en ratas de laboratorio y se incluyeron 8,794 genes ante 74 condiciones experimentales [177]. Este conjunto reporta el nivel de expresión genética de 4 ratas que fueron sometidas a tratamiento de CHC y fue provisto por el Dr. Saul Villa-Trevino del Departamento de Biología Celular del Centro de Investigación y de Estudios Avanzados del IPN (CINVESTAV-IPN).

La tabla 5.14 detalla las características o condiciones experimentales que describen al experimento. Se observa que durante los primeros días, solo se reporta el análisis de expresión genética en la etapa inicial. En el periodo de 1 mes a 9 meses se reporte la

[7] Un nódulo o lesión preneoplásico es una displacia de grado alto que no necesariamente es clasificada como cáncer pero si no se trata, puede conducir a el.

Tabla 5.14: Resumen de condiciones experimentales del análisis microarreglos asociados al desarrollo temporal de nódulos preneoplásicos y carcinomas hepatocelulares. Cada fila representa el tiempo en que se obtuvo la cuantificación de la expresión genética. Cada columna representa si se reportó o no la presencia de nódulos ni tumores. Estos datos fueron obtenidos a partir del análisis de microarreglos en un experimento que incluyó a cuatro ratas de laboratorio.

Tiempo	Etapa inicial	Nódulo	No nódulo	Tumor	No tumor
24 horas	•				
7 días	•				
11 días	•				
16 días	•				
1 mes		•	•		
5 meses		•	•		
9 meses		•	•	•	•
12 meses				•	•
18 meses				•	•

presencia de nódulos [8] y finalmente en un periodo mayor o igual a 9 meses se describe la presencia o ausencia de tumor [9].

La hibridación de los microarreglos se realizó a través de la plataforma de`Affymetrix` [10] obteniendo el valor único que describe el nivel de expresión genética de la transcripción de ARN de un gen específico ante una condición experimental. El objetivo de este análisis además de evaluar la eficacia y la eficiencia del marco de trabajo auto-adaptable se enfocó en la definición de correlaciones entre el genotipo (información contenida en el genoma de la rata), los niveles de expresión genética y el desarrollo de la enfermedad (fenotipo). Lo anterior permitió identificar regiones genómicas que influyen en el desarrollo de la enfermedad.

Para poder procesar el conjunto de datos se utilizó un enfoque basado en el proceso *Knowledge Discovery Data* (KDD) siguiendo la metodología que se describe a continuación:

- **Selección del conjunto de datos.** En esta fase se consideró el análisis de los 8,274 (94 %) genes y las 74 condiciones experimentales. El nivel de expresión genética en cada una de las condiciones experimentales representa el conjunto

[8] Un nódulo se define como una masa o lesión existente en un tejido

[9] A diferencia del nódulo que solo representa una lesión firme en un tejido, un tumor se define como una colección anormal de células. Este tumor puede ser maligno (cancerosa) o benigno (no cancerosa). Cuando de un tumor benigno se trata, entonces no se propaga más allá de donde están y no reaparece después de que se extirpa. Sin embargo, a veces son peligrosos dependiendo de dónde se encuentren.

[10] `Affymetrix` es una marca de productos de microarreglos que se enfoca en la investigación, desarrollo y fabricación de Biotecnología. Estos microarreglos permiten el análisis de los perfiles de expresión genética en una muestra biológica.

Tabla 5.15: Análisis estadístico de los datos relacionados al desarrollo temporal del hepatocarcinoma en ratas. Se describe el valor mínimo, el valor máximo, la mediana, la media y los principales cuartiles de cada experimento. Se aprecia que general si se presentario cambios significativos con base en la condición basal, los cuales generalmente sugieren que los genes tienden a inhibirse, lo cual podría suponer la existencia de silenciamiento genético básico. Se marca en color azul claro el experimento correspondiente a los 9 meses debido a que es en el que se pudiera observar un cambio significativo en el perfil de expresión genética en la ausencia y presencia tanto de nódulo como de tumor.

Experimento	Mín	Q_1	Q_2	Q_3	Máx	Media
Condición basal	2.807	5.287	7.149	8.609	13.368	7.047
24 horas	1.497	3.187	5.060	7.282	13.234	5.393
7 días	1.557	3.155	5.086	7.057	13.395	5.313
11 días	1.636	3.319	5.095	7.067	13.036	5.356
16 días	1.574	3.490	5.054	6.956	13.355	5.406
1 mes NN	1.668	3.273	5.054	6.949	13.363	5.318
1 mes N	1.628	3.368	5.008	6.966	13.438	5.357
5 meses NN	1.583	3.259	4.908	6.995	13.483	5.307
5 meses N	1.538	3.368	4.962	6.958	13.418	5.338
9 meses NN	1.567	4.187	5.286	6.932	13.730	5.455
9 meses N	1.559	3.474	5.035	6.859	13.386	5.351
9 meses NT	1.525	3.403	4.952	6.923	13.504	5.338
9 meses T	1.605	3.292	5.000	6.951	13.303	5.312
12 meses NT	1.642	3.314	4.989	6.996	13.780	5.338
12 meses T	1.483	3.358	4.929	6.904	13.587	5.301
18 meses NT	1.606	3.458	4.999	6.883	13.585	5.342
18 meses T	1.467	3.323	5.012	6.966	13.458	5.312

Q_1= cuartil 1, Q_3=cuartil 3, NN=no nódulo, N=nódulo, NT=no tumor, T=Tumor.

de variables independientes que sirvieron para realizar el análisis diferencial de expresión genética en los cuatro ratones que se incluyeron en el experimento. Se seleccionaron principalmente dos variables objetivo: (1) Subconjunto de genes que se comportan de manera similar en todas las condiciones experimentales; (2) Subconjunto de condiciones que tienen el mismo perfil de expresión genética en todos los genes. Sin embargo, y debido a que se quiere obtener un análisis más refinado, se incluye una tercera variable objetivo definida por el subconjunto de genes que tienen un comportamiento correlacionado o anticorrelacionado ante un subconjunto de condiciones experimentales. En este último, en lo que al marco de trabajo se refiere solo se describe la definición de banderas `GCC/G++` que optimizan el rendimiento del algoritmo de agrupamiento.

- **Pre-procesamiento y transformación.** El conjunto de datos fue procesado con la biblioteca `Bioconductor` [11] obteniendo una matriz de expresión genética $H \in \mathcal{R}^{n \times m}$, donde $n = 8,274$ genes y $m = 74$ condiciones experimentales, entre las cuales se incluye la condición basal [12] del experimento y el identificador, según `Affymetrix`, de cada uno de los genes incluidos en el experimento.

  En la tabla 5.15 se describe un análisis estadístico de los datos. Se describe el valor mínimo, el valor máximo, la mediana, la media y los principales cuartiles de cada experimento. En general, se aprecia que los valores comparados con la condición basal tienen a disminuir y por tanto se puede decir que existe la presencia de silenciamiento genético básico, el cual se describe como un sistema de inactivación de genes para la producción de proteínas estructurales.

- **Proceso de MD.** En esta etapa se realizó el análisis diferencial de expresión genética. Sin embargo, antes de su descripción se presenta en la figura 5-8 la matriz de expresión genética que contiene el conjunto total de datos. Las filas de la matriz representan los genes involucrados en el experimento, mientras que las columnas representan cada una de las condiciones experimentales. Como se aprecia, es difícil encontrar agrupaciones de genes o condiciones que tengan un comportamiento correlacionado ingenuamente. Esto hace evidente la aplicación de la metodología que aquí se describe.

  En la figura 5-9 se describe el perfil de expresión genética de los genes cuando son sometidos a cada una de las 16 condiciones experimentales seleccionadas en el experimento. En el eje vertical se representan cada uno de los genes, mientras que el eje horizontal describe el perfil de expresión de cada uno de los genes. Cada condición representa la evolución temporal de la patología. Esta imagen confirma el silenciamiento genético básico, el cual se define como un sistema de inactivación de genes mediado por la molécula de ARN descrita en el capítulo 2.

[11] `Bioconductor` es una bibliotca de `R-studio` que da soporte a plataformas de microarreglos, incluidas, `Affymetrix`, `Illumina`, `Nimblegen`, `Agilent` y otras tecnologías que se enfocan en la hibridación de microarreglos con base en dos colores diferentes.

[12] Dentro de la Biología Celular, el estado basal se define como el nivel de actividad de una función orgánico durante el reposo y el ayuno.

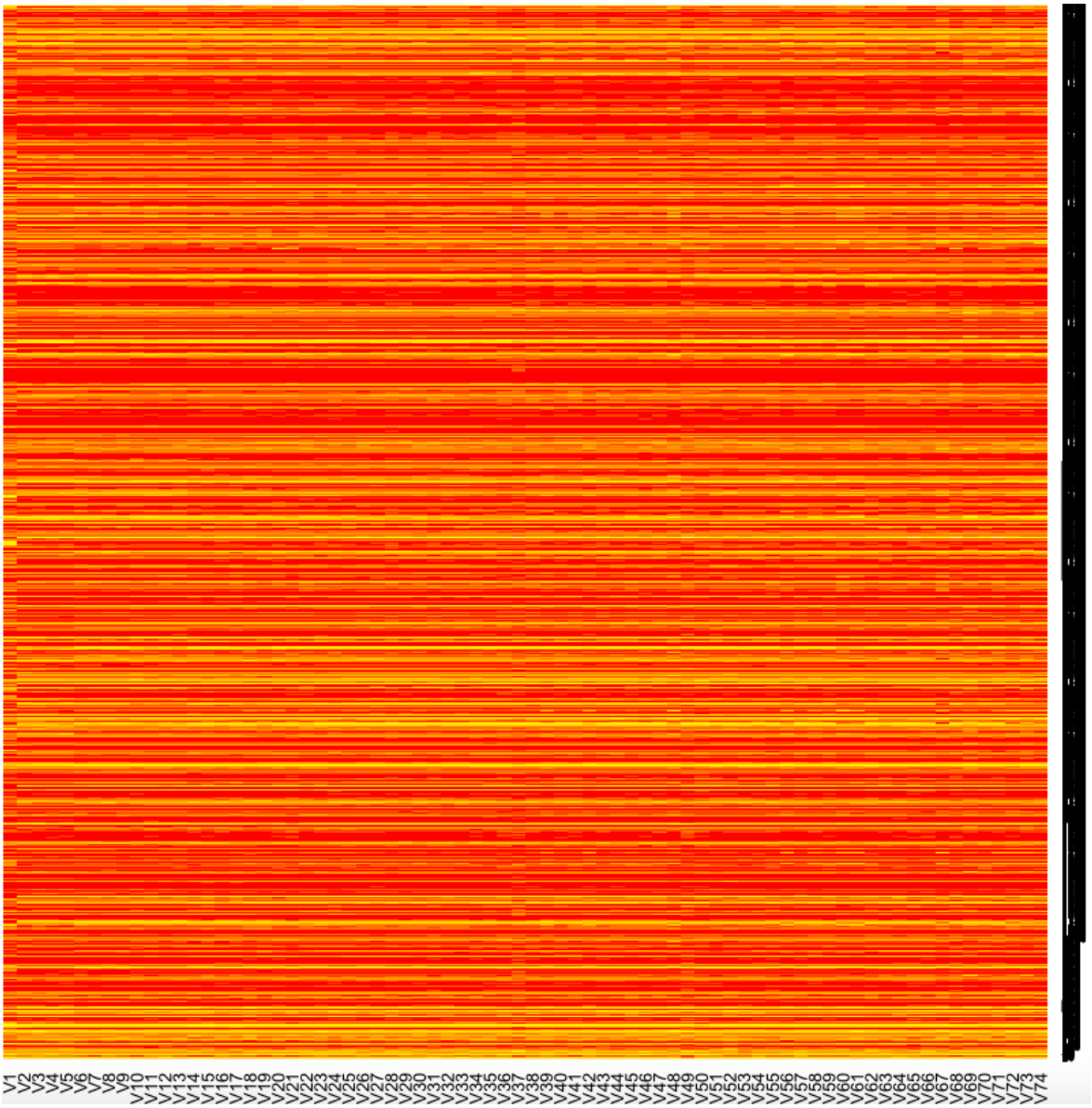


Figura 5-8: Mapa de calor que representa la matriz de expresión genética en el conjunto de datos relacionado a hepatocarcinomas. Las filas de la matriz representan los genes involucrados en el experimento, mientras que las columnas representan cada una de las condiciones experimentales. Cabe mencionar que el análisis de agrupamiento a simple vista es difícil considerando que es un conjunto de datos de alta dimensionalidad.

Tabla 5.16: Resumen de las estadísticas obtenidas para el análisis de agrupamiento del conjunto de datos relacionado a hepatocarcinomas. Se reportan el número de patrones analizados y el Error Cuadrático Medio (ECM) extragrupal e intragrupal para cuando $K = \{10, 12, 13, 15, 17\}$. Se observa que cuando $K = 13$ el Error Cuadrático Medio (ECM) intragrupal es mínimo.

Experimento	K				
Número de grupos	10	12	13	15	17
Número de patrones	8,794	8,794	8,794	8,794	8,794
ECM extragrupal	1,301.0	1,315.4	1,320.25	1,319.8	1,319.3
ECM intragrupal	91.31	76.93	72.05	72.43	72.95
ECM total	1,392.3	1,392.3	1,392.3	1,392.3	1,392.3

Lo anterior sucede especialmente en la región genética definida por el intervalo que se ubica entre 6,000 y 6,400 pares de bases nitrogenadas.

Para definir el número de grupos contenidos en la matriz de expresión genética se realizó un análisis exploratorio, en el cual se ejecutó el algoritmo KM secuencial variando el número de grupos definido por el parámetro K. En cada caso se obtuvo la varianza que existe en cada uno de los grupos. En la figura 5-18 se observa que a medida que el valor de K incrementa, el valor $WCSS$ disminuye de tal forma que la curva denominada *elbow* se obtiene cuando $K = 13$. En este experimento se analizaron las 16 condiciones experimentales y se utilizó una escala normalizada.

En la tabla 5.16 se describe el resumen de las estadísticas obtenidas para poder obtener una estimación del número óptimo de grupos. Se reportan los valores obtenidos para cuando $K = \{10, 12, 13, 15, 17\}$. Se observa que cuando $K = 13$ el Error Cuadrático Medio (ECM) intragrupal es mínimo. Finalmente, en la tabla 5.17 se describen los centroides obtenidos de este experimento.

Se describe la eficiencia computacional del marco de trabajo aplicado a tres experimentos que involucran tanto técnicas de agrupamiento enfocadas en los algoritmos KM y FCM, como una técnica de biagrupamiento enfocada en el algoritmo BIMAX. De manera general y con base en los resultados obtenidos previamente, para los algoritmos de agrupamiento se definió el parámetro $K = 13$.

Análisis de expresión diferencial utilizando la misma técnica de agrupamiento. En este experimento se analiza el rendimiento del marco de trabajo auto-adaptable utilizando el mismo algoritmo (KM) en cada uno de los nodos sin variar el tipo de métrica de similitud en todo el experimento. La métrica de similitud que se ha utilizado es la distancia euclidiana descrita en la sección 3.2.1. Se ha utilizado la matriz de expresión genética sin el proceso de discretización. Para los algoritmos de consenso se definieron los parámetros como $\epsilon = 0{,}001$ y $t_{max} = 200$. El número total de iteraciones para el algoritmo de agrupamiento

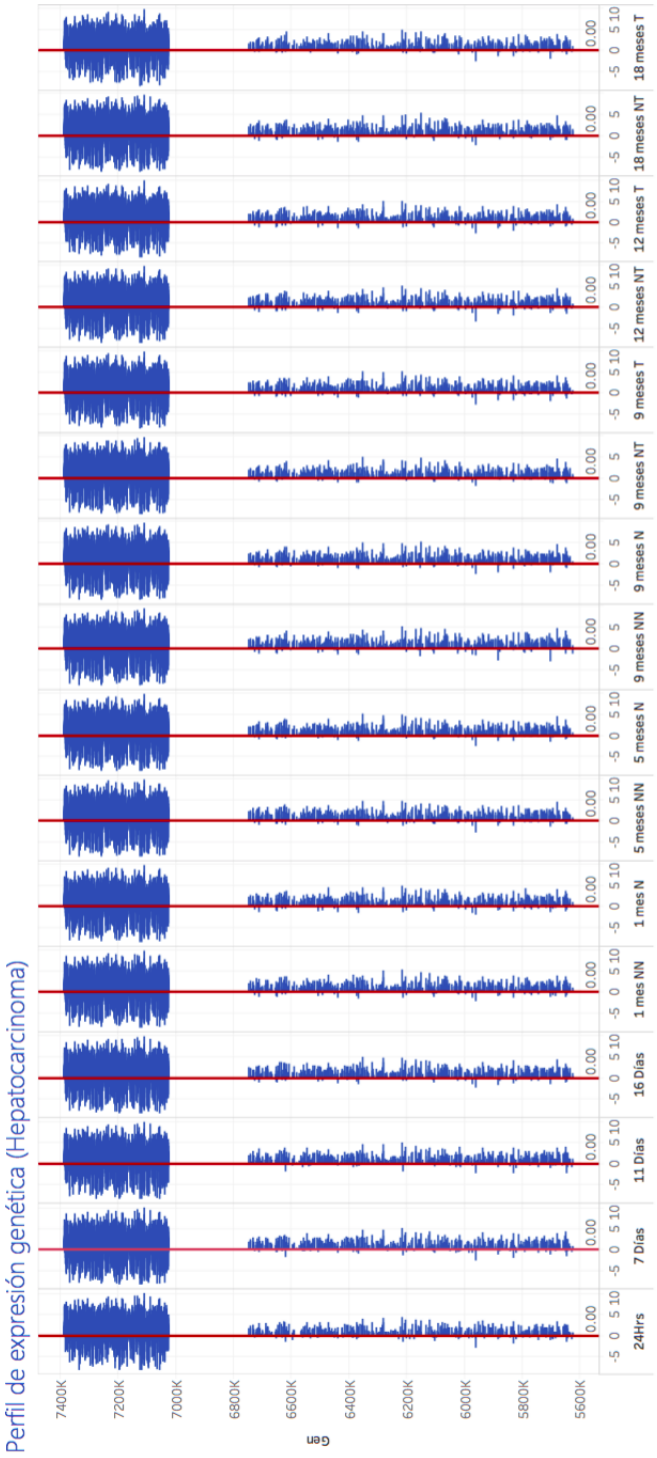


Figura 5-9: perfil de expresión genética de los 8,794 genes ante su exposición a 16 condiciones experimentales. En el eje vertical se representan cada uno de los genes, mientras que el eje horizontal describe el perfil de expresión de un gen específico en una condición experimental particular. El perfil se obtuvo a través de la diferencia con respecto a la condición basal. En este sentido se observa que en la mayoría de los casos, el perfil de expresión tiende a disminuir con respecto a la condición basal. Esto demuestra el silenciamiento genético.

Tabla 5.17: Resumen de los centroides para cada una de las condiciones experimentales involucradas en el experimento cuando $K = 13$. Se reporta el total de patrones contenido en cada una de las agrupaciones. Se observa que los grupos 1 y 2 son los que contienen una mayor cantidad de patrones (≈ 2000), mientras que el grupo 13 es el que se caracteriza con el menor número de patrones (47 elementos).

grupo	N_{pat}	24h	7d	11d	16d	1mNN	1mN
1	1,894	0.5105	0.5210	0.5241	0.5264	0.5297	0.5294
2	1,948	0.5358	0.5515	0.5517	0.5563	0.5584	0.5587
3	410	0.4412	0.4337	0.4433	0.4439	0.4484	0.4482
4	849	0.6032	0.6170	0.6197	0.6271	0.6272	0.6288
5	1,465	0.5636	0.5800	0.5802	0.5867	0.5886	0.5892
6	196	0.4152	0.3740	0.3928	0.3814	0.3814	0.3815
7	266	0.7379	0.7440	0.7470	0.7511	0.7463	0.7500
8	495	0.6607	0.6692	0.6717	0.6788	0.6761	0.6788
9	886	0.4719	0.4803	0.4862	0.4949	0.4977	0.4970
10	115	0.3286	0.2780	0.2989	0.2836	0.2903	0.2890
11	105	0.2257	0.1647	0.1910	0.1719	0.1744	0.1785
12	117	0.8371	0.8481	0.8509	0.8547	0.8489	0.8515
13	47	0.0877	0.0599	0.0652	0.0619	0.0682	0.0705

grupo	5mNN	5mN	9mNN	9mN	9mNT	9mT	12mNT
1	0.5344	0.5275	0.5388	0.5394	0.5338	0.5343	0.5343
2	0.5626	0.5553	0.5643	0.5394	0.5591	0.5614	0.5610
3	0.4455	0.4420	0.4666	0.4542	0.4502	0.4522	0.4491
4	0.6357	0.6280	0.6419	0.6395	0.6361	0.6295	0.6322
5	0.5941	0.5874	0.5973	0.5969	0.5918	0.5917	0.5920
6	0.3740	0.3702	0.3938	0.3805	0.3767	0.3869	0.3825
7	0.7607	0.7520	0.7592	0.7611	0.7581	0.7545	0.7552
8	0.6886	0.6809	0.6920	0.6912	0.6884	0.6819	0.6844
9	0.5018	0.4959	0.5182	0.5096	0.5058	0.5007	0.5027
10	0.2828	0.2774	0.3050	0.2887	0.2907	0.2983	0.2929
11	0.1600	0.1586	0.1748	0.1657	0.1611	0.1906	0.1780
12	0.8625	0.8525	0.8596	0.8636	0.8587	0.8566	0.8559
13	0.0599	0.0582	0.0685	0.0641	0.0639	0.0805	0.0731

grupo	12mT	18mNT	18mT
1	0.5222	0.5391	0.5302
2	0.5489	0.5640	0.5544
3	0.4433	0.4559	0.4611
4	0.6168	0.6408	0.6203
5	0.5790	0.5962	0.5830
6	0.3817	0.3832	0.4029
7	0.7382	0.7628	0.7399
8	0.6677	0.6934	0.6684
9	0.4907	0.5120	0.5014
10	0.2950	0.2898	0.3167

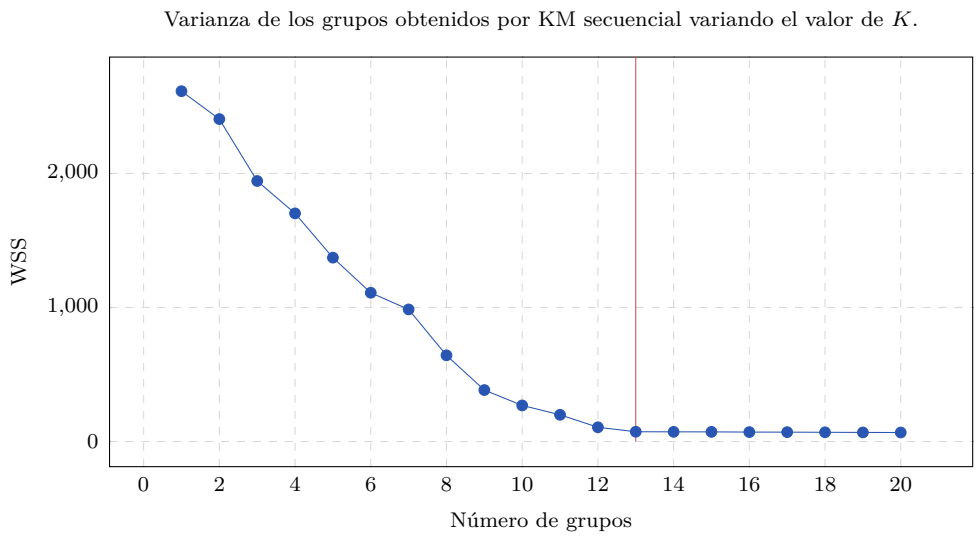


Figura 5-10: Método del codo para estimar el número óptimo de grupos mediante la ejecución secuencial del algoritmo de KM. Se observa que a medida que se aumenta el valor de K, el valor `WCSS` disminuye de tal forma que la gráfica adopta una forma de codo (*elbow*). Para seleccionar el valor óptimo de K se selecciona a partir de este punto, en donde, ya no se producen variaciones importantes, es decir cuando $K >= 13$.

se definieron como $R = 1000$ iteraciones y $\rho = 0{,}001$. Finalmente, considere que la arquitectura es homogénea.

En la figura 5-11 se describe la convergencia de la función objetivo para este experimento. Se describe el tiempo secuencial optimizado con las banderas `-O1`, `-O2` y `-O3`. El experimento se enfocó en la estimación del tamaño de bloque y las banderas de *GCC* que optimizan el tiempo computacional del algoritmo en una arquitectura con $n = 4$ nodos. Se observa que la optimización mejora si durante el proceso de compilación se realiza utilizando la bandera `-O2`, mientras que las banderas `-O3` y `-O1` no contribuyen en dicha optimización.

En la tabla 5.18 se presenta un resumen de las primeras 10 banderas de `GCC` que más contribuyen en la mejora del rendimiento durante la ejecución del algoritmo KM. La importancia se define como la ralentización cuando se elimina la bandera de la configuración final en la consola `GCC`. Se han resaltado tres banderas que también han sido reportadas con anterioridad en la sección 5.5.7. Se observa que el 36.1 % lo ocupa la bandera `-funroll-all-loops`, la cual se encarga de desenrollar todos los bucles automáticamente sin ningún esfuerzo por parte del usuario. Respecto a este resultado, y como en el estado del arte lo menciona que el desenrollamiento de bucles en algunos casos puede ralentizar el rendimiento por ejemplo cuando existe una sobrecarga de la función `printf`, en este caso simplemente se incluyen operaciones matemáticas anidadas para ciclos que iteran elementos de la matriz de expresión génica y la cual tiene una gran cantidad de elementos.

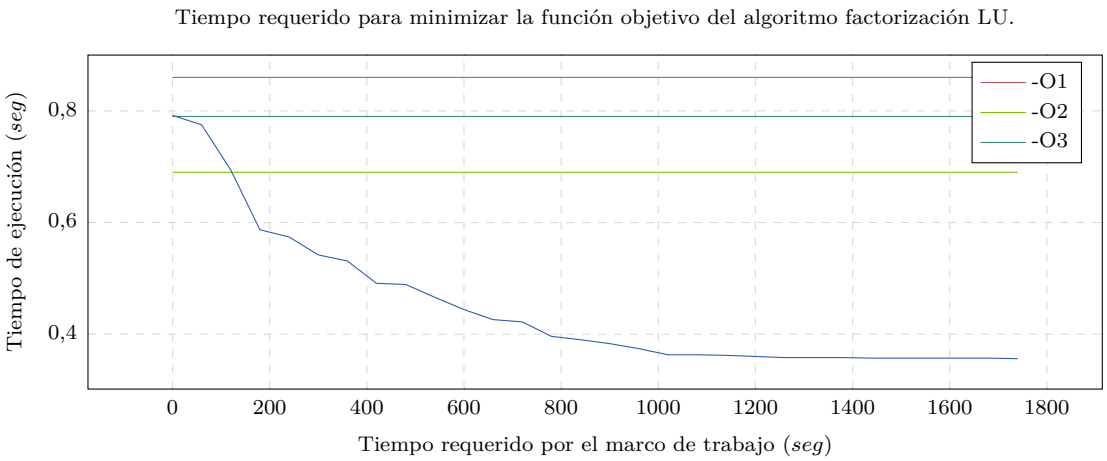


Figura 5-11: Convergencia de la función objetivo definida por la permutación de parámetros que minimizan el tiempo de ejecución durante la auto-adaptación de parámetros para el primer experimento.

Por otro lado, la bandera `-ffast-math` aumenta la velocidad de las operaciones de punto flotante, sin embargo, pueden afectar el redondeo de los valores resultantes debido a que no entra en el estándar de la IEEE. Lo mismo sucede con la bandera `funsafe-math-optimizations`. La explicación más clara de este punto es que la aritmética de punto flotante no es asociativa, y entonces, tanto el orden como la factorización de las operaciones afectarán los resultados debido al redondeo. Se debe considerar que la aritmética de punto flotante no es asociativa y por tanto, el orden y la factorización de las operaciones afectarán los resultados debido al redondeo. Cabe mencionar que esta optimización no se realiza bajo un comportamiento estricto. En este caso el resultado será correcto, sin embargo puede diferir en los últimos bits debido a las operaciones de redondeo.

Análisis de expresión diferencial utilizando diferentes técnicas de agrupamiento. En este experimento se analiza el rendimiento del marco de trabajo auto-adaptable utilizando diferentes algoritmos (KM, FCM) en cada uno de los nodos sin variar el tipo de métrica de similitud en todo el experimento. La métrica de similitud utilizada es la distancia euclidiana para KM y la distancia de coseno para FCM. Cabe mencionar que esta última tiene alto rendimiento para la agrupación de datos de alta dimensionalidad. Se ha utilizado la matriz de expresión genética no discretizada. Para los algoritmos de consenso se definieron los parámetros como $\epsilon = 0{,}001$ y $t_{max} = 200$. El número total de iteraciones para el algoritmo de agrupamiento se definieron como $R = 1000$ iteraciones y $\rho = 0{,}001$. Finalmente, considere que la arquitectura es homogénea. Para los algoritmos de consenso se definieron los parámetros como $\epsilon = 0{,}001$ y $t_{max} = 200$. El número total de iteraciones para el algoritmo de agrupamiento se definieron como $R = 1000$ iteraciones y $\rho = 0{,}001$. Finalmente, considere que la arquitectura es homogénea.

Tabla 5.18: Las primeras 10 banderas de `GCC` que más contribuyen en la mejora del rendimiento durante la ejecución del algoritmo KM. La importancia se define como la ralentización cuando se elimina la bandera de la configuración final en la consola `GCC`. La importancia se normaliza para sumar 100 %.

Bandera	Importancia (%)
`-funroll-all-loops`	36.1 %
`-fwrapv`	17.6 %
`-funsafe-math-optimizations`	14.5 %
`-ffast-math`	8.4 %
`-ffinite-loops`	6.9 %
`-fstrict-overflow`	4.5 %
`-finline-small-functions`	2.8 %
`-freorder-blocks-algorithm=stc`	1.7 %
`-param=max-sched-extend-regions-iters=2`	0.5 %
`-finline-small-functions`	0.1 %
`272 otras banderas`	16.9 %

En la figura 5-12 se describe la convergencia de la función objetivo para este experimento. Se describe el tiempo secuencial optimizado con las banderas `-O1`, `-O2` y `-O3`. El experimento se enfocó en la estimación del tamaño de bloque y las banderas de *GCC* que optimizan el tiempo computacional del algoritmo en una arquitectura con $n = 4$ nodos. Se observa que la optimización mejora si durante el proceso de compilación se realiza utilizando las banderas `-O2` y `-O3`, mientras que las `-O1` no contribuye en dicha optimización.

En la tabla 5.19 se presenta un resumen de las primeras 10 banderas de `GCC` que más contribuyen en la mejora del rendimiento durante la ejecución del algoritmo KM. Se han resaltado cuatro banderas que también han sido reportadas con anterioridad en la sección 5.5.7. En este experimento la bandera `-funroll-all-loops` es la que más aporta en la mejora del rendimiento de tiempo computacional. Sin embargo, se presenta por primera vez la bandera `-floop-parallelize-all` que distribuye código secuencial en código multiproceso. Además, genera automáticamente código paralelo para construcciones de ciclos específicas utilizando la biblioteca `gomp` en unión de la biblioteca `omp`.

Análisis de expresión diferencial utilizando técnicas de biagrupamiento. En este experimento se analiza el rendimiento del marco de trabajo utilizando la biblioteca de operaciones elementales para mejorar el rendimiento del algoritmo BIMAX [193], específicamente de las operaciones elementales. En este experimento se utilizó la matriz de expresión genética discretizada tal y como lo pide el algoritmo. En este caso, solo se especificó el número mínimo de genes ($min_{rows} = 4$) y condiciones ($min_{cols} = 4$) que debian de tener cada una de las agrupaciones. En este sentido, es importante hacer mención que para las

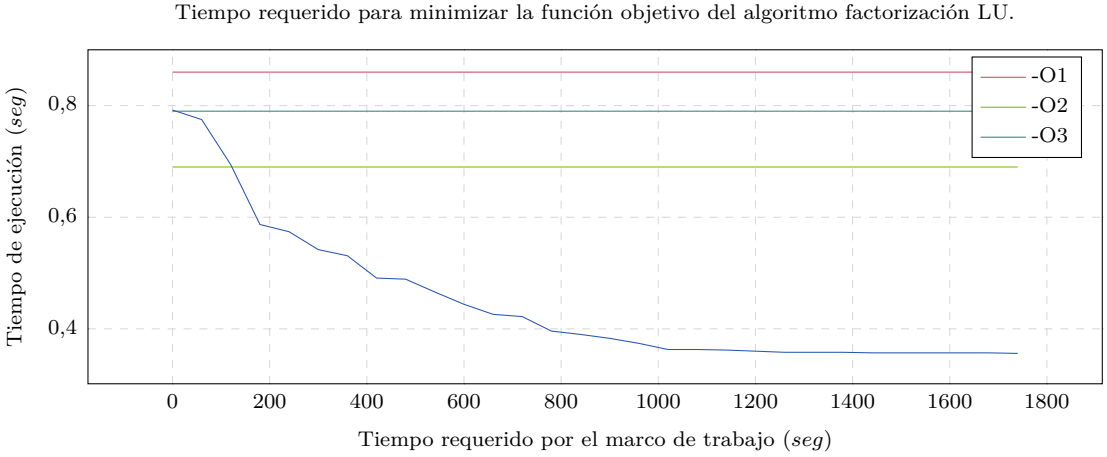


Figura 5-12: Convergencia de la función objetivo definida por la permutación de parámetros que minimizan el tiempo de ejecución durante la auto-adaptación de parámetros para el segundo experimento.

Tabla 5.19: Las primeras 10 banderas de `GCC` que más contribuyen en la mejora del rendimiento durante la ejecución del algoritmo FCM y KM en una arquitectura con $n = 4$ nodos. La importancia se define como la ralentización cuando se elimina la bandera de la configuración final en la consola GCC. La importancia se normaliza para sumar 100 %.

Bandera	Importancia (%)
`-funroll-all-loops`	31.8 %
`-fwrapv`	23.4 %
`-funsafe-math-optimizations`	17.3 %
`-fno-exceptions`	10.4 %
`-floop-parallelize-all`	5.2 %
`-ftree-loop-vectorize`	1.8 %
`-ffinite-math-only`	0.8 %
`max-unroll-times=4`	0.4 %
`-funswitch-loops`	0.3 %
`-fsingle-precision-constant`	0.2 %
`272 otras banderas`	8.4 %

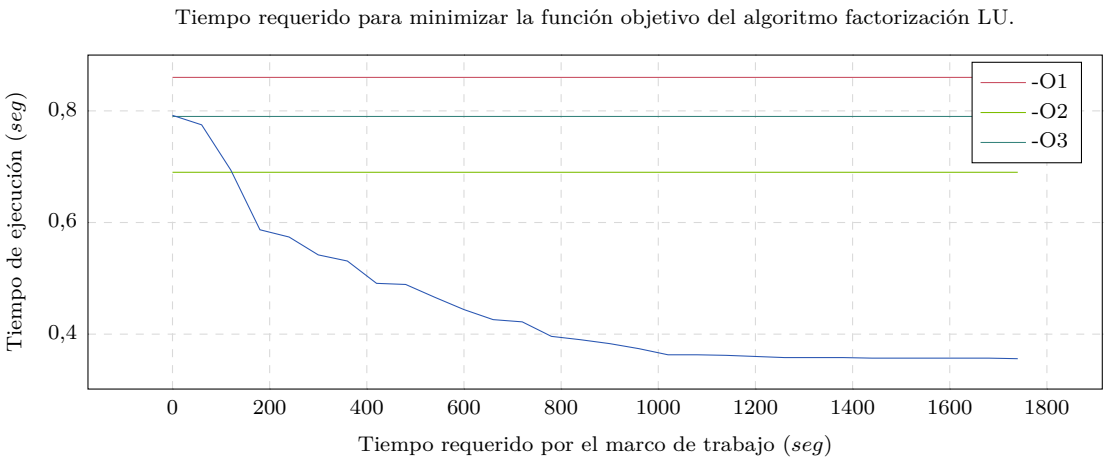


Figura 5-13: Convergencia de la función objetivo definida por la permutación de parámetros que minimizan el tiempo de ejecución durante la auto-adaptación de parámetros para el tercer experimento.

técnicas de biagrupamiento no es necesario especificar el valor de K.

En la tabla 5.20 se describen el resumen de las banderas que mayor aporte tienen en la mejora del rendimiento computacional del algoritmo durante el proceso de compilación. Como se aprecia en la mayoría de los experimentos se presentan las banderas `-funroll-all-loops`, `-funsafe-math-optimizations` y `-fwrapv`. En este sentido se puede afirmar que el compilador realiza las optimizaciones necesarias basadas en suposiciones fundamentadas sobre el orden en que deben ejecutarse las instrucciones. Sin embargo, lo anterior podría implicar la carga de instrucciones del ejecutable en la memoria en el caché de instrucciones integrado en la unidad de procesamiento (l-caché). Esto tiene una cantidad limitada de instrucciones a las que se puede acceder rápidamente, pero puede bloquearse cuando es necesario cargar nuevas instrucciones desde la memoria. Finalmente, se debe considerar que la bandera `-funroll-all-loops` proporciona más beneficios para ciclos más cortos, pero podría terminar arruinando el rendimiento si se tiene la intención de repetir la instrucción una gran cantidad de veces, considerando que si el tamaño de la matriz es variable, la bandera no tendrá ningún efecto.

- **Interpretación y evaluación.** Una vez obtenido el modelo, se procedió a su validación comprobando que las conclusiones obtenidas son válidas y representan el contexto biológico descrito en el estado del arte [177]. El modelo tuvo que ser ajustado hasta que obtener resultados congruentes con la realidad. En este sentido, el proceso de *feedback* se aplicó principalmente a la etapa de «preprocesamiento y transformación» y a la fase relacionada al «proceso de MD». Se describen los resultados más relevantes.

  En la figura 5-14 se describe el perfil de expresión genética considerando $K = 13$. El eje vertical describe las pares de bases nitrogenadas, mientras que el eje horizontal las 16 condiciones experimentales. Se puede apreciar que existe un

Tabla 5.20: Las primeras 10 banderas de `GCC` que más contribuyen en la mejora del rendimiento durante la ejecución del algoritmo BIMAX. La importancia se define como la ralentización cuando se elimina la bandera de la configuración final en la consola GCC. La importancia se normaliza para sumar 100 %.

Bandera	Importancia (%)
`-funroll-all-loops`	25.8 %
`-fwrapv`	24.9 %
`-funsafe-math-optimizations`	16.7 %
`-fno-exceptions`	12.4 %
`-fno-rerun-cse-after-loop`	3.5 %
`-funswitch-loops`	0.8 %
`-ffinite-loops`	0.8 %
`-fschedule-insns2`	0.7 %
`-funswitch-loops`	0.3 %
`-fthread-jumps`	0.2 %
`272 otras banderas`	13.9 %

cambio significativo en el intervalo comprendido entre las 5700 y 5800 pares, en los cuales se aprecia la sobreexpresión cuando existe la presencia de tumor con respecto a las demás condiciones experimentales. Sin embargo, en la mayoría de los casos es más evidente que los genes tienen a suprimirse cuando existe un nódulo o un tumor.

Por otro lado, en la figura 5-15 se describe la densidad de cada uno de los grupos para cuando $K = 13$. El eje vertical representa las pares de bases nitrogenadas que componen a cada uno de los genes, mientras que el eje horizontal describe a las 13 agrupaciones. Los grupos 7, 10, 11, 12, y 13 tienen poca densidad en las primeras 1200 pares de bases. Por otro lado, los grupos 1, 2, 5, y 9 son los que tienen una mayor densidad. Existen regiones que se comportan de manera similar en los grupos 5 y 9 siendo en su mayoría el complemento del grupo 1. Se observa que al menos el 61 % de los grupos tienen una densidad marcada en las últimas 400 pares de bases. Sin embargo, en la literatura, estos pares de bases no tienen relevancia biológica.

Se realizó el análisis del perfil de expresión genética a través de comparaciones directas entre la condición experimental relacionada a la ausencia de nódulos (NN) contra la presencia de nódulos (N) en 1, 5 y 9 meses (ver figura 5-16a). En la figura 5-16b se describe el análisis de agrupación para cuando $K = 9$. Para conocer el valor óptimo de K se procedió de la misma forma que en la sección 5-18. Se observa que durante el primer mes no existe un cambio significativo, sin embargo, para 5 y 9 meses es evidente la inhibición del nivel de expresión genética. Lo anterior se acentúa más mientras avanza el tiempo.

Se realizó el análisis de la evolución de la enfermedad considerando la ausen-

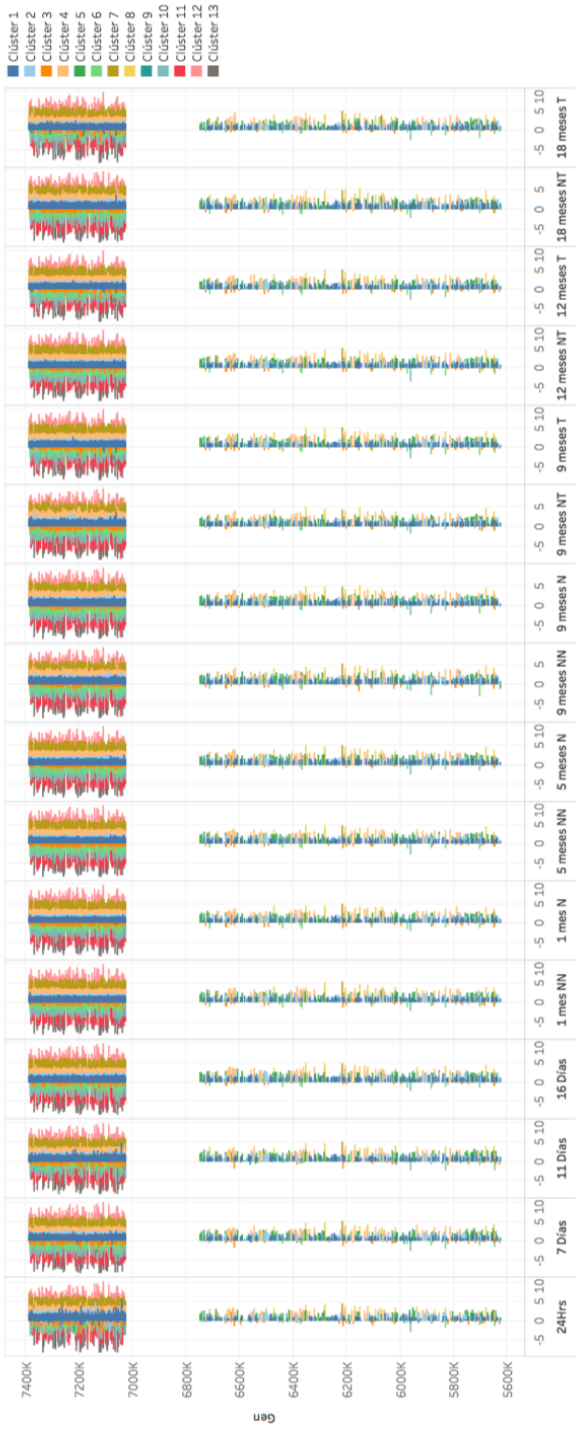


Figura 5-14: Perfil de expresión genética obtenido a través del cuando $K = 13$. El eje vertical describe las pares de bases nitrogenadas, mientras que el eje horizontal las 16 condiciones experimentales. Se puede apreciar que existe un cambio significativo en el intervalo comprendido entre las 5700 y 5800 pares, en los cuales se aprecia la sobreexpresión cuando existe la presencia de tumor con respecto a las demás condiciones experimentales. Sin embargo, en la mayoría de los casos es más evidente que los genes tienen a suprimirse cuando existe un nódulo o un tumor.

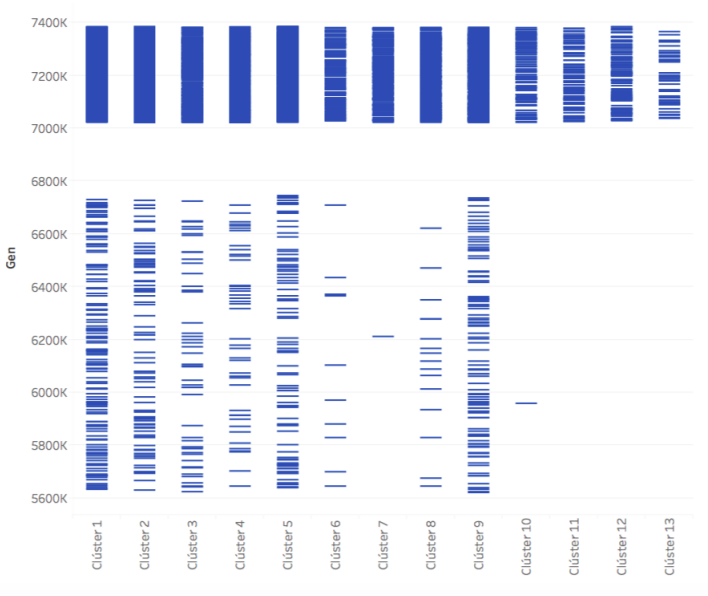
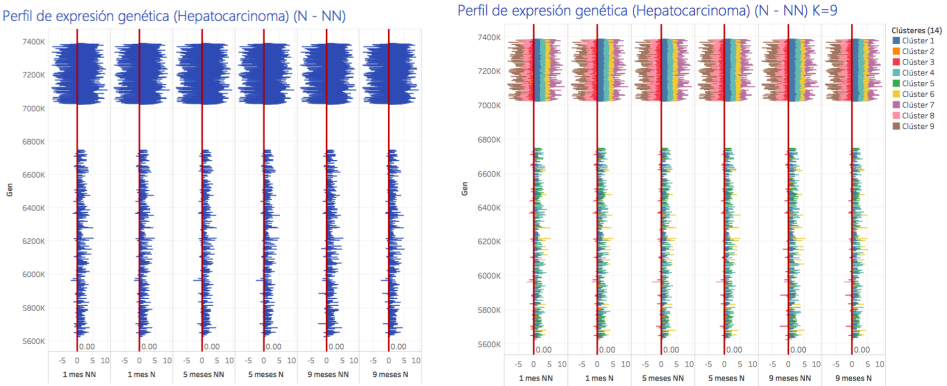
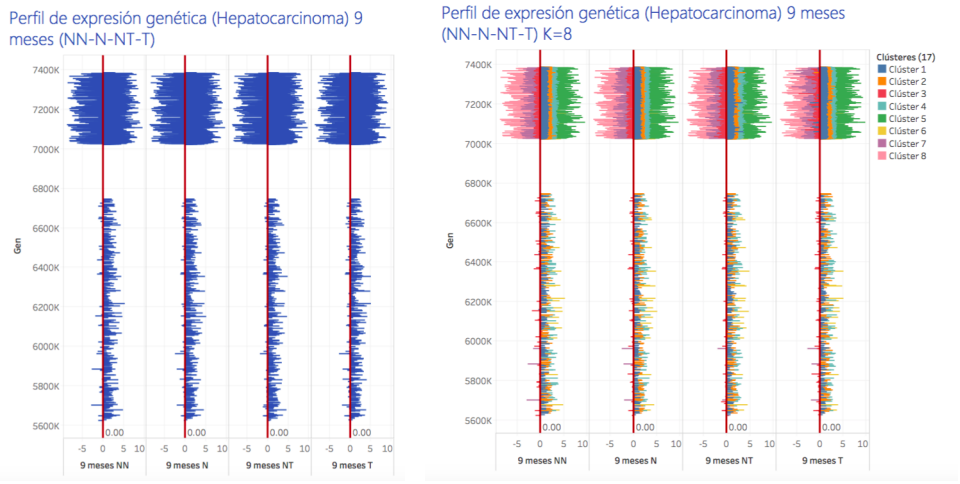


Figura 5-15: Densidad de cada uno de los grupos para cuando $K = 13$. El eje vertical representa las pares de bases nitrogenadas que componen a cada uno de los genes, mientras que el eje horizontal describe a las 13 agrupaciones. Los grupos 7, 10, 11, 12, y 13 tienen poca densidad en las primeras 1200 pares de bases. En contraste, los grupo 1, 2, 5, y 9 son los que tienen una mayor densidad. Existen regiones que se comportan de manera similar en los grupos 5 y 9 siendo en su mayoría el complemento del grupo 1.



(a) no agrupación **(b)** agrupación

Figura 5-16: Análisis del perfil de expresión genética a través de comparaciones directas entre la condición experimental relacionada a la ausencia de nódulos (NN) contra la presencia de nódulos (N) en 1, 5 y 9 meses. Se observa que durante el primer mes no existe un cambio significativo, sin embargo, para 5 y 9 meses es evidente la inhibición del nivel de expresión genética. Lo anterior se acentúa más mientras avanza el tiempo.



(a) no agrupación **(b)** agrupación

Figura 5-17: Análisis de la evolución de la enfermedad considerando la ausencia y presencia tanto de nódulo, como de tumor a los 9 meses. Si se realiza la comparación entre la ausencia de tumor (NT) y la presencia de tumor (T), se observa que al menos el 15 % de los genes se expresa en una cantidad mínima ($\approx 0{,}2$ por arriba del nivel basal). Este incremento se da desde que se presenta un nódulo en el tejido. Por otro lado, si se realiza la comparación entre la presencia de nódulo (N) y no nódulo (NN) se observa que en vez que aumentar el nivel de expresión, disminuye de manera considerable.

cia y presencia tanto de nódulo, como de tumor. En este sentido, se analizó el microarreglo correspondiente al experimento relacionado a los 9 meses de desarrollo de la enfermedad con un valor de $K = 9$. Si se realiza la comparación entre la ausencia de tumor (NT) y la presencia de tumor (T), se observa que al menos el 15 % de los genes se expresa en una cantidad mínima ($\approx 0{,}2$ por arriba del nivel basal). Este incremento se da desde que se presenta un nódulo en el tejido. Por otro lado, si se realiza la comparación entre la presencia de nódulo (N) y no nódulo (NN) se observa que en vez que aumentar el nivel de expresión, disminuye de manera considerable.

Además, se indagó en la pregunta relacionada al conjunto de genes que están implicados en el desarrollo tumoral de CHC. Para proponer una solución se clasificaron los genes que se sobre-expresaban o inhibían en la condición basal, en la condición relacionada a la presencia de nódulo y en la presencia de tumor. En la figura 5-18 se describe un diagrama de Venn, en el que se aprecia que el 15.9 % de los genes se encuentran activos en la fase de cambio de condición basal a nódulo. Además, se tiene un 19.6 % de genes que se sobre-expresan tanto en presencia de nódulo, como de tumor. Finalmente, existe un 18.4 % de genes que solo se activan en la condición de tumor y un 13.3 % de genes que están presentes en la condición de nódulo.

En la figura 5-19 se describe la anotación del 67.2 % de los genes con comportamiento correlacionado en sobreexpresión, específicamente en el desarrollo

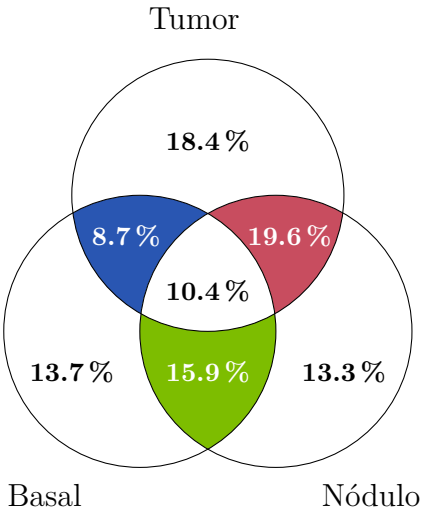


Figura 5-18: Diagrama de Venn correspondiente al 17.3 % de genes con comportamiento correlacionado en sobreexpresión en nódulo y tumor utilizando algoritmos de agrupamiento (KM - FCM) en una arquitectura heterogénea con 4 nodos.

tumoral. Esta gráfica muestra que al menos el 39 % de los genes están involucrados con funciones metabólicas de la célula y por tanto, podría indicar que la célula requiere una mayor cantidad de nutrientes.

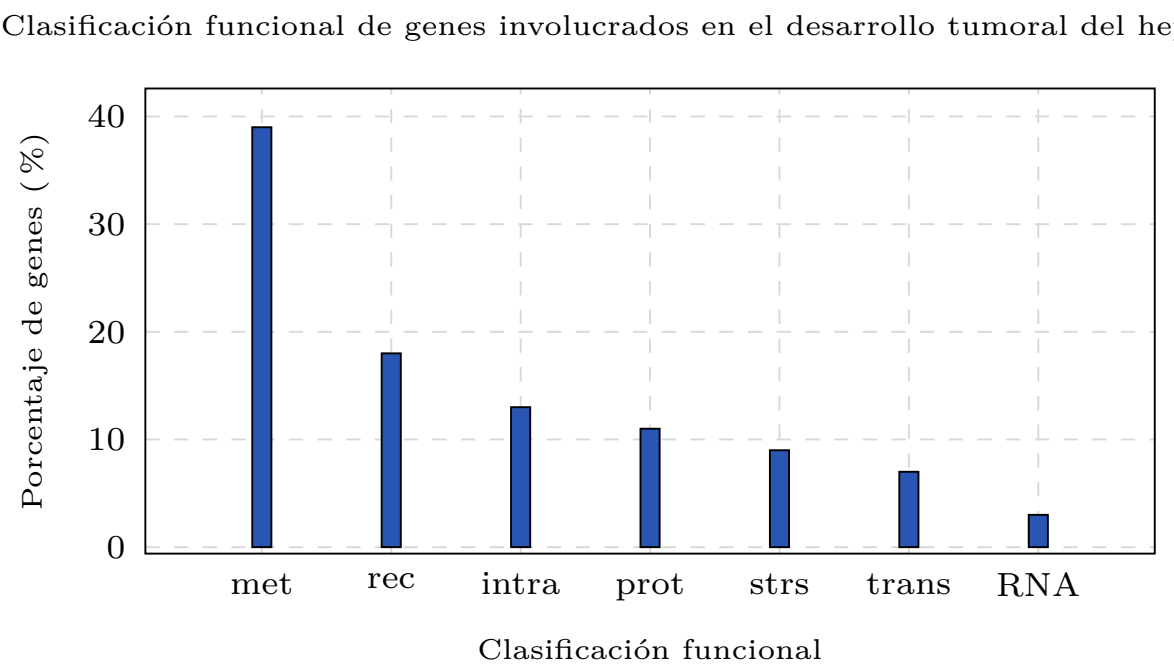


Figura 5-19: Clasificación funcional del 67.2 % de genes con comportamiento correlacionado en sobreexpresión específicamente en el desarrollo tumoral del hepatocito. El 39 % de estos genes están involucrados en funciones metabólicas.

Capítulo 6
Conclusiones y trabajo a futuro

Se han estudiado a profundidad los mecanismos de auto-adaptación orientados a mejorar el rendimiento computacional. Se han analizado las ventajas del marco de trabajo *Self-Adapting Numerical Software* (`SANS`) para la propuesta de una metodología robusta orientada en la maximización del rendimiento computacional que reduzca el tiempo de ejecución de un algoritmo específico en tiempo de compilación

Considerando los componentes básicos de una metodología `SANS`, en este trabajo se implementó un marco de trabajo de auto-adaptación basado en técnicas de optimización que permite minimizar el tiempo de ejecución de un algoritmo mediante la selección de un conjunto de parámetros durante el proceso de compilación, independientemente de la arquitectura y del compilador que se esté utilizando, con el objetivo de asegurar la portabilidad y el uso eficiente de los recursos computacionales. Lo anterior ha demostrado ser válido para técnicas de MD enfocadas en tareas de clasificación y agrupación, especialmente para algoritmos enfocados en tareas de agrupamiento y biagrupamiento cuyo objetivo es el análisis diferencial de datos «ómicos» de alta dimensionalidad.

Este marco de trabajo realiza la auto-adaptación en dos fases: (1) Durante la instalación realizando un conjunto de experimentos que involucran la ejecución simultánea de operaciones elementales. Durante esta fase, se recolecta información propia de la arquitectura, tal como el tiempo necesario para iniciar la comunicación y el tiempo para enviar un tipo de dato específico; (2) Durante la ejecución del algoritmo a través de la minimización de una función objetivo basada en un modelo matemático que refleja el tiempo de ejecución de un algoritmo en una arquitectura.

El marco de trabajo se integra de tres componentes principales: (1) Componente de *configuración* el cual define las técnicas que el adaptador utilizará para leer y escribir distintas configuraciones; (2) Componente de *evaluación* cuyo objetivo es estimar el valor de la función objetivo con base en las configuraciones dadas; y una (3) Base de datos la cual registra todos los resultados recopilados durante la instalación y la adaptación de un programa específico. Durante un experimento, el componente (1) y (2) interactuán de manera simultánea utilizando diferentes técnicas de búsqueda hasta encontrar una solución óptima. Cabe mencionar que para futuras adaptaciones de código, el `configuración` considera los datos históricos almacenados en el componente (3) con el objetivo de retroalimentar una la toma de decisiones sobre un nuevo experimento.

Los resultados demostraron que la propuesta es capaz de obtener configuraciones óptimas de los parámetros durante la ejecución de un algoritmo con base en un análisis comparativo con otras propuestas del estado del arte: `HPL` y `PetaBricks`. El marco de trabajo es capaz de analizar que tipo de banderas de `GCC` se pueden o no activar para mejorar el rendimiento computacional. En principio, la función objetivo utilizada minimiza el tiempo de ejecución de un algoritmo en particular a través de algoritmos de optimización, tales como, optimización por enjambre de partículas, evolución diferencial y el algoritmo de escalada simple.

La aplicación del marco de trabajo en el análisis diferencial de datos genómicos, especialmente en técnicas agrupamiento con algoritmos como KM, FCM y EM mostró resultados competitivos que logran reducir el tiempo de ejecución de un algoritmo específico. En este sentido, cabe mencionar que a pesar de que el marco de trabajo requiere de 33 minutos aproximadamente para encontrar una permutación óptima de parámetros que minimice el tiempo de ejecución de un algoritmo de FCM con base en las propiedades algorítmicas y de sistema de una arquitectura específica durante un experimento, en ocasiones, es recomendable esperar este tiempo para obtener un resultado más rápido, que la ejecución simple del algoritmo que pueda tardar días, semanas o meses. Finalmente, el marco de trabajo ha sido implementado en `Python` considerando que es un lenguaje interpretado, dinámico y multiplataforma.

El material presentado en este trabajo contribuye a incrementar el estado del arte en técnicas de auto-adaptación y a motivar su uso tanto para implementar algoritmos de MD, como para resolver un problema determinado. Se ha mostrado el potencial de incluir en el modelo matemático durante la adaptación de parámetros para buscar diferentes conjunto. A continuación, se exponen las conclusiones, así como el trabajo a futuro que resultó de la investigación.

De forma particular, se han investigado las principales técnicas de agrupamiento particional para el análisis de datos de expresión genética. Las técnicas de agrupamiento se han diseñado para organizar los datos en clases de aprendizaje no supervisado con base en dos principios: (1) Maximización, el cual busca encontrar el mayor grado de relación entre los datos; (2) Minimización, el cual busca conseguir la mayor diferencia entre las clases. Se ha propuesto la generalización del algoritmo particional de manera iterativa. Se propuso un algoritmo distribuido de agrupamiento particional basado en mecanismos de paso de mensaje utilizando la biblioteca `MPI`. Este enfoque se basa en arquitecturas de memoria distribuida orientadas a resolver de forma simultánea algoritmos de agrupamiento sobre subconjuntos de patrones. El objetivo es entonces poder llegar a un resultado único de entre los distintos resultados (potencialmente conflictivos) a través de un protocolo de consenso (ver sección 3.4).

Para mejorar el rendimiento del algoritmo, se implementó una biblioteca que permite aprovechar los recursos computacionales de las diversas arquitecturas disponibles. La principal característica de esta biblioteca es la paralelización de código para arquitecturas de múltiples unidades de procesamiento por medio del estándar de `CUDA` y `OpenMP`. La biblioteca involucra tres componentes principales: (1) `Templates` que describen expresiones convencionales de operaciones elementales; (2) Una polibiblioteca que funciona como una interfaz capaz de instanciar un método específico, independientemente del estándar utilizado; (3) Base de datos cuyo objetivo es al-

macenar los parámetros y resultados obtenidos durante una ejecución. Esto permite retroalimentar futuras ejecuciones a través de un mecanismo de semi auto-adaptación. Durante el análisis de rendimiento, se demostró que se puede tener un rendimiento aceptable en la portabilidad de algoritmos paralelos mediante una comparación de bibliotecas similares. Se apoya de un generador de código que transforma el código haciendo uso de `templates` para ser ejecutado por el compilador. Debido a que este último componente implica una sobrecarga adicional a la unidad de procesamiento (ver sección 3.5).

Los resultados demostraron que el uso de una `GPU` no implica automáticamente un mayor rendimiento. Lo anterior se debe al tamaño de la memoria caché de las `CPU`. Su aplicabilidad más amplia a las cargas de trabajo de propósito general puede resultar en un rendimiento general óptimo. En este sentido, las diferencias de rendimiento rara vez son mayores que un factor de dos para las cargas de trabajo típicas del Álgebra Lineal de matrices dispersas cuando se comparan hardware similares en términos de costo y rendimiento energético.

Con base en el algoritmo iterativo de agrupamiento, se analizaron los aspectos particulares de los esquemas iterativos en arquitecturas homogéneas y heterogéneas. Se estudiaron los mecanismos que evalúan cuantitativamente el tiempo de ejecución de este tipo de algoritmos. Debido a que el algoritmo iterativo de agrupamiento comparte el mismo conjunto de patrones de interacción, se seleccionó al mecanismo basado en esqueletos. Para intentar predecir el comportamiento del algoritmo iterativo basado en esqueletos se propuso un modelo matemático que refleja el costo del cómputo y de las comunicaciones. Por lo tanto, se ha propuesto un sistema compuesto de dos subsistemas: (1) Subsistema de cómputo se considera el costo computacional de las operaciones que se ejecutan con base en la biblioteca previamente propuesta; (2) Subsistema de comunicación que modela el tiempo de comunicación entre dos unidades de procesamiento cualesquiera. En este subsistema se ha considerado el tiempo necesario para iniciar una comunicación y el tiempo que se requiere para enviar un mensaje; el objetivo, es entonces minimizar la función objetivo que depende de los parámetros del subsistema de cómputo y el subsistema de comunicación. Los resultados, hicieron evidente que la propuesta de modelos matemáticos con alto grado de abstracción permite entender el comportamiento del algoritmo con mayor independencia de la arquitectura; sin embargo, conforme sea más refinado el modelo será capaz de mejorar las características de la arquitectura, a costa de la pérdida de portabilidad. En tanto, se requiere establecer un compromiso entre un modelo con la suficiente abstracción que permita portabilidad entre las distintas plataformas. El hecho de usar arquitecturas homogéneas supone una ventaja para el desarrollo de técnicas auto-adaptación pues simplifica el estudio teórico y la propuesta de modelos. Sin embargo, los resultados demostraron que en ocasiones se requiere relajar la definición de arquitectura homogénea debido a que en la mayoría de las veces, el rendimiento de las comunicaciones o de procesamiento no será el mismo, incluso cuando se trate de un conjunto de unidades de procesamiento con características idénticas. En este sentido, la topología de la red es un factor importante que afecta el envío y recepción de mensajes.

A lo largo de la metodología se construyó un enfoque de auto-adaptación con el objetivo de minimizar el tiempo computacional de un algoritmo y proporcionar por-

tabilidad del rendimiento en una nueva arquitectura. Cabe mencionar que el diseño de este marco de trabajo ha iniciado desde el análisis algorítmico de las técnicas de agrupamiento enfocadas en el análisis de expresión genética, hasta la propuesta de un modelo matemático que evalúe el costo computacional de un algoritmo en arquitecturas homogéneas y heterogéneas. Se consideraron dos aspectos fundamentales: (1) Representación de la configuración correcta que considere las distintas configuraciones que pueda tener un parámetro; (2) Definición del tamaño del espacio de búsqueda válido en el que se implementaron heurísticas que permiten minimizar la función determinada por el modelo matemático que evalúa el tiempo computacional propuesto. Para validar los componentes del marco de trabajo de auto-adaptación, se ha optado por el modelo `SANS`. En este sentido, el marco de trabajo se compone de un componente inteligente que estudia la estructura de los datos que recibe el algoritmo; un componente de sistema capaz de recopilar información de los recursos de la arquitectura; un componente histórico que almacena una serie de datos que representan las características de hardware y software que describen un problema resuelto con anterioridad; un lenguaje que permite la comunicación entre el usuario y el sistema; un diccionario de los datos que se almacenan en el componente histórico; y un conjunto de bibliotecas compiladas de manera óptima en diferentes arquitecturas. La ejecución del marco de trabajo incluye dos etapas: (1) Instalación, en la que el componente de sistema recaba las características de toda la arquitectura y las almacena en un componente histórico (base de datos). Durante esta fase se ejecutan versiones reducidas de las operaciones elementales en cada unidad de procesamiento que permiten estimar el costo computacional de cada una de ellas; (2) Ejecución, en la que un conjunto de técnicas de búsqueda se ejecutan al mismo tiempo, con un conjunto de configuraciones candidatas. Las técnicas que obtienen buenas configuraciones son ponderadas, mientras que las que no tienen buen rendimiento son deshabilitadas. Cabe mencionar que la propuesta de este marco de trabajo no se limita a la optimización del tiempo computacional, es decir, puede ser adaptado para optimizar el consumo de energía y el uso de la memoria.

Se aplicó el marco de trabajo al análisis de datos de expresión genética construidos *in-silico* e *in-vitro*. Los resultados obtenidos son alentadores. En la mayoría de los casos, el marco de trabajo es capaz de mejorar el rendimiento computacional de los algoritmos y recuperar la cantidad correcta de grupos (K). La metodología propuesta en la sección 5.6 es capaz de asignar al menos el 95 % de los patrones a los grupos correctos. Estos resultados pueden visualizarse en la comparación del índice Rand para cualquier método (Ver las columnas KM, CC_{KM} en las tablas 5.10 y 5.12). Sin embargo, se observó que la metodología obtiene un desempeño menor cuando el tamaño de la muestra es relativamente pequeño con relación al número de clases. Este es un ejemplo de un diseño experimental mal formulado. Por lo tanto, en el análisis de agrupamiento en datos de expresión genética se debe hacer un esfuerzo por recopilar datos que sean lo suficientemente homogéneos en el que se pueda esperar que el número de clases sea razonablemente pequeño, y acorde con el tamaño de la muestra disponible.

Se aplicó la metodología en el análisis de datos de expresión genética derivados del análisis del desarrollo de hepatocarcinomas en ratones. Se ha demostrado que el algo-

ritmo de agrupación optimizado por mecanismos auto-adaptables es capaz de afinar la aparente estabilidad de una partición aleatoria. En este sentido, se podría realizar el estudio que permita comprobar si existe una sobreinterpretación de la estabilidad de la agrupación en un estudio real. Sin embargo, es evidente el trabajo multidisciplinario que le dé un significado biológico a los resultados obtenidos. Esta es una aportación importante, por un lado en el campo de la biología, permite procesar múltiples veces un experimento con una gran cantidad de datos biológicos en un tiempo razonable, mientras que en el campo de las ciencias computacionales, se ha hecho un aporte importante al establecer un marco de trabajo genérico y semi-automático que es capaz de obtener parámetros algorítmicos que tienen casi el mismo rendimiento que las soluciones óptimas para un código paralelizado en una arquitectura específica. Esto le da al usuario final las herramientas necesarias para preocuparse únicamente por su ámbito de investigación y mejora la portabilidad del código en Ingeniería de Software. Además, se tiene la ventaja de que es posible extraer información de soluciones para mejorar la calidad del resultado actual.

Es importante mencionar que los resultados dependerán del algoritmo de agrupación que se utilice. En este caso se utilizaron los algoritmos KM y FCM para realizar los experimentos. Se observó que el marco de trabajo con FCM produce resultados ligeramente mejores que cuando se utilizó KM. Esto refuerza el hecho de que cada algoritmo de agrupamiento tiene sus propias ventajas y limitaciones.

Finalmente, es evidente la necesidad del preprocesamiento de datos con técnicas de normalización de datos o detección de *outliers*. La elección de los métodos de preprocesamiento puede afectar o mejorar los resultados del análisis de agrupamiento. Como se señalo, los datos utilizados estaban normalizados por columnas y filas. También se realizó el análisis con el marco de trabajo utilizando datos no normalizados y los resultados coincidieron en un 37 % con los resultados obtenidos con datos normalizados. Por lo tanto, la selección del algoritmo de agrupamiento, así como la elección de la técnica de preprocesamiento de datos, representan elementos básicos que deben tomarse a consideración al realizar un análisis de este tipo. La evaluación experimental muestra que el algoritmo distribuido descrito en la sección 3.4 es un criterio útil para mejorar el rendimiento de un algoritmo de agrupamiento.

6.1. Trabajo a futuro

Este trabajo ha dejado muchas preguntas abiertas e interesantes de abordar. Aunque los resultados presentados en este trabajo son prometedores, hay algunos puntos que deben abordarse para que el marco de trabajo auto-adaptable sea aplicable a una clase más amplia de problemas bioinformáticos u otras áreas del conocimiento.

El trabajo a futuro ha sido agrupado en seis direcciones:

- **Modelo matemático del tiempo de ejecución.** El trabajo a futuro relacionado a este ámbito se orienta a la mejora de los modelos teóricos que representan el tiempo de ejecución de los algoritmos iterativos tanto en sistemas homogéneos, como en sistemas heterogéneos considerando el «problema del mapeo».

Proponer una solución a este problema es intentar mejorar la asignación de proceso a unidades de procesamiento. Sin embargo, este es un problema NP-completo y por lo tanto, aún no existe un algoritmo de tiempo polinomial que proponga una solución.

- **Esquemas algorítmicos.** Este trabajo está enfocado en esquemas paralelos iterativos, sin embargo, la metodología de auto-adaptación debe ser válida para otros esquemas. Un trabajo a futuro es su aplicación en esquemas de tipo divide y vencerás, *backtracking*, descomposición matricial, maestro - esclavo, por mencionar algunos. En este sentido, se podría considerar, la extensión de la biblioteca a otras técnicas de MD para su aplicación en otras áreas del conocimiento.
  Este trabajo también podría ser aplicado a algoritmos basados en programación dinámica, tal como, la alineación de secuencias. Por lo tanto, se podría indagar en este aspecto para analizar el rendimiento del marco de trabajo.

- **Marco de trabajo de auto-adaptación.** Respecto en lo que al trabajo a futuro se refiere, se propone la extensión a diferentes funciones objetivo tal como, el análisis de energía para arquitecturas paralelas. En este sentido, el trabajo se basa en la propuesta del modelo matemático que refleje el consumo energético de un algoritmo específico debido a que, el marco de trabajo ha sido diseñado para poder admitir distintas funciones objetivo, y que sea el mismo usuario quien decida que desea optimizar.

- **Base de datos.** Actualmente, se manejan dos bases de datos: (1) La base de datos que compone a la biblioteca de operaciones elementales; (2) La base de datos del marco de trabajo. En este sentido, se plantea una única base de datos que permita almacenar información derivada de experimentos realizados de forma descentralizada. Esto podría contribuir de forma positiva a la minimización de las diferentes funciones objetivo debido a que se almacenarían los resultados obtenidos por diferentes experimentos en diferentes arquitecturas de manera general. Esto podría mejorar el hecho del bajo rendimiento cuando el sistema de archivos es relativamente pequeño. Sin embargo, podría suponer poca generalización del marco de trabajo al enfocarlo a la utilización de una sola biblioteca, sin darle al usuario la posibilidad de utilizar BLAS, por mencionar alguna.

- **Interfaz gráfica.** La implementación de un componente que sirva como lenguaje de comunicación entre el marco de trabajo y el usuario representa una motivación para que la propuesta de este trabajo sea ampliamente utilizada por la comunidad científica. Sin embargo, esto conlleva la implementación de un servicio web que permita tener a disposición del usuario una herramienta robusta que utilice los recursos computacionales de una arquitectura para resolver un problema.
  Finalmente, se propone la implementación de un pipeline que integre algoritmos de limpieza y normalización de datos, seguido de la aplicación de estos y otros

algoritmos de MD utilizados dentro de la Bioinformática. Este pipeline debe ser transparente al usuario durante la elección de parámetros.

- **Introducción de métrica de similitud basada en percentiles.** Se propone analizar el rendimiento de los algoritmos utilizando como métrica de similitud los percentiles. En este sentido, se pretende indagar la densidad de los datos entre los cuartiles.

Esta página se dejó en blanco intencionalmente.

Bibliografía

[1] Abdelfattah, A., Haidar, A., Tomov, S., and Dongarra, J. . Performance, design, and autotuning of batched gemm for gpus. In *International Conference on High Performance Computing*, pages 21–38. Springer, 2016.

[2] Agullo, E., Demmel, J., Dongarra, J., Hadri, B., Kurzak, J., Langou, J., ... and Tomov, S. Numerical linear algebra on emerging architectures: The plasma and magma projects. In *Journal of Physics: Conference Series*, volume 180, page 012037. IOP Publishing, 2009.

[3] Alexandrov, A., Ionescu, M. F., Schauser, K. E., and Scheiman, C. Loggp: Incorporating long messages into the logp model for parallel computation. *Journal of parallel and distributed computing*, 44(1):71–79, 1997.

[4] Almeida, F., Gonzalez, D., Moreno, L. M., and Rodriguez, C. Pipelines on heterogeneous systems: models and tools. *Concurrency and Computation: Practice and Experience*, 17(9):1173–1195, 2005.

[5] Almeida, F., González, D., and Moreno, L. M. . The master–slave paradigm on heterogeneous systems: A dynamic programming approach for the optimal mapping. *Journal of Systems Architecture*, 52(2):105–116, 2006.

[6] Alpar, O., and Krejcar, O. Detection of irregular thermoregulation in hand thermography by fuzzy C-means. In *International Conference on Bioinformatics and Biomedical Engineering*, pages 255–265. Springer, 2018.

[7] Ansel, J., Chan, C., Wong, Y. L., Olszewski, M., Zhao, Q., Edelman, A., and Amarasinghe, S. PetaBricks: a language and compiler for algorithmic choice. *ACM Sigplan Notices*, 44(6):38–49, 2009.

[8] Ansel, J., Pacula, M., Amarasinghe, S., and O'Reilly, U. M. . An efficient evolutionary algorithm for solving bottom up problems. In *Annual Conference on Genetic and Evolutionary Computation, Dublin, Ireland*, volume 10, 2011.

[9] Arthur, D., Manthey, B., and Roglin, H. K-means has polynomial smoothed complexity. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 405–414. IEEE, 2009.

[10] Backus, J. Can programming be liberated from the von neumann style? a functional style and its algebra of programs. *Communications of the ACM*, 21(8):613–641, 1978.

[11] Bagbaba, A. Improving collective I/O performance with machine learning supported auto-tuning. In *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 814–821. IEEE, 2020.

[12] Balaprakash, P., Dongarra, J., Gamblin, T., Hall, M., Hollingsworth, J. K., Norris, B., and Vuduc, R. Autotuning in high-performance computing applications. *Proceedings of the IEEE*, 106(11):2068–2083, 2018.

[13] Banino, C., Beaumont, O., Carter, L., Ferrante, J., Legrand, A., and Robert, Y. Scheduling strategies for master-slave tasking on heterogeneous processor platforms. *IEEE Transactions on Parallel and Distributed Systems*, 15(4):319–330, 2004.

[14] Bar-Noy, A., and Kipnis, S. . Designing broadcasting algorithms in the postal model for message-passing systems. *Mathematical systems theory*, 27(5):431–452, 1994.

[15] Bateni, M., Behnezhad, S., Derakhshan, M., Hajiaghayi, M., Kiveris, R., Lattanzi, S., and Mirrokni, V. . Affinity clustering: Hierarchical clustering at scale. In *Advances in Neural Information Processing Systems*, pages 6864–6874, 2017.

[16] BBigger, C. B., Guerra, B., Brasky, K. M., Hubbard, G., Beard, M. R., Luxon, B. A., and Lanford, R. E. Intrahepatic gene expression during chronic hepatitis C virus infection in chimpanzees. *Journal of virology*, 78(24):13779–13792, 2004.

[17] Ben-Hur, A., Elisseeff, A., and Guyon, I. A stability based method for discovering structure in clustered data. In *Biocomputing 2002*, pages 6–17. World Scientific, 2001.

[18] Benoit, A., Dobrila, A., Nicod, J. M., and Philippe, L. Workload balancing and throughput optimization for heterogeneous systems subject to failures. In *European Conference on Parallel Processing*, pages 242–254. Springer, 2011.

[19] Blackford, L. S., Petitet, A., Pozo, R., Remington, K., Whaley, R. C., Demmel, J., ... and Heroux, M. . An updated set of basic linear algebra subprograms (blas). *ACM Transactions on Mathematical Software*, 28(2):135–151, 2002.

[20] Blackford, S., Corliss, G., Demmel, J., Dongarra, J., Duff, I., Hammarling, S., ... and Kaufmann, L. . Basic linear algebra subprograms technical (blast) forum standard. *Int. J. High Perform. Comput*, 15:3–4, 2001.

[21] Bleuler, S., Prelic, A., and Zitzler, E. . An EA framework for biclustering of gene expression data. In *Evolutionary Computation, 2004. CEC2004. Congress on*, volume 1, pages 166–173. IEEE, 2004.

[22] Boguski, M. S., Lowe, T. M., and Tolstoshev, C. M. dbEST: database for expressed sequence tags. *Nature genetics*, 4(4):332–333, 1993.

[23] Brazas, M. D., Yim, D. S., Yamada, J. T., and Ouellette, B. F. The 2011 bioinformatics links directory update: more resources, tools and databases and features to empower the bioinformatics community. *Nucleic acids research*, 39(suppl 2):W3–W7, 2011.

[24] Brewer, E. A. *Portable high-performance supercomputing: high-level platform-dependent optimization*. PhD thesis, Citeseer, 1994.

[25] Brown, M. P., Grundy, W. N., Lin, D., Cristianini, N., Sugnet, C. W., Furey, T. S., ... and Haussler, D. Knowledge-based analysis of microarray gene expression data by using support vector machines. *Proceedings of the National Academy of Sciences*, 97(1):262–267, 2000.

[26] Bullard, J. H., Purdom, E., Hansen, K. D., and Dudoit, S. Evaluation of statistical methods for normalization and differential expression in mRNA−Seq experiments. *BMC bioinformatics*, 11(1):94, 2010.

[27] Bulut, H., Onan, A., and Korukoğlu, S. An improved ant-based algorithm based on heaps merging and fuzzy c-means for clustering cancer gene expression data. *Sādhanā*, 45(1):1–17, 2020.

[28] Busquets, X., and Agustí, A. G. N. Chip genético (ADN *array)*: el futuro ya está aquí. *Archivos de Bronconeumología*, 37(9):394–396, 2001.

[29] Campbell, D. K. A survey of models of parallel computation. *Report-University Of York Department Of Computer Science YCS*, 1997.

[30] Cancer Genome Atlas Research Network and others. Integrated genomic analyses of ovarian carcinoma. *Nature*, 474(7353):609–615, 2011.

[31] Cannon, R. L., Dave, J. V., and Bezdek, J. C. . Efficient implementation of the fuzzy c-means clustering algorithms. *IEEE transactions on pattern analysis and machine intelligence*, (2):248–255, 1986.

[32] Castro, A. L. . *Optimización de algoritmos bioinspirados en sistemas heterogéneos CPU-GPU*. PhD thesis, Universidad Católica San Antonio de Murcia, 2016.

[33] Chandra, R., Dagum, L., Kohr, D., Menon, R., Maydan, D., and McDonald, J. *Parallel programming in OpenMP*. Morgan kaufmann, 2001.

[34] Chen, J. J. Key aspects of analyzing microarray gene-expression data. (5):473–482, 2007.

[35] Chen, W., Xie, G., Li, R., Bai, Y., Fan, C., and Li, K. Efficient task scheduling for budget constrained parallel applications on heterogeneous cloud computing systems. *Future Generation Computer Systems*, 74:1–11, 2017.

[36] Cheng, S., Ashley, J., Kurleto, J. D., Lobb-Rabe, M., Park, Y. J., Carrillo, R. A., and Ozkan, E. . Molecular basis of synaptic specificity by immunoglobulin superfamily receptors in drosophila. *Elife*, 8:e41028, 2019.

[37] Cheng, Y., and Church, G. M. Biclustering of expression data. In *Ismb*, volume 8, pages 93–103, 2000.

[38] Cho, H., Dhillon, I. S., Guan, Y., and Sra, S. Minimum Sum-Squared Residue Co-Clustering of Gene Expression Data. In *SDM*, volume 3, page 3. SIAM, 2004.

[39] Christen, M., Schenk, O., and Burkhart, H. . Patus: A code generation and autotuning framework for parallel iterative stencil computations on modern microarchitectures. In *2011 IEEE International Parallel & Distributed Processing Symposium*, pages 676–687. IEEE, 2011.

[40] Churchill, G. A., Airey, D. C., Allayee, H., Angel, J. M., Attie, A. D., Beatty, J., and Kempermann, G. The Collaborative Cross, a community resource for the genetic analysis of complex traits. *Nature genetics*, 36(11):1133–1137, 2004.

[41] Clarke, T. H., Brinkac, L. M., Sutton, G., and Fouts, D. E. . GGRaSP: a R-package for selecting representative genomes using Gaussian mixture models. *Bioinformatics*, 34(17):3032–3034, 2018.

[42] Cohen, J. Bioinformatics−an introduction for computer scientists. *ACM Computing Surveys (CSUR)*, 36(2):122–158, 2004.

[43] Cole, M. I. *Algorithmic skeletons: structured management of parallel computation*. Pitman London, 1989.

[44] Cole, M. I. Algorithmic skeletons. In *Research Directions in Parallel Functional Programming*, pages 289–303. Springer, 1999.

[45] Coulouris, G. F., Dollimore, J., and Kindberg, T. . *Distributed systems: concepts and design*. pearson education, 2005.

[46] Crick, F. H. . Central dogma of molecular biology. *Nature*, 227(5258):561–563, 1970.

[47] Crick, F. H. On protein synthesis. In *Symp Soc Exp Biol*, volume 12, page 8, 1958.

[48] Cruz, R. A., Bentes, C., Breder, B., Vasconcellos, E., Clua, E., de Carvalho, P. M., and Drummond, L. M. Maximizing the gpu resource usage by reordering concurrent kernels submission. *Concurrency and Computation: Practice and Experience*, 31(18):e4409, 2019.

[49] Culler, D., Karp, R., Patterson, D., Sahay, A., Schauser, K. E., Santos, E., ... and Von Eicken, T. . Logp: Towards a realistic model of parallel computation. In *Proceedings of the fourth ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 1–12, 1993.

[50] A. C. Cusp. Templated library for sparse linear algebra on cuda, 2010.

[51] Dackland, K., and Kågström, B. . An hierarchical approach for performance analysis of scalapack-based routines using the distributed linear algebra machine. In *International Workshop on Applied Parallel Computing*, pages 186–195. Springer, 1996.

[52] Dahm, R. Discovering dna: Friedrich miescher and the early years of nucleic acid research. *Human genetics*, 122(6):565–581, 2008.

[53] Dawkins, R. . El gen egoísta [1976]. *Barcelona: Salvat*, 2002.

[54] De Amorim, R. C., and Mirkin, B. Minkowski metric, feature weighting and anomalous cluster initializing in k-means clustering. *Pattern Recognition*, 45(3):1061–1075, 2012.

[55] De la Torre, P., and Kruskal, C. P. Submachine locality in the bulk synchronous setting. In *European Conference on Parallel Processing*, pages 352–358. Springer, 1996.

[56] DeRisi, J., Penland, L., Brown, P. O., Bittner, M. L., Meltzer, P. S., Ray, M., ... and Trent, J. M. Use of a cDNA microarray to analyse gene expression patterns in human cancer. *Nature genetics*, 14(4):457–460, 1996.

[57] D'haeseleer, P. How does gene expression clustering work? *Nature biotechnology*, 23(12):1499–1501, 2005.

[58] Dongarra, J., & Eijkhout, V. . Self-adapting numerical software (SANS) effort. *IBM Journal of Research and Development*, 50(2.3):223–238, 2006.

[59] Dongarra, J. J., and Whaley, R. C. Lapack working note 94 a user's guide to the blacs v1. *Tech.£ eport*, 1997.

[60] Dongarra, J. J., Du Croz, J., Hammarling, S., and Duff, I. S. . A set of level 3 basic linear algebra subprograms. *ACM Transactions on Mathematical Software (TOMS)*, 16(1):1–17, 1990.

[61] Dongarra, J. J., Meuer, H. W., and Strohmaier, E. TOP500 supercomputer sites. *Supercomputer*, 13:89–111, 1997.

[62] Doroshenko, A., Ivanenko, P. A., Novak, O., and Yatsenko, O. Optimization of parallel software tuning with statistical modeling and machine learning. In *ICTERI*, pages 219–226, 2018.

[63] Doroshenko, Y., Ivanenko, A., Novak, S., and Yatsenko, A. . Parallel software auto-tuning using statistical modeling and machine learning. *PROBLEMS IN PROGRAMMING*, (2-3):46–53, 2018.

[64] Dudoit, S., and Fridlyand, J. A prediction-based resampling method for estimating the number of clusters in a dataset. *Genome biology*, 3(7):research0036–1, 2002.

[65] Duijn, M. V. Extending the bsp model to hierarchical heterogeneous architectures. Master's thesis, 2018.

[66] Edgar, R., Domrachev, M., and Lash, A. E. Gene Expression Omnibus: NCBI gene expression and hybridization array data repository. *Nucleic acids research*, 30(1):207–210, 2002.

[67] Fan, X. Optimize your code: Matrix multiplication. *ht tps://tinyurl. com/-kuvzbp9*, 2009.

[68] Farber, R. *CUDA application design and development*. Elsevier, 2011.

[69] Fayyad, U. M., Haussler, D., and Stolorz, P. E. . KDD for Science Data Analysis: Issues and Examples. In *KDD*, pages 50–56, 1996.

[70] Flatt, H. P., and Kennedy, K. . Performance of parallel processors. *Parallel Computing*, 12(1):1–20, 1989.

[71] Foster, I. . *Designing and building parallel programs: concepts and tools for parallel software engineering*. Addison-Wesley Longman Publishing Co., Inc., 1995.

[72] Frigo, M., and Johnson, S. G. Fftw: An adaptive software architecture for the fft. In *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP'98 (Cat. No. 98CH36181)*, volume 3, pages 1381–1384. IEEE, 1998.

[73] Frigui, H., and Krishnapuram, R. . Clustering by competitive agglomeration. *Pattern recognition*, 30(7):1109–1119, 1997.

[74] GANOW, G. Possible relation between deox-yribonucleic acid and protein structure. *Nature*, 173:318, 1954.

[75] Garcia Pinto, V., Mello Schnorr, L., Stanisic, L., Legrand, A., Thibault, S., and Danjean, V. A visual performance analysis framework for task-based parallel applications running on hybrid clusters. *Concurrency and Computation: Practice and Experience*, 30(18):e4472, 2018.

[76] Ge, Z., Song, Z., Ding, S. X., and Huang, B. Data mining and analytics in the process industry: The role of machine learning. *Ieee Access*, 5:20590–20616, 2017.

[77] Geng, W., Cosman, P., Baek, J. H., Berry, C. C., and Schafer, W. R. Quantitative classification and natural clustering of Caenorhabditis elegans behavioral phenotypes. *Genetics*, 165(3):1117–1126, 2003.

[78] Gonzalez, J. A., Leon, C., Piccoli, F., Printista, M., Roda, J. L., Rodríguez, C., and de Sande, F. Oblivious bsp. In *European Conference on Parallel Processing*, pages 682–685. Springer, 2000.

[79] González, D., Almeida, F., Moreno, L., and Rodrıguez, C. . Towards the automatic optimal mapping of pipeline algorithms. *Parallel Computing*, 29(2):241–254, 2003.

[80] González, D., Almeida, F., Roda, J., and Rodriguez, C. . From the theory to the tools: parallel dynamic programming. *Concurrency: practice and experience*, 12(1):21–34, 2000.

[81] González-Domínguez, J., and Expósito, R. R. Accelerating binary biclustering on platforms with cuda-enabled gpus. *Information Sciences*, 496:317–325, 2019.

[82] Goudreau, M., Lang, K., Rao, S., Suel, T., and Tsantilas, T. . Towards efficiency and portability: Programming with the bsp model. In *Proceedings of the eighth annual ACM symposium on Parallel algorithms and architectures*, pages 1–12, 1996.

[83] Goudreau, M. W., Lang, K., Rao, S. B., Suel, T., and Tsantilas, T. . Portable and efficient parallel computing using the bsp model. *IEEE Transactions on Computers*, 48(7):670–689, 1999.

[84] Greg, H., and Elkan, C. Alternative to the k-means algorithm that find better clustering. In *ACM Conference on Information and knowledge Management (CIKM 2002)*, pages 600–607, 2002.

[85] Gremse, F., Hofter, A., Schwen, L. O., Kiessling, F., and Naumann, U. . GPU-accelerated sparse matrix-matrix multiplication by iterative row merging. *SIAM Journal on Scientific Computing*, 37(1):C54–C71, 2015.

[86] Griffith, M., Griffith, O. L., Mwenifumbo, J., Goya, R., Morrissy, A. S., Morin, R. D., Corbett, R., Tang, M. J., Trevor, J. and Marra, M. A. Alternative expression analysis by RNA sequencing. *Nature methods*, 7(10):843–847, 2010.

[87] Gropp, W., Gropp, W. D., Lusk, E., Skjellum, A., and Lusk, A. D. F. E. E. . *Using MPI: portable parallel programming with the message-passing interface*, volume 1. MIT press, 1999.

[88] Hamerly, G., and Elkan, C. Alternatives to the k-means algorithm that find better clusterings. In *Proceedings of the eleventh international conference on Information and knowledge management*, pages 600–607, 2002.

[89] Hamerly, G. J., and Elkan, C. P. *Learning structure and concepts in data through data clustering*. PhD thesis, University of California, San Diego, 2003.

[90] Han, H. A novel feature selection for rna-seq analysis. *Computational biology and chemistry*, 71:245–257, 2017.

[91] Hannenhalli, S. Eukaryotic transcription factor binding sites−modeling and integrative search methods. *Bioinformatics*, 24(11):1325–1331, 2008.

[92] Hardcastle, T. J., and Kelly, K. A. . bayseq: empirical bayesian methods for identifying differential expression in sequence count data. *BMC bioinformatics*, 11(1):422, 2010.

[93] Hatem, A., Bozdağ, D., Toland, A. E. and Çatalyürek, Ü. V. Benchmarking short sequence mapping tools. *BMC bioinformatics*, 14(1):184, 2013.

[94] Hindorff, L. A., Junkins, H. A., Mehta, J. P., and Manolio, T. A. A catalog of published genome-wide association studies. *National Human Genome Research Institute*, 2010.

[95] Hockney, R. W., and Jesshope, C. R. *Parallel Computers 2: architecture, programming and algorithms*. CRC Press, 2019.

[96] Howe, E. A., Sinha, R., Schlauch, D., and Quackenbush, J. Rna-seq analysis in mev. *Bioinformatics*, 27(22):3209–3210, 2011.

[97] Hsu, C. C., Huang, Y. P., and Chang, K. W. i. Extended Naive Bayes classifier for mixed data. *Expert Systems with Applications*, 35(3):1080–1083, 2008.

[98] Huezo, M. S. G., Ávila, J. F. S., de Gastroenterología, A. M., de Radiología, S. M., de Oncología, S. M., and de Carcinoma, G. M. D. C. Consenso mexicano de diagnóstico y manejo del carcinoma hepatocelular. *Revista de Gastroenterología de México*, 79(4):250–262, 2014.

[99] Huss-Lederman, S., Jacobson, E. M., Johnson, J. R., Tsao, A., and Turnbull, T. . Implementation of Strassen's algorithm for matrix multiplication. In *Supercomputing'96: Proceedings of the 1996 ACM/IEEE Conference on Supercomputing*, pages 32–32. IEEE, 1996.

[100] Hussain, H. M., Benkrid, K., Seker, H., and Erdogan, A. T. . Fpga implementation of k-means algorithm for bioinformatics application: An accelerated approach to clustering microarray data. In *2011 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pages 248–255. IEEE, 2011.

[101] Hutter, F., Xu, L., Hoos, H. H., and Leyton-Brown, K. . Algorithm runtime prediction: Methods and evaluation. *Artificial Intelligence*, 206:79–111, 2014.

[102] Ino, F., Fujimoto, N., and Hagihara, K. Loggps: a parallel computational model for synchronization analysis. In *Proceedings of the eighth ACM SIGPLAN symposium on Principles and practices of parallel programming*, pages 133–142, 2001.

[103] Jain, M. Next−generation sequencing technologies for gene expression profiling in plants. *Briefings in functional genomics*, 11(1):63–70, 2012.

[104] Jain, M., and Murty, M. N. . Flynn: Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[105] Jakobsson, A. Automatic cost analysis for imperative bsp programs. *International Journal of Parallel Programming*, 47(2):184–212, 2019.

[106] Johnson, S. C. . Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254, 1967.

[107] Jordan, H., Pellegrini, S., Thoman, P., Kofler, K., and Fahringer, T. INSPIRE: The Insieme parallel intermediate representation. In *Proceedings of the 22nd international conference on Parallel architectures and compilation techniques*, pages 7–17. IEEE, 2013.

[108] Jose, J. T., Zachariah, U., Lijo, V. P., Gnanasigamani, L. J., and Mathew, J. Case Study on Enhanced K-Means Algorithm for Bioinformatics Data Clustering. *International Journal of Applied Engineering Research*, 12(24):15147–15151, 2017.

[109] Juhász, S., & Charaf, H. Execution time prediction for parallel data processing tasks. In *Parallel, Distributed and Network-based Processing, 2002. Proceedings. 10th Euromicro Workshop on*, pages 31–38. IEEE, 2002.

[110] Juurlink, B. H., and Wijshoff, H. A. A quantitative comparison of parallel computation models. *ACM Transactions on Computer Systems (TOCS)*, 16(3):271–318, 1998.

[111] Juurlink, B. H. H., and Wijshoff, H. A. G. The e-bsp model: Incorporating unbalanced communication and general locality into the bsp model. In *Proc. of the Annual International EURO-PAR Conference, LNCS*, pages 339–347, 1995.

[112] Kent, W. J. BLAT−the BLAST−like alignment tool. *Genome research*, 12(4):656–664, 2002.

[113] Kerschke, P., Hoos, H. H., Neumann, F., and Trautmann, H. Automated algorithm selection: Survey and perspectives. *Evolutionary computation*, 27(1):3–45, 2019.

[114] Khan, A., Katanic, D., and Thakar, J. . Meta-analysis of cell-specific transcriptomic data using fuzzy c-means clustering discovers versatile viral responsive genes. *BMC bioinformatics*, 18(1):295, 2017.

[115] Khatri, P., and Draghici, S. Ontological analysis of gene expression data: current tools, limitations, and open problems. *Bioinformatics*, 21(18):3587–3595, 2005.

[116] Khorana, H. G., Büuchi, H., Ghosh, H., Gupta, N., Jacob, T. M., Kössel, H., ... and Wells, R. D. . Polynucleotide synthesis and the genetic code. In *Cold Spring Harbor symposia on quantitative biology*, volume 31, pages 39–49. Cold Spring Harbor Laboratory Press, 1966.

[117] Kielmann, T., Bal, H. E., and Gorlatch, S. Bandwidth-efficient collective communication for clustered wide area systems. In *Proceedings 14th International Parallel and Distributed Processing Symposium. IPDPS 2000*, pages 492–499. IEEE, 2000.

[118] Kielmann, T., Bal, H. E., and Verstoep, K. Fast measurement of logp parameters for message passing platforms. In *International Parallel and Distributed Processing Symposium*, pages 1176–1183. Springer, 2000.

[119] Kimura, M. Model of effectively neutral mutations in which selective constraint is incorporated. *Proceedings of the National Academy of Sciences*, 76(7):3440–3444, 1979.

[120] Kirk, D. B., and Wen-Mei, W. H. *Programming massively parallel processors: a hands-on approach*. Morgan kaufmann, 2016.

[121] Kuchen, H., and Cole, M. The integration of task and data parallel skeletons. *Parallel Processing Letters*, 12(02):141–155, 2002.

[122] Kumar, S. P., Sumithra, M. G., and Saranya, N. Artificial bee colony-based fuzzy c means (ABC-FCM) segmentation algorithm and dimensionality reduction for leaf disease detection in bioinformatics. *The Journal of Supercomputing*, 75(12):8293–8311, 2019.

[123] Lan, K., Wang, D. T., Fong, S., Liu, L. S., Wong, K. K., and Dey, N. A survey of data mining and deep learning in bioinformatics. *Journal of medical systems*, 42(8):139, 2018.

[124] Langenberg, R. *Analysis and Optimization of the Offline Software of the ATLAS Experiment at CERN*. PhD thesis, Technische Universität München, 2017.

[125] Langmead, B. Aligning short sequencing reads with bowtie. *Current protocols in bioinformatics*, 32(1):11–7, 2010.

[126] Lattanzi, S., and Lavastida, T. A framework for parallelizing hierarchical clustering methods. In *European Conference on Machine Learning*, 2019.

[127] Lattanzi, S., and Sohler, C. . A better k-means++ algorithm via local search. In *International Conference on Machine Learning*, pages 3662–3671, 2019.

[128] Lausted, C., Dahl, T., Warren, C., King, K., Smith, K., Johnson, M., ... and Lasky, S. R. . POSaM: a fast, flexible, open-source, inkjet oligonucleotide synthesizer and microarrayer. *Genome Biology*, 5(8):R58, 2004.

[129] P. A. Levene and L. W Bass. *Nucleic acids*. Chemical Catalog Company, Inc.; New York, 1931.

[130] Li, J., Seo, B., and Lin, L. . Optimal transport, mean partition, and uncertainty assessment in cluster analysis. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 12(5):359–377, 2019.

[131] Li, R., Li, Y., Kristiansen, K., and Wang, J. . Soap: short oligonucleotide alignment program. *Bioinformatics*, 24(5):713–714, 2008.

[132] Li, Y., Zhao, K., Chu, X., and Liu, J. Speeding up k-means algorithm by gpus. *Journal of Computer and System Sciences*, 79(2):216–229, 2013.

[133] Lin, C. R., and Chen, M. S. . Combining partitional and hierarchical algorithms for robust and efficient data clustering with cohesion self-merging. *IEEE Transactions on Knowledge and Data Engineering*, 17(2):145–159, 2005.

[134] Lin, X., and Chen, Y. Identification of potentially functional CircRNA-miRNA-mRNA regulatory network in hepatocellular carcinoma by integrated microarray analysis. *Medical science monitor basic research*, 24:70, 2018.

[135] Liu, B., Xin, Y., Cheung, R. C., and Yan, H. Gpu-based biclustering for microarray data analysis in neurocomputing. *Neurocomputing*, 134:239–246, 2014.

[136] Liu, W., and Chen, L. A parallel algorithm for gene expressing data biclustering. *JCP*, 3(10):71–77, 2008.

[137] Liu, Z., Song, Y. Q., Xie, C. H., and Tang, Z. A new clustering method of gene expression data based on multivariate Gaussian mixture models. *Signal, Image and Video Processing*, 10(2):359–368, 2016.

[138] Lunter, G., and Goodson, M. Stampy: a statistical algorithm for sensitive and fast mapping of illumina sequence reads. *Genome research*, 21(6):936–939, 2011.

[139] Luscombe, N. M., Greenbaum, D., and Gerstein, M. What is bioinformatics? an introduction and overview. *Yearbook of Medical Informatics*, 1(1):83–99, 2001.

[140] MacQueen, J. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967.

[141] Madeira, S. C., and Oliveira, A. L. A linear time biclustering algorithm for time series gene expression data. In *Algorithms in Bioinformatics*, pages 39–52. Springer, 2005.

[142] Madhulatha, T. S. . An overview on clustering methods. *arXiv preprint arXiv:1205.1117*, 2012.

[143] Mahapatra, S., Dash, R. R., and Pradhan, S. K. A heuristic for scheduling of uniform parallel processors. In *2016 2nd International Conference on Computational Intelligence and Networks (CINE)*, pages 78–83. IEEE, 2016.

[144] Maintainer, M. B. P., GenomicRanges, B., and Rsamtools, B. Package 'genomicalignments'. 2015.

[145] Marini, F., and Walczak, B. . Particle swarm optimization (pso). a tutorial. *Chemometrics and Intelligent Laboratory Systems*, 149:153–165, 2015.

[146] Marquer, Y., and Gava, F. . An axiomatization for bsp algorithms. In *International Conference on Algorithms and Architectures for Parallel Processing*, pages 72–88. Springer, 2018.

[147] McDowell, I. C., Manandhar, D., Vockley, C. M., Schmid, A. K., Reddy, T. E., and Engelhardt, B. E. Clustering gene expression time series data using an infinite Gaussian process mixture model. *PLoS computational biology*, 14(1):e1005896, 2018.

[148] Mi, S., Lu, J., Sun, M., Li, Z., Zhang, H., Neilly, M. B., ... and Bohlander, S. K. MicroRNA expression signatures accurately discriminate acute lymphoblastic leukemia from acute myeloid leukemia. *Proceedings of the National Academy of Sciences*, 104(50):19971–19976, 2007.

[149] Mitra, S., and Hayashi, Y. Bioinformatics with soft computing. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 36(5):616–635, 2006.

[150] Mo, Y., and Murray, R. M. Privacy preserving average consensus. *IEEE Transactions on Automatic Control*, 62(2):753–765, 2016.

[151] Mohanavalli, S., Jaisakthi, S. M., and Aravindan, C. . Strategies for parallelizing kmeans data clustering algorithm. In *International Conference on Advances in Information Technology and Mobile Communication*, pages 427–430. Springer, 2011.

[152] Monti, S., Tamayo, P., Mesirov, J., and Golub, T. Consensus clustering: a resampling-based method for class discovery and visualization of gene expression microarray data. *Machine learning*, 52(1-2):91–118, 2003.

[153] Morales, D., Roda, J., Almeida, F., Rodríguez, C., and Garcia, F. Integral knapsack problems: Parallel algorithms and their implementations on distributed systems. In *Proceedings of the 9th International Conference on Supercomputing*, pages 218–226, 1995.

[154] Mullapudi, R. T., Adams, A., Sharlet, D., Ragan-Kelley, J., and Fatahalian, K. Automatically scheduling halide image processing pipelines. *ACM Transactions on Graphics (TOG)*, 35(4):1–11, 2016.

[155] Muskinja, M., Chapman, J. D., Duehrssen, M., and Gray, H. . Geant4 performance optimization in the atlas experiment. Technical report, ATL-COM-SOFT-2019-098, 2019.

[156] Naik, D. S. B., Kumar, S. D., and Ramakrishna, S. V. Parallel processing of enhanced k-means using openmp. In *2013 IEEE International Conference on Computational Intelligence and Computing Research*, pages 1–4. IEEE, 2013.

[157] Najar, F., Bourouis, S., Bouguila, N., and Belghith, S. A comparison between different gaussian-based mixture models. In *2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA)*, pages 704–708. IEEE, 2017.

[158] Nisar, A., Ahmad, W., Liao, W. K., and Choudhary, A. High performance parallel/distributed biclustering using barycenter heuristic. In *Proceedings of the 2009 SIAM International Conference on Data Mining*, pages 1050–1061. SIAM, 2009.

[159] Nita, S. L., and Mihailescu, M. . Parallelism and concurrency. In *Haskell Quick Syntax Reference*, pages 131–135. Springer, 2019.

[160] Ohta, T. Evolution by nearly-neutral mutations. In *Mutation and Evolution*, pages 83–90. Springer, 1998.

[161] Omran, M. G., Engelbrecht, A. P., and Salman, A. An overview of clustering methods. *Intelligent Data Analysis*, 11(6):583–605, 2007.

[162] Orzechowski, P., and Boryczko, K. . Rough assessment of gpu capabilities for parallel pcc-based biclustering method applied to microarray data sets. *Bio-Algorithms and Med-Systems*, 11(4):243–248, 2015.

[163] Orzechowski, P., Sipper, M., Huang, X., and Moore, J. H. Ebic: an evolutionary-based parallel biclustering algorithm for pattern discovery. *Bioinformatics*, 34(21):3719–3726, 2018.

[164] Ospovat, D. *The development of Darwin's theory: Natural history, natural theology, and natural selection, 1838-1859*. Cambridge University Press, 1995.

[165] Palamadai Natarajan, E., Mehri Dehnavi, M., and Leiserson, C. Autotuning divide-and-conquer stencil computations. *Concurrency and Computation: Practice and Experience*, 29(17):e4127, 2017.

[166] Pancake, C. Is parallelism for you? *IEEE Computational Science and Engineering*, 3(2):18–37, 1996.

[167] Parkinson, H., Kapushesky, M., Shojatalab, M., Abeygunawardena, N., Coulson, R., Farne, A., and Brazma, A. ArrayExpress—a public database of microarray experiments and gene expression profiles. *Nucleic acids research*, 35(suppl 1):D747–D750, 2007.

[168] Pearson, W. R., and Lipman, D. J. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences*, 85(8):2444–2448, 1988.

[169] Pelagatti, S. . *Structured development of parallel programs*, volume 102. Taylor & Francis London, 1998.

[170] Pelagatti, S. . Task and data parallelism in p3l. In *Patterns and skeletons for parallel and distributed computing*, pages 155–186. Springer, 2003.

[171] Peláez, I., Almeida, F., and Suárez, F. . Dpskel: A skeleton based tool for parallel dynamic programming. In *International Conference on Parallel Processing and Applied Mathematics*, pages 1104–1113. Springer, 2007.

[172] Perezleo-Solórzano, L., Arencibia-Jorge, R., Conill-González, C., Achón-Veloz, G., and Araújo Ruiz, J. A. . Impacto de la bioinformática en las ciencias biomédicas. *Acimed*, 11(4):0–0, 2003.

[173] Pevsner, J. *Bioinformatics and Functional Genomics*. Wiley, 2013.

[174] Piatetsky-Shapiro, G., Khabaza, T., and Ramaswamy, S. Capturing best practice for microarray gene expression data analysis. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 407–415. ACM, 2003.

[175] Pontes, B., Giráldez, R., Divina, F., and Martínez-Álvarez, F. Evaluación de biclusters en un entorno evolutivo. *IV Taller nacional de minería de datos y aprendizaje (TAMIDA)*, pages 1–10, 2007.

[176] Püschel, M., Moura, J. M., Singer, B., Xiong, J., Johnson, J., Padua, D., ... & Johnson, R. W. Spiral: A Generator for Platform-Adapted Libraries of Signal Processing Algorithms. *The International Journal of High Performance Computing Applications*, 18(1):21–45, 2004.

[177] Pérez-Carreón, J. I., López-García, C., Fattel-Fazenda, S., Arce-Popoca, E., Alemán-Lazarini, L., Hernández-García, S., ... and Villa-Trevino, S. . Gene expression profile related to the progression of preneoplastic nodules toward hepatocellular carcinoma in rats. *Neoplasia*, 8(5):373–IN7, 2006.

[178] Quackenbush, J. Microarray data normalization and transformation. *Nature genetics*, 32:496–501, 2002.

[179] Rabhi, F. *Patterns and skeletons for parallel and distributed computing*. Springer Science & Business Media, 2003.

[180] Rehm, B. Bioinformatic tools for DNA/protein sequence analysis, functional assignment of genes and protein classification. *Applied microbiology and biotechnology*, 57(5-6):579–592, 2001.

[181] Remli, M. A., Daud, K. M., Nies, H. W., Mohamad, M. S., Deris, S., Omatu, S., ... and Sulong, G. K-means clustering with infinite feature selection for classification tasks in gene expression data. In *International Conference on Practical Applications of Computational Biology & Bioinformatics*, pages 50–57. Springer, 2017.

[182] Robertson, G., Schein, J., Chiu, R., Corbett, R., Field, M., Jackman, S. D., Mungall, K., Lee, S., Okada, H. M., and Birol, I. De novo assembly and analysis of RNA−Seq data. *Nature methods*, 7(11):909–912, 2010.

[183] Robles, J. A., Qureshi, S. E., Stephen, S. J., Wilson, S. R., Burden, C. J., and Taylor, J. M. Efficient experimental design and analysis strategies for the detection of differential expression using RNA-Sequencing. *BMC genomics*, 13(1):484, 2012.

[184] Rojas Ospina, D. A. Comparación de genes que codifican resistencia antibiótica en mycobacterium avium complex y mycobacterium tuberculosis complex. *Departamento de Ingeniería de Sistemas e Industrial*.

[185] Rossi, F., Van-Beek, P. and Walsh, T. *Handbook of Constraint Programming*. Foundations of Artificial Intelligence. Elsevier Science, 2006.

[186] Ruiz, R., Aguilar-Ruiz, J. S., Riquelme, J. C., and Díaz-Díaz, N. Analysis of feature rankings for classification. In *Advances in Intelligent Data Analysis VI*, pages 362–372. Springer, 2005.

[187] Sagi, O., and Rokach, L. Ensemble learning: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 8(4):e1249, 2018.

[188] Sanderson, C., and Curtin, R. . An open source C++ implementation of multi-threaded Gaussian mixture models, k-means and expectation maximisation. In *2017 11th International Conference on Signal Processing and Communication Systems (ICSPCS)*, pages 1–8. IEEE, 2017.

[189] Schena, M., Shalon, D., Davis, R. W., and Brown, P. O. Quantitative monitoring of gene expression patterns with a complementary DNA microarray. *Science*, 270(5235):467–470, 1995.

[190] Schulze-Lefert, P. . Plant immunity: the origami of receptor activation. *Current Biology*, 14(1):R22–R24, 2004.

[191] Schwann, T. Theorie der zellen. *Schleiden, MJ; Schwann, T.; Schultze, M.(1987): Klassische Schriften zur Zellenlehre, S*, pages 98–129, 1839.

[192] Serot, J., and Falcou, J. Functional meta-programming for parallel skeletons. In *International Conference on Computational Science*, pages 154–163. Springer, 2008.

[193] Serrano Rubio, Angélica Alejandra and Meneses Viveros, Amilcar and Morales Luna, Guillermo B and Paredes López, Mireya. Towards BIMAX: Binary Inclusion-MAXimal Parallel Implementation for Gene Expression Analysis Alejandra. *Computación y Sistemas*, 24(1), 2020.

[194] Shahapurkar, S. S., and Sundareshan, M. K. Comparison of self-organizing map with k-means hierarchical clustering for bioinformatics applications. In *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No. 04CH37541)*, volume 2, pages 1221–1226. IEEE, 2004.

[195] Shen, C., Shen, T., and Lin, J. Comparative assessment of alignment algorithms for ngs data: Features, considerations, implementations, and future. In *Algorithms for Next-Generation Sequencing Data*, pages 187–202. Springer, 2017.

[196] Sheng, J., Xiong, Q., Yang, C., and Herbordt, M. C. . Collective communication on fpga clusters with static scheduling. *ACM SIGARCH Computer Architecture News*, 44(4):2–7, 2017.

[197] Shi, S., Yang, R., Zhang, X., You, H., and Fan, D. An adaptive tuning sparse fast fourier transform. In *Pacific Rim Conference on Multimedia*, pages 991–999. Springer, 2017.

[198] Shiang, W. C., Aziz, I. A., Haron, N. S., Jaafar, J., Ismail, N. N., and Mehat, M. THE HIGH PERFORMANCE LINPACK (HPL) BENCHMARK EVALUATION ON UTP HIGH PERFORMANCE CLUSTER COMPUTING. *Jurnal Teknologi*, 78(9-3), 2016.

[199] Sieger, T., Hurley, C. B., Fiser, K., and Beleites, C. Interactive dendrograms: The r packages idendro and idendr0. 2017.

[200] Silvano, C., Agosta, G., Cherubin, S., Gadioli, D., Palermo, G., Bartolini, A., ... and Bispo, J. The ANTAREX approach to autotuning and adaptivity for energy efficient HPC systems. In *Proceedings of the ACM International Conference on Computing Frontiers*, pages 288–293, 2016.

[201] Sinwar, D., and Kaushik, R. Study of euclidean and manhattan distance metrics using simple k-means clustering. *Int. J. Res. Appl. Sci. Eng. Technol*, 2(5):270–274, 2014.

[202] Slater, E. P., Diehl, S. M., Langer, P., Samans, B., Ramaswamy, A., Zielke, A., and Bartsch, D. K. . Analysis by cDNA microarrays of gene expression patterns of human adrenocortical tumors. *European Journal of Endocrinology*, 154(4):587–598, 2006.

[203] Slonim, D. K., Tamayo, P., Mesirov, J. P., Golub, T. R., and Lander, E. S. . Class prediction and discovery using gene expression data. In *Proceedings of the fourth annual international conference on Computational molecular biology*, pages 263–272, 2000.

[204] Song, Y., Chen, W. Y., Bai, H., Lin, C. J., and Chang, E. Y. Parallel spectral clustering. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 374–389. Springer, 2008.

[205] Stoica, L. C., Cristescu, M. P., and Stancu, A. M. R. . Making the k-means algorithm using the hamming distance. *Group*, 1(1):1, 2017.

[206] Su, L., and Vaidya, N. H. Reaching approximate Byzantine consensus with multi-hop communication. *Information and Computation*, 255:352–368, 2017.

[207] Takahashi, D. Parallel fft algorithms for shared-memory parallel computers. In *Fast Fourier Transform Algorithms for Parallel Computers*, pages 69–76. Springer, 2019.

[208] Tapus, C., Chung, I. H., and Hollingsworth, J. K. Active harmony: Towards automated performance tuning. In *SC'02: Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*, pages 44–44. IEEE, 2002.

[209] Tavassoli, M. The cell theory: a foundation to the edifice of biology. *The American journal of pathology*, 98(1):44, 1980.

[210] Team, R. . RStudio: Integrated Development for R. RStudio, Inc., Boston, MA. 2015. *URL: https://www. rstudio. com/products/rstudio*, 2019.

[211] Trapnell, C., Pachter, L., and Salzberg, S. L. . Tophat: discovering splice junctions with rna-seq. *Bioinformatics*, 25(9):1105–1111, 2009.

[212] Trapnell, C., Roberts, A., Goff, L., Pertea, G., Kim, D., Kelley, D. R., and Pachter, L. . Differential gene and transcript expression analysis of RNA-seq experiments with TopHat and Cufflinks. *Nature protocols*, 7(3):562–578, 2012.

[213] Trovero, M. F., and Geisinger, A. Los arns no codificantes largos y su vinculación con las patologías testiculares.

[214] Troyanskaya, O., Cantor, M., Sherlock, G., Brown, P., Hastie, T., Tibshirani, R., ... and Altman, R. B. Missing value estimation methods for DNA microarrays. *Bioinformatics*, 17(6):520–525, 2001.

[215] Turi, R. H. . *Clustering-based colour image segmentation*. Monash University PhD thesis, 2001.

[216] Vadhiyar, S. S., Fagg, G. E., & Dongarra, J. Automatically tuned collective communications. In *Proceedings of the 2000 ACM/IEEE Conference on Supercomputing*, page 3. IEEE Computer Society, 2000.

[217] Velculescu, V. E., Zhang, L., Vogelstein, B., and Kinzler, K. W. Serial analysis of gene expression. *Science*, 270(5235):484–487, 1995.

[218] Virchow, R. *Post-mortem examinations: With especial reference to medico-legal practice*. P. Blakiston, Son and Company, 1885.

[219] Vogel, I., Blanshard, R. C., and Hoffmann, E. R. . SureTypeSC—a Random Forest and Gaussian mixture predictor of high confidence genotypes in single-cell data. *Bioinformatics*, 35(23):5055–5062, 2019.

[220] Vuduc, R., Demmel, J. W., and Yelick, K. A. OSKI: A library of automatically tuned sparse matrix kernels. In *Journal of Physics: Conference Series*, volume 16, page 521. IOP Publishing, 2005.

[221] Wang, J., Bø, T. H., Jonassen, I., Myklebost, O., and Hovig, E. Tumor classification and marker gene prediction by feature selection and fuzzy c-means clustering using microarray data. *BMC bioinformatics*, 4(1):1–12, 2003.

[222] Wang, Z., Zeng, P., Zhou, M., Li, D., and Wang, J. Cluster-based maximum consensus time synchronization for industrial wireless sensor networks. *Sensors*, 17(1):141, 2017.

[223] Whaley, R. C., Petitet, A., & Dongarra, J. J. Automated empirical optimizations of software and the ATLAS project. *Parallel Computing*, 27(1):3–35, 2001.

[224] Whaley, R. C., Petitet, A., and Dongarra, J. J. Automated empirical optimizations of software and the atlas project. *Parallel computing*, 27(1-2):3–35, 2001.

[225] Wilhelm, B. T., and Landry, J. R. . Rna-seq—quantitative measurement of expression through massively parallel rna-sequencing. *Methods*, 48(3):249–257, 2009.

[226] Wolpert, L. Evolution of the cell theory. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 349(1329):227–233, 1995.

[227] woo Kim, C., Ahn, S. H., and Yoon, T. Comparison of flavivirus using datamining-apriori, k-means, and decision tree algorithm. In *2017 19th International Conference on Advanced Communication Technology (ICACT)*, pages 454–457. IEEE, 2017.

[228] Wood, C., Georgakoudis, G., Beckingsale, D., Poliakoff, D., Gimenez, A., Huck, K., and Gamblin, T. Artemis: Automatic runtime tuning of parallel execution parameters using machine learning. In *International Conference on High Performance Computing*, pages 453–472. Springer, 2021.

[229] Wu, X., Marathe, A., Jana, S., Vysocky, O., John, J., Bartolini, A., ... and Bhalachandra, S. Toward an end-to-end auto-tuning framework in hpc powerstack. *arXiv preprint arXiv:2008.06571*, 2020.

[230] Xiao, Y., Zhang, N., Li, J., Lou, W., and Hou, Y. T. . Distributed consensus protocols and algorithms. *Blockchain for Distributed Systems Security*, 25, 2019.

[231] Xie, G., Chen, Y., Liu, Y., Wei, Y., Li, R., and Li, K. Resource consumption cost minimization of reliable parallel applications on heterogeneous embedded systems. *IEEE Transactions on Industrial Informatics*, 13(4):1629–1640, 2016.

[232] Xie, G., Jiang, J., Liu, Y., Li, R., and Li, K. Minimizing energy consumption of real-time parallel applications using downward and upward approaches on heterogeneous systems. *IEEE Transactions on Industrial Informatics*, 13(3):1068–1078, 2017.

[233] Yamato, Y., Noguchi, H., Kataoka, M., and Isoda, T. . Study to achieve environment adaptive software. *arXiv preprint arXiv:1903.06854*, 2019.

[234] You, H., Seymour, K., and Dongarra, J. An effective empirical search method for automatic software tuning. *UTK CS Technical Report*, 2005.

[235] F. Zapata. *Fundamentos y casos exitosos de la biotecnología moderna*. El Colegio Nacional, 2004.

[236] Zhang, B. Generalized k-harmonic means–boosting in unsupervised learning. *HP LABORATORIES TECHNICAL REPORT HPL*, (137), 2000.

[237] Zhao, W., Ma, H., and He, Q. Parallel k-means clustering based on mapreduce. In *IEEE International Conference on Cloud Computing*, pages 674–679. Springer, 2009.