UNIVERSIDAD DE GRANADA

TRABAJO FIN DE GRADO

INGENIERÍA INFORMÁTICA

Dockerización de Aplicación Paralela y Distribuida para Clasificación de EEGs: Análisis de Viabilidad y Rendimiento

DockEEG

Autor
Fernando Cuesta Bueno

Director
Juan José Escobar Pérez

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, septiembre de 2025

UNIVERSIDAD
DE GRANADA

DockEEG

Dockerización de Aplicación Paralela y Distribuida para Clasificación de EEGs: Análisis de Viabilidad y Rendimiento

Autor
Fernando Cuesta Bueno

Director
Juan José Escobar Pérez

DockEEG: Dockerización de Aplicación Paralela y Distribuida para Clasificación de EEGs: Análisis de Viabilidad y Rendimiento

Fernando Cuesta Bueno

Palabras clave: palabra_clave1, palabra_clave2, palabra_clave3, ...

Resumen

Poner aquí el resumen.

DockEEG: Dockerization of a Parallel and Distributed Application for EEG Classification: Feasibility and Performance Analysis

Fernando Cuesta Bueno

Keywords: Keyword1, Keyword2, Keyword3, ...

Abstract

Write here the abstract in English.

Yo, **Fernando Cuesta Bueno**, alumno de la titulación Graduado en Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 77150866B, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Fernando Cuesta Bueno

Granada a 5 de septiembre de 2025.

D. **Juan José Escobar Pérez**, Profesor del Área de XXXX del Departamento de Lengua de la Universidad de Granada.

Informa:

Que el presente trabajo, titulado ***Título del proyecto, Subtítulo del proyecto***, ha sido realizado bajo su supervisión por **Nombre Apellido1 Apellido2 (alumno)**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a X de mes de 202 .

Los directores:

Nombre Apellido1 Apellido2 (tutor1) **Nombre Apellido1 Apellido2 (tutor2)**

Agradecimientos

Poner aquí agradecimientos...

Índice general

Índice de figuras

Índice de tablas

Acrónimos

[Ejemplo de introducción y referencia de acrónimos Programmable Logic Controller (PLC). Para introducir nuevos acrónimos, ir al fichero `acro_list.tex`]

PLC Programmable Logic Controller

HPC High-Performance Computing

EEG Electroencephalogram

BCI Brain-Computer Interface

MOFS Multi-Objective Feature Selection

Machine Learning Aprendizaje Automático

MPI Message Passing Interface

OpenMP Open Multi-Processing

OpenCL Open Computing Language

MOGA Multi-Objective Genetic Algorithm

NSGA-II Non-dominated Sorting Genetic Algorithm II

WCSS Within-Cluster Sum of Squares

BCSS Between-Cluster Sum of Squares

Capítulo 1

Introducción

1.1. Motivación

1.2. Objetivos

Analizar la viabilidad y las limitaciones del uso de contenedores, concretamente Docker, para encapsular y ejecutar aplicaciones de alto rendimiento (HPC) en arquitecturas heterogéneas modernas —como big.LITTLE— y entornos multiplataforma, con el fin de facilitar su portabilidad, uso y adopción por parte de la comunidad científica.

1.2.1. Objetivos específicos

- **OB1.** Investigar el estado del arte en el ámbito de la tecnología de contenedores y su aplicación en entornos de computación de alto rendimiento.
- **OB2.** Diseñar e implementar un conjunto de experimentos para evaluar el rendimiento de aplicaciones HPC contenerizadas en diferentes arquitecturas y plataformas.
- **OB3.** Comparar el rendimiento de las aplicaciones contenerizadas con su ejecución nativa en diferentes entornos y arquitecturas, identificando las ventajas y desventajas de cada enfoque.
- **OB4.** Analizar los resultados obtenidos en los experimentos para identificar las limitaciones y desafíos asociados al uso de contenedores en entornos HPC, así como establecer futuras líneas de investigación.

Capítulo 2

Gestión del Proyecto

2.1. Tareas

Tareas de planificación del proyecto

- **Definición de objetivos y alcance del proyecto**
  Establecer claramente los objetivos principales y específicos del proyecto, así como el alcance y las limitaciones.
- **Planificación temporal**
  Desarrollar un cronograma detallado que incluya todas las fases del proyecto, desde la investigación inicial hasta la redacción final del informe.
- **Asignación de recursos**
  Identificar y asignar los recursos necesarios, tanto humanos como materiales, para llevar a cabo el proyecto de manera eficiente.
- **Gestión de riesgos**
  Identificar posibles riesgos que puedan afectar al desarrollo del proyecto y establecer planes de contingencia para mitigarlos.
- **Comunicación y seguimiento**
  Establecer canales de comunicación efectivos entre todos los miembros del equipo y realizar un seguimiento regular del progreso del proyecto para asegurar que se cumplen los plazos establecidos.

Tareas del OB1 (Estudio del estado del arte)

- **Revisión del estado del arte en HPC**
  Estudiar la evolución histórica y tendencias actuales en la computación de alto rendimiento.

- **Análisis del uso de contenedores en HPC**
  Revisar tecnologías de contenedores aplicadas a entornos científicos y de alto rendimiento. Comparar contenedores frente a máquinas virtuales en cuanto a eficiencia, overhead y portabilidad en HPC.

- **Estudio del uso de GPU en aplicaciones HPC**
  Revisar el papel de las GPUs en la aceleración de aplicaciones científicas y de ingeniería. Identificar librerías y frameworks para programación en GPU. Analizar casos de éxito en la integración de GPU en entornos HPC.

- **Investigación sobre el soporte de GPU en contenedores**
  Revisar soluciones actuales para ejecutar GPUs dentro de contenedores. Analizar el grado de compatibilidad con diferentes sistemas operativos y arquitecturas. Estudiar el impacto en rendimiento del uso de GPUs en entornos contenerizados en comparación con la ejecución nativa.

Tareas del OB2 (Diseño e implementación de la propuesta)

- **Selección de la aplicación o problema HPC a estudiar**
  Se seleccionará una aplicación representativa del ámbito HPC, justificando su elección en función de su relevancia científica, disponibilidad de código abierto y viabilidad técnica para su ejecución en diferentes plataformas y entornos contenerizados.

- **Preparar entornos y dependencias**
  Se identificarán y documentarán las librerías y herramientas necesarias, incluyendo MPI, OpenMP y CUDA. Se garantizará la homogeneidad de las configuraciones en todos los sistemas de prueba y se detallarán los requisitos específicos para cada plataforma (Linux, Windows, macOS).

- **Diseñar y construir imágenes de contenedor**
  Se desarrollarán Dockerfiles reproducibles que incluyan todas las dependencias necesarias, asegurando soporte para GPU mediante NVIDIA Container Toolkit. Las imágenes serán versionadas y almacenadas en un registro para facilitar su reutilización y trazabilidad.

- **Definir casos de prueba y parámetros de ejecución**
  Se establecerán experimentos mononodo variando el número de hebras, experimentos multinodo con diferentes cantidades de nodos y casos combinados que exploren el espacio de parámetros hebras $\times$ nodos.

- **Automatización y orquestación**
  Se implementarán scripts en para automatizar la ejecución de lotes de pruebas, así como la recogida y almacenamiento de logs y resultados.
- **Instrumentación y métricas**
  Se instrumentará la aplicación para medir tiempos totales de ejecución, uso de CPU y otros recursos. Se calcularán métricas como aceleración, eficiencia, throughput y overhead comparando la ejecución en contenedor frente a la nativa. Se generarán gráficos comparativos para el análisis de resultados.
- **Reproducibilidad y trazabilidad**
  Se mantendrá un repositorio con los Dockerfiles, scripts y documentación del proyecto. Se etiquetarán las versiones de las imágenes y dependencias para asegurar la reproducibilidad de los experimentos.

Tareas del OB3 (Evaluación de rendimiento)

- **Definición de criterios de comparación**
  Se establecerán las métricas principales para la comparación de rendimiento y se garantizará la paridad de configuraciones (versiones de compiladores, librerías, drivers).
- **Ejecución de pruebas comparativas**
  Se ejecutarán las mismas baterías de experimentos tanto en modo nativo como en contenedor. Se registrarán logs completos de cada ejecución.
- **Recopilación y organización de resultados**
  Se guardarán los tiempos de ejecución y métricas de uso de recursos, clasificando los datos según plataforma (Linux, Windows, macOS) y tipo de acelerador (CPU, GPU). Se establecerá un formato homogéneo para los ficheros de resultados (CSV o base de datos).
- **Visualización de resultados comparativos**
  Se generarán gráficos y tablas que destaquen los casos extremos (mejores y peores comportamientos), facilitando la interpretación de los resultados.

Tareas del OB4 (Análisis de resultados)

- **Revisión sistemática de los resultados experimentales**
  Se analizarán de manera estructurada los datos obtenidos en las pruebas, comparando el rendimiento entre ejecución nativa y contenerizada en las distintas plataformas (Linux, Windows, macOS) y ante el uso o

no de aceleradores (CPU, GPU). Se identificarán tendencias generales, anomalías y comportamientos consistentes.

- **Análisis cuantitativo del rendimiento**
  Se calcularán diferencias absolutas y relativas entre ejecución nativa y contenerizada, estimando overheads medios y por caso. Se evaluará la escalabilidad en cada escenario y se aplicará análisis estadístico para validar la significancia de las diferencias observadas.

- **Análisis cualitativo**
  Se identificarán ventajas no estrictamente de rendimiento, como portabilidad, reproducibilidad y facilidad de despliegue, así como limitaciones observadas relacionadas con drivers de GPU, gestión de red en contenedores y problemas de compatibilidad.

- **Detección de desafíos en la adopción de contenedores en HPC**
  Se evaluará la complejidad asociada a la configuración, despliegue y mantenimiento de entornos contenerizados en HPC, incluyendo la integración de aceleradores como GPU y la gestión de dependencias específicas.

- **Propuesta de líneas de investigación futura**
  A partir de los resultados y desafíos identificados, se propondrán posibles líneas de trabajo futuro.

Tareas de redacción del informe final

- **Estructuración del informe**
  Se definirá un esquema claro y lógico para el informe, asegurando que cada sección fluya de manera coherente hacia la siguiente.

- **Redacción de secciones técnicas**
  Se describirán detalladamente los métodos, experimentos y resultados obtenidos, utilizando un lenguaje técnico preciso y adecuado para la audiencia objetivo.

- **Incorporación de gráficos y tablas**
  Se incluirán visualizaciones que apoyen y clarifiquen los puntos clave del informe, asegurando que estén correctamente etiquetadas y referenciadas en el texto.

- **Revisión y edición**
  Se realizará una revisión exhaustiva del informe para corregir errores gramaticales, mejorar la claridad y coherencia del texto, y asegurar que todas las referencias bibliográficas estén correctamente citadas.

- **Formato y presentación**
  Se aplicarán las normas de formato establecidas por la institución o publicación a la que se destina el informe, asegurando una presentación profesional y uniforme.

2.2. Planificación temporal

En la tabla 2.1 se presenta una estimación del tiempo necesario para completar cada una de las tareas principales del proyecto, desglosado en horas dedicadas por el desarrollador y el tutor.

Tarea	**Desarrollador (h)**	**Tutor (h)**
Planificación	20	5
Estado del arte	40	10
Implementación	85	8
Evaluación	55	7
Análisis	30	5
Informe	30	15
Total	**260**	**50**

Tabla 2.1: Planificación temporal de tareas y horas estimadas

En la imagen 2.1 se muestra un diagrama de Gantt que ilustra la distribución temporal de las tareas a lo largo del proyecto.

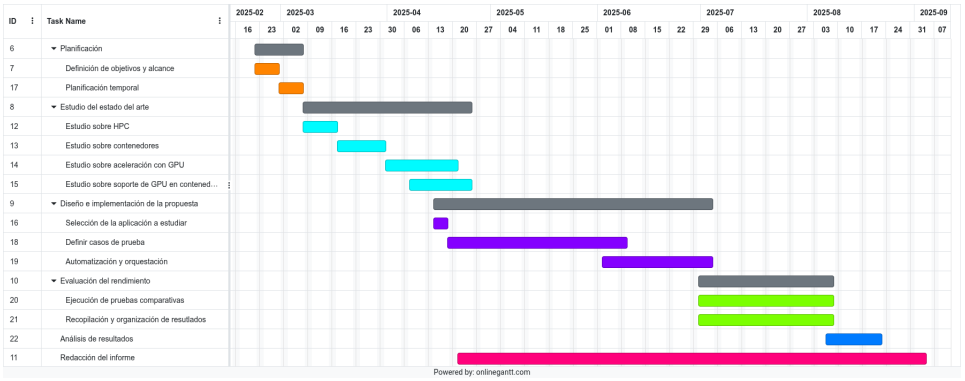

Figura 2.1: Diagrama de Gantt del proyecto

2.3. Estimación de costes

A continuación, se detallan los recursos necesarios para llevar a cabo el proyecto, incluyendo hardware, software y recursos humanos, junto con una estimación de los costes asociados.

Hardware

- Ordenador portátil LG Gram 14Z90Q-G.AA75B, este equipo se utilizará para el desarrollo general del trabajo: creación del código para las pruebas, gestión de las pruebas en el clustery creación de la memoria. Cuenta con un procesador Intel Core i7-1260P, 16 GB de RAM y 512 GB de almacenamiento SSD.

- Ordenador portátil Lenovo Legion 5, utilizado como plataforma principal para la ejecución de pruebas de rendimiento, especialmente aquellas que requieren aceleración por GPU. Este equipo está equipado con un procesador AMD Ryzen 7 4800H (8 núcleos y 16 hilos), 16 GB de memoria RAM DDR4, 512 GB de almacenamiento SSD NVMe y una tarjeta gráfica dedicada NVIDIA GeForce RTX 2060 con 6 GB de VRAM GDDR6. La presencia de la GPU NVIDIA permite la ejecución y evaluación de aplicaciones HPC que hacen uso de CUDA, así como la integración y validación del soporte de GPU en entornos contenerizados mediante NVIDIA Container Toolkit. Además, este equipo facilita la comparación de resultados entre ejecución nativa y contenerizada en escenarios reales de computación de alto rendimiento.

- Ordenador Apple Mac Mini M4, utilizado como plataforma de pruebas para la ejecución y validación de aplicaciones HPC en entornos macOS y arquitectura ARM. Este equipo incorpora un procesador Apple M4 con arquitectura ARM64, núcleos de alto rendimiento y eficiencia, GPU integrada de última generación y 16 GB de memoria unificada. El Mac Mini M4 permitirá evaluar la compatibilidad y el rendimiento de contenedores en sistemas Apple Silicon.

Software

- Sistema operativo Ubuntu 24.04 LTS. Será la distribución Linux principal con la que vamos a trabajar, tanto en forma nativa, como en los contenedores.

- Sistema operativo Microsoft Windows 11 Home. Será la distribución con la que se ejecutarán las pruebas de compatibilidad y rendimiento en entornos Windows.

- Sistema operativo macOS Sequoia 15.5. Será la distribución con la que se ejecutarán las pruebas de compatibilidad y rendimiento en entornos Apple.

Recursos humanos.

Dispositivo	**Descripción**	**Coste (€)**
LG Gram 14Z90Q-G.AA75B	Portátil principal de desarrollo	1 200
Lenovo Legion 5	Portátil de pruebas	1 000
Apple Mac Mini M4	Dispositivo de pruebas Apple	599
Total		**2 799**

Tabla 2.2: Costes estimados de hardware para el proyecto

En cuanto al software, los sistemas operativos Microsoft Windows 11 Home y macOS Sequoia 15.5 vienen incluidos en los dispositivos correspondientes, por lo que no se ha considerado un coste adicional. El sistema operativo Ubuntu 24.04 LTS es de código abierto y gratuito, por lo que tampoco se ha considerado un coste adicional.

Recursos humanos

En la tabla 2.3 se detalla el coste por hora, las horas estimadas y el coste total de los recursos humanos necesarios para llevar a cabo el proyecto.

Recurso	**Puesto**	**€/h**	**Horas**	**Total (€)**
Fernando Cuesta Bueno	Desarrollador	20	260	5 200
Juan José Escobar Pérez	Tutor/Supervisor	50	50	2 500
Total				**7 700**

Tabla 2.3: Costes estimados de recursos humanos para el proyecto

Coste total del proyecto

El coste total del proyecto se calcula sumando los costes de hardware, software y recursos humanos. En la tabla 2.4 se detalla el coste total estimado.

Concepto	**Coste (€)**
Hardware	2 799
Software	0
Recursos humanos	7 700
Total	**10 499**

Tabla 2.4: Coste total estimado del proyecto

Capítulo 3

Estado del arte

[En el estado del arte se necesita hacer un estudio tanto sobre la tecnología que soporta el proyecto como sobre el problema que se aborda en él. Se puede estructurar por secciones y se aconseja utilizar referencias a los documentos e información que se describe aquí.

Como norma general y más en proyectos con carácter investigador, se recomienda añadir un párrafo por cada documento/referencia que estudie del estado del arte, finalizando esta sección con un párrafo explicativo de la novedad/característica que propone, modifica o añade el proyecto sobre dicho estado del arte.]

3.1. Sección

3.1.1. Sub-seccion

Capítulo 4

Propuesta principal

4.1. Introducción de la propuesta

En esta sección se presenta la propuesta general del trabajo, explicando la metodología seguida para evaluar el uso de contenedores en entornos HPC. Se justifica la elección de los experimentos en función de los objetivos del TFG y del estado del arte, y se define el alcance de los mismos, indicando qué se medirá y evaluará.

4.2. Aplicación seleccionada: HPMoon

4.2.1. Origen y contexto

HPMoon es un software desarrollado en el marco de la tesis doctoral *"Energy-Efficient Parallel and Distributed Multi-Objective Feature Selection on Heterogeneous Architectures"*, defendida por Juan José Escobar Pérez el 20 de febrero de 2020 en la Universidad de Granada. El programa se concibe como una herramienta del proyecto *e-hpMOBE* y está disponible públicamente en un repositorio de GitHub. Su desarrollo fue apoyado por proyectos de investigación nacionales financiados por el Ministerio de Ciencia, Innovación y Universidades (MICIU) y fondos FEDER, así como por una beca de NVIDIA. Esta aplicación se utiliza como caso de estudio en este TFG por su significativa relevancia científica y tecnológica en el ámbito de la computación de alto rendimiento (HPC).

4.2.2. Objetivo de HPMoon

El objetivo principal de HPMoon es abordar la clasificación no supervisada de señales de Electroencefalograma (EEG) en tareas de Interfaz Cerebro-

Computadora (BCI). Este es un problema de alta dimensionalidad y computacionalmente costoso debido a las características de las señales EEG y al gran número de características que pueden contener. La aplicación combina la selección de características multiobjetivo (MOFS) con algoritmos paralelos y energéticamente eficientes, buscando minimizar el tiempo de ejecución y el consumo de energía, cruciales en problemas de Machine Learning y bioingeniería que requieren plataformas de alto rendimiento.

4.2.3. Arquitectura y funcionamiento

HPMoon implementa un procedimiento paralelo multinivel y energéticamente eficiente para la selección de características multiobjetivo (MOFS). La versión descrita como la "más eficiente desarrollada hasta la fecha" es ODGA. La arquitectura y funcionamiento se pueden desglosar en:

- **Enfoque Wrapper con Algoritmos Genéticos Multiobjetivo (MOGA)**: HPMoon utiliza un enfoque wrapper donde un MOGA, específicamente una adaptación del algoritmo NSGA-II, realiza la selección de características. Este algoritmo opera con poblaciones de individuos (cromosomas) que codifican diferentes selecciones de características.

- **Evaluación de Fitness con K-means**: La fitness de cada individuo se evalúa mediante el algoritmo K-means para clasificación no supervisada. El objetivo es minimizar dos funciones de coste:

  - f_1 (minimización de WCSS), que representa la suma de distancias dentro de los clústeres.
  - f_2 (maximización de BCSS), que se refiere a la suma de distancias entre los centroides.

  Estas funciones se normalizan en el intervalo (0,1).

- **Paralelismo Multinivel**: La aplicación explota hasta cuatro niveles de paralelismo en arquitecturas heterogéneas:

  1. Distribución entre nodos del clúster mediante MPI.
  2. Distribución entre dispositivos CPU/GPU por nodo, gestionado con OpenMP.
  3. Distribución entre Unidades de Cómputo (CUs) o hilos de CPU.
  4. Paralelismo de datos en GPU para K-means, aprovechando el paralelismo de datos para cálculo de distancias y actualización de centroides.

- **Tecnologías de implementación**: El código está desarrollado principalmente en C++, utilizando:
  - MPI (*Message Passing Interface*) para la comunicación entre nodos en sistemas distribuidos.
  - OpenMP para el paralelismo en CPU.
  - OpenCL para los kernels de K-means en GPU.
- **Optimización de carga de trabajo**: Incluye esquemas master-worker con balanceo dinámico de carga entre CPU y GPU, así como entre nodos del clúster. Las versiones posteriores, como ODGA, optimizan la comunicación y distribución para minimizar desequilibrio de carga.
- **Configuración y uso**: El programa se ejecuta desde la línea de comandos, permitiendo la configuración de parámetros (número de subpoblaciones, tamaño de la población, migraciones, generaciones, número de características y uso de CPU/GPU) mediante un archivo XML.

4.2.4. Justificación de la elección para este TFG

HPMoon se selecciona como caso de estudio por las siguientes razones, que lo convierten en un candidato ideal para analizar portabilidad, rendimiento y overhead de contenedores:

- **Relevancia en HPC y problemas reales**: HPMoon aborda un problema intensivo en cómputo (clasificación EEG de alta dimensionalidad) frecuente en bioinformática e ingeniería biomédica. Su complejidad permite generar métricas relevantes en HPC.
- **Paralelización multinivel y arquitecturas heterogéneas**: Diseñado para explotar múltiples niveles de paralelismo en CPUs multinúcleo, GPUs y sistemas distribuidos (clústeres), permitiendo análisis exhaustivo en diversas configuraciones.
- **Énfasis en eficiencia energética**: Minimiza consumo de energía y tiempo de ejecución, aspecto crucial en HPC moderna.
- **Estrategias de balanceo de carga**: Distribución dinámica de carga entre CPU y GPU, permitiendo manejar diferencias de capacidad y consumo energético de dispositivos heterogéneos.
- **Disponibilidad y madurez**: Software robusto derivado de tesis doctoral y publicaciones científicas, con versiones evolutivas (SGA, PGA, OPGA, MDGA, MPGA, DGA, DGA-II, ODGA, GAAM) y documentación completa.

- **Modelos de energía-tiempo**: Incluye desarrollo y evaluación de modelos para predecir comportamiento de algoritmos en sistemas monocomputador y distribuidos, proporcionando base sólida para comparar resultados en contenedores.
- **Complejidad y desafíos de optimización**: Presenta retos como balanceo de carga en entornos heterogéneos, irregularidades en accesos a memoria y gestión de comunicación entre nodos y dispositivos, ideales para evaluar impacto de contenedores en optimización del código.

4.2.5. Configuración y parámetros de compilación y ejecución

La aplicación HPMoon requiere una fase previa de compilación y, posteriormente, una configuración en tiempo de ejecución a través de un fichero XML (con soporte parcial mediante parámetros de línea de comandos).

Compilación

La compilación se realiza mediante un *Makefile*, los principales parámetros de compilación son:

- **N_FEATURES (NF)**: número de características de la base de datos (columnas). Debe estar entre 4 y el máximo disponible.
- **COMP (COMPILER)**: compilador MPI a emplear (por defecto, `mpic++`).

Estos parámetros se procesan en tiempo de compilación para evitar el uso de memoria dinámica y maximizar el rendimiento. El ejecutable resultante, denominado `hpmoon`, se ubica en el directorio `bin`.

Configuración en tiempo de ejecución

La configuración en tiempo de ejecución se realiza principalmente mediante un fichero XML, que permite ajustar tanto los parámetros de los algoritmos evolutivos como la gestión de recursos computacionales. Los parámetros más relevantes son:

- **NSubpopulations**: número total de subpoblaciones (modelo de islas).
- **SubpopulationSize**: tamaño de cada subpoblación (número de individuos).

- **NGlobalMigrations**: número de migraciones entre subpoblaciones en diferentes nodos.
- **NGenerations**: número de generaciones a simular.
- **MaxFeatures**: número máximo de características permitidas.
- **DataFileName**: fichero de salida con la aptitud de los individuos del primer frente de Pareto.
- **PlotFileName**: fichero con el código de gnuplot para visualización.
- **ImageFileName**: fichero de salida con la gráfica generada.
- **TournamentSize**: número de individuos en el torneo de selección.
- **NInstances**: número de instancias (filas) de la base de datos a utilizar.
- **FileName**: nombre del fichero de la base de datos de entrada.
- **Normalize**: indica si la base de datos debe ser normalizada.
- **NDevices**: número de dispositivos OpenCL empleados en el nodo.
- **Names**: lista de nombres de dispositivos OpenCL, separados por comas.
- **ComputeUnits**: unidades de cómputo por dispositivo OpenCL, en el mismo orden que `Names`.
- **WiLocal**: número de *work-items* por unidad de cómputo, alineado con los dispositivos.
- **CpuThreads**: número de hilos de CPU para la evaluación de aptitud. Si es 1 y `NDevices=0`, la ejecución es secuencial.
- **KernelsFileName**: fichero con los kernels de OpenCL.

Muchos de estos parámetros pueden especificarse también mediante opciones en la línea de comandos al ejecutar `hpmoon`, aunque no todos están disponibles de esta forma. Esta circunstancia se refleja en la columna `CMD OPTION` de la Tabla 4.1, donde un guion (`-`) indica ausencia de soporte por línea de comandos.

PARÁMETRO	**RANGO**	**CMD OPTION**
N_FEATURES	$4 \leq$ NF $\leq$ Número de características de la base de datos	-
NSubpopulations	$1 \leq$ NP	-ns
SubpopulationSize	$4 \leq$ PS	-ss
NGlobalMigrations	$1 \leq$ NM	-ngm
NGenerations	$0 \leq$ NG	-g
MaxFeatures	$1 \leq$ MaxF	-maxf
DataFileName	Nombre de fichero válido	-plotdata
PlotFileName	Nombre de fichero válido	-plotsrc
ImageFileName	Nombre de fichero válido	-plotimg
TournamentSize	$2 \leq$ TS	-ts
NInstances	$4 \leq$ NI $\leq$ Número de instancias de la base de datos	-trni
FileName	Base de datos de entrenamiento existente	-trdb
Normalize	1 ó 0	-trnorm
NDevices	$0 \leq$ ND	-
Names	Nombre de dispositivo existente	-
ComputeUnits	$1 \leq$ CU	-
WiLocal	$1 \leq$ WL $\leq$ Máx. work-items locales del dispositivo	-
CpuThreads	$0 \leq$ CT	-
KernelsFileName	Fichero de kernels existente	-ke
Display usage	-	-h
List OpenCL devices	-	-l

Tabla 4.1: Rango de valores de los parámetros de entrada y su uso desde la línea de argumentos (si está disponible).

4.2.6. Selección de parámetros de estudio

La decisión de dejar la mayoría de los parámetros con sus valores por defecto y estudiar únicamente la relación entre el número de subpoblaciones y el número de hebras se basa en la arquitectura de paralelismo multinivel inherente al diseño del programa y su impacto directo en la distribución de la carga de trabajo y el rendimiento. Algunas de las recomendaciones mencionadas provienen de la guía de usuario de la aplicación.

A continuación, se explica en detalle:

1. **Diseño del programa con paralelismo multinivel:**
   - El programa es un algoritmo evolutivo basado en subpoblaciones con paralelismo multinivel.
   - Utiliza MPI (Message Passing Interface) para distribuir las subpoblaciones entre los nodos de un clúster.
   - Dentro de cada nodo, emplea OpenMP para distribuir dinámicamente las subpoblaciones o individuos entre los dispositivos disponibles (CPU y GPU).
   - La evaluación de la aptitud de los individuos se realiza utilizando OpenMP en la CPU y OpenCL en la GPU, ofreciendo hasta tres niveles de paralelismo en la CPU y hasta cuatro en la GPU.
2. **Importancia del número de subpoblaciones (NSubpopulations):**
   - NSubpopulations representa el número total de subpoblaciones, un parámetro fundamental para el modelo basado en islas del algoritmo.
   - Se sugiere que aumentar el número de subpoblaciones puede generar resultados de buena calidad, incluso más que aumentar el número de individuos [1].
   - Este parámetro es crucial porque define cómo se divide el trabajo a nivel más alto (distribución MPI entre nodos) y cómo estas unidades de trabajo pueden ser gestionadas posteriormente por OpenMP dentro de cada nodo [1].
3. **Importancia del número de hebras (CPU y GPU):**
   - El programa hace un uso intensivo de hilos (*threads*) para la evaluación de la aptitud.
   - El parámetro `CpuThreads` especifica el número de hilos de CPU a utilizar en la evaluación de aptitud. La recomendación es que este valor sea igual al número de núcleos lógicos de la CPU para un buen rendimiento.

- Para la GPU, `NDevices` (número de dispositivos OpenCL), `ComputeUnits` (unidades de cómputo, λ) y `WiLocal` (número de elementos de trabajo o hilos por unidad de cómputo) son esenciales. Se aconseja que `WiLocal` sea un múltiplo de 32 o 64 para mejorar el rendimiento de OpenCL, y que la combinación óptima de `WiLocal` y `ComputeUnits` depende de las características del problema y del dispositivo.
- Estos parámetros controlan directamente la paralelización de la función de evaluación en la GPU con OpenCL y la distribución dinámica de individuos entre los dispositivos mediante OpenMP [1].

4. **Importancia del número de nodos:**
   - El programa está diseñado para ejecutarse en sistemas distribuidos, como clústeres, donde múltiples nodos están interconectados [1, 2].
   - La versión DGA (Distributed Genetic Algorithm) añade un cuarto nivel de paralelismo al distribuir las subpoblaciones entre los nodos del clúster utilizando MPI. El proceso maestro (MPI 0) se encarga de esta distribución dinámica [1, 2].
   - La escalabilidad y el rendimiento del algoritmo se ven directamente afectados por el número de nodos utilizados. Los experimentos demuestran que usar más nodos puede llevar a reducciones significativas en el tiempo de ejecución y el consumo de energía, logrando picos de speedup de hasta 83 veces y reduciendo el consumo energético a un 4.9 % en el mejor de los casos [1, 2].
   - La heterogeneidad de los nodos y su configuración de CPU/GPU también influye en la eficiencia de la distribución de carga y el consumo de energía. Por lo tanto, el número de nodos es un parámetro fundamental para el estudio del paralelismo a nivel de sistema distribuido [1, 2].

5. **Razones para mantener otros parámetros por defecto:**
   - **Prioridad del estudio de paralelismo:** La configuración de `NSubpopulations` y de los hilos (`CpuThreads`, `WiLocal`, `ComputeUnits`) son los que más influyen directamente en la estrategia de paralelismo y la asignación de recursos computacionales, que son el objetivo principal de este estudio inicial [1].
   - **Parámetros de compilación vs. ejecución:** Parámetros como `N_FEATURES` se establecen en tiempo de compilación (`make`), lo que los hace menos flexibles para pruebas de rendimiento en tiempo de ejecución. Los parámetros de subpoblaciones y hebras, por el contrario, son de tiempo de ejecución (fichero XML).

- **Impacto de la función de evaluación:** La función de evaluación (algoritmo K-means) consume más del 99 % del tiempo de ejecución para tamaños de población moderados [1]. Por lo tanto, el enfoque en cómo se paraleliza esta parte (a través de subpoblaciones y hebras) es crítico para el rendimiento.
- **Otros parámetros secundarios:** Otros parámetros, como `NGlobalMigrations`, `NGenerations`, `MaxFeatures`, `TournamentSize`, `NInstances`, `FileName`, `Normalize`, etc., están más relacionados con la configuración específica del algoritmo evolutivo, la gestión de datos o la calidad de la solución final, pero no afectan tan directamente al comportamiento fundamental del paralelismo y la escalabilidad del sistema [1].
- **Consideraciones de recursos:** El uso de bases de datos muy grandes (`NInstances`, `N_FEATURES`) puede provocar que el programa aborte debido a limitaciones de memoria local en la GPU. Es más prudente estudiar estos factores una vez que se comprenda bien el rendimiento del paralelismo fundamental.

En resumen, al concentrarse en el número de subpoblaciones, el número de hebras y el número de nodos, se aborda directamente la capacidad del programa para aprovechar arquitecturas paralelas y la distribución de la carga de trabajo en todos los niveles, sentando las bases para análisis más detallados posteriormente [1].

4.2.7. Diseño de los experimentos

Los experimentos se estructuran en varias fases con el fin de determinar los parámetros óptimos y evaluar la escalabilidad y portabilidad de HPMoon en distintos entornos y plataformas.

Determinación del número óptimo de subpoblaciones

Se realizarán ejecuciones exploratorias con distintas configuraciones de subpoblaciones y hebras, para definir el parámetro de `NSubpopulations` a usar en los tests posteriores:

- Se lanzarán ejecuciones con 1, 2, 4, 8 y 16 subpoblaciones.
- Para cada configuración de subpoblaciones, se evaluará con 1, 2, 4, 8 y 16 hebras por subpoblación.

Estudio del comportamiento al requerir más hebras de las disponibles

En el dispositivo Lenovo Legion 5, que dispone de un máximo de 16 hebras, se evaluará cómo se comporta la aplicación al requerir un número de hebras mayor al disponible:

- En las ejecuciones multinodo, cada nodo tendrá un número de hebras tal que el número total de hebras requerido sea igual a: *número de nodos* $\times$ *hebras por nodo*.
- Se realizarán dos variantes de test:
  1. Variante 1: si el número total de hebras requerido supera el máximo del dispositivo (16), el test no se ejecuta.
  2. Variante 2: se ejecutan los tests aunque el número total de hebras requerido sea mayor a 16.
- El objetivo es determinar si es necesario limitar el número de hebras solicitadas en los tests posteriores.

Experimentos de escalabilidad y barrido de hebras

Una vez definido el número óptimo de subpoblaciones y el rango válido de hebras, se realizarán los siguientes experimentos:

- **Escalabilidad mononodo:** con un único nodo fijo, se escala el número de hebras.
- **Escalabilidad multinodo:** con una única hebra por nodo, se escala el número de nodos.
- **Barrido de hebras:** se ejecutan todas las combinaciones posibles de número de nodos y hebras por nodo.

Plataformas y entornos de ejecución

Los experimentos se realizarán en distintas plataformas y entornos, tanto con uso único de CPU como la combinación con uso de GPU (salvo en el caso de Mac, donde el uso de GPU en contenedores es objeto de estudio):

- Ubuntu 24.04: ejecución nativa, y en contenedores Docker y Podman.
- Windows 11 Home: ejecución en contenedores Docker y Podman.

- MacOS Sequoia 15.5: ejecución en contenedores Docker y Podman.

Esta estructura permitirá evaluar la escalabilidad, portabilidad y rendimiento de HPMoon en entornos contenerizados y nativos, así como en arquitecturas heterogéneas con CPU y GPU.

4.3. Herramientas y scripts utilizados

Para llevar a cabo los experimentos propuestos, se han desarrollado diversos scripts que automatizan las pruebas en diferentes plataformas, entornos y configuraciones. Estos scripts permiten evaluar la escalabilidad, portabilidad y rendimiento de HPMoon en arquitecturas heterogéneas y entornos contenerizados. A continuación, se describen los principales scripts y su propósito:

4.3.1. Compilación del programa

- **Carpeta:** `utils`
- **Script:**
  - `build_hpmoon.sh`
- **Descripción:** Compila el programa HPMoon en el directorio de trabajo especificado utilizando `make` con parámetros específicos para maximizar el rendimiento, como el número de características (`N_FEATURES=3600`). Este paso asegura que el ejecutable esté disponible antes de iniciar los experimentos.

4.3.2. Experimentos específicos

- **Carpeta:** `experiments`
- **Script:**
  - `run_ubuntu_native_limit.sh`
  - `run_ubuntu_native_no-limit.sh`
  - `run_ubuntu_native_subpops_threads.sh`
- **Descripción:** Diseñados para pruebas concretas, como el impacto de exceder el número máximo de hebras disponibles o la relación entre subpoblaciones y hebras, permitiendo un análisis más detallado de parámetros clave.

4.3.3. Experimentos de escalabilidad mononodo

- **Carpeta:** `single-node`
- **Script:**
  - `run_ubuntu_native.sh`
- **Descripción:** Ejecuta pruebas de escalabilidad en un único nodo, variando el número de hebras utilizadas, evaluando el rendimiento nativo en sistemas Ubuntu sin contenedores.
- **Variantes:**
  - Versiones para contenedores:
    - `run_ubuntu_container.sh`
    - `run_mac_container.sh`
    - `run_windows_container.ps1`
  - Versiones para GPU:
    - `run_ubuntu_gpu_container.sh`
    - `run_mac_gpu_container.sh`
    - `run_windows_gpu_container.ps1`

4.3.4. Experimentos de escalabilidad multinodo

- **Carpeta:** `multi-node`
- **Script:**
  - `run_ubuntu_native.sh`
- **Descripción:** Distribuye las subpoblaciones entre varios nodos utilizando una única hebra por nodo, evaluando el rendimiento nativo en sistemas distribuidos.
- **Variantes:**
  - Versiones para contenedores:
    - `run_ubuntu_container.sh`
    - `run_mac_container.sh`
    - `run_windows_container.ps1`
  - Versiones para GPU:
    - `run_ubuntu_gpu_container.sh`
    - `run_mac_gpu_container.sh`
    - `run_windows_gpu_container.ps1`

4.3.5. Barrido de hebras

- **Carpeta:** `thread-sweep`
- **Script:**
  - `run_ubuntu_native.sh`
- **Descripción:** Ejecuta todas las combinaciones posibles de número de nodos y hebras por nodo, permitiendo identificar configuraciones óptimas y evaluar el comportamiento del programa en diferentes escenarios.
- **Variantes:**
  - Versiones para contenedores:
    - `run_ubuntu_container.sh`
    - `run_mac_container.sh`
    - `run_windows_container.ps1`
  - Versiones para GPU:
    - `run_ubuntu_gpu_native.sh`
    - `run_ubuntu_gpu_container.sh`
    - `run_mac_gpu_container.sh`
    - `run_windows_gpu_container.ps1`

4.3.6. Ejecución automatizada

- **Carpeta:** `utils`
- **Script:**
  - `run_ubuntu_all.sh`
  - `run_mac_all.sh`
  - `run_windows_all.ps1`
  - `run_cluster_all.sh`
- **Descripción:** Estos scripts agrupan y ejecutan automáticamente todos los experimentos correspondientes a cada plataforma o entorno, facilitando la ejecución secuencial de múltiples pruebas según la metodología definida.

4.4. Repositorio del proyecto

Todo el trabajo desarrollado en este TFG, incluyendo scripts, configuraciones, documentación y resultados de los experimentos, está recogido y disponible públicamente en un repositorio de GitHub[1].

[1] `https://github.com/FerniCuesta/DockEEG`

Capítulo 5

Experimentación

5.1. Resultados de los Experimentos

5.1.1. Pruebas Mononodo

Ejecución Nativa

Solo CPU

- Gráficas de tiempo de ejecución vs número de hebras.
- Gráficas de uso de CPU vs número de hebras.
- Tablas con datos clave (tiempo, uso de CPU, eficiencia, etc.).

CPU + GPU

- Gráficas de tiempo de ejecución vs número de hebras.
- Gráficas de uso de CPU/GPU vs número de hebras.
- Comparativa entre solo CPU y CPU + GPU.

Contenedores en Nativo

Solo CPU

- Comparativa entre ejecución nativa, Docker y Podman.
- Gráficas y tablas de tiempo de ejecución y uso de CPU.

CPU + GPU

- Comparativa entre ejecución nativa, Docker y Podman con GPU habilitada.
- Gráficas y tablas de tiempo de ejecución y uso de CPU/GPU.

Contenedores en Windows

Solo CPU

- Gráficas y tablas de tiempo de ejecución y uso de CPU.
- Comparativa con ejecución nativa.

CPU + GPU

- Gráficas y tablas de tiempo de ejecución y uso de CPU/GPU.
- Comparativa con ejecución nativa.

Contenedores en Mac

Solo CPU

- Gráficas y tablas de tiempo de ejecución y uso de CPU.
- Comparativa con ejecución nativa.

CPU + GPU

- Gráficas y tablas de tiempo de ejecución y uso de CPU/GPU.
- Comparativa con ejecución nativa.

5.1.2. Pruebas Multinodo

Ejecución Nativa

Solo CPU

- Gráficas de tiempo de ejecución vs número de nodos.
- Gráficas de uso de CPU vs número de nodos.
- Tablas con datos clave (tiempo, uso de CPU, eficiencia, etc.).

CPU + GPU

- Gráficas de tiempo de ejecución vs número de nodos.
- Gráficas de uso de CPU/GPU vs número de nodos.
- Comparativa entre solo CPU y CPU + GPU.

Contenedores en Nativo

Solo CPU

- Comparativa entre ejecución nativa, Docker y Podman.
- Gráficas y tablas de tiempo de ejecución y uso de CPU.

CPU + GPU

- Comparativa entre ejecución nativa, Docker y Podman con GPU habilitada.
- Gráficas y tablas de tiempo de ejecución y uso de CPU/GPU.

Contenedores en Windows

Solo CPU

- Gráficas y tablas de tiempo de ejecución y uso de CPU.
- Comparativa con ejecución nativa.

CPU + GPU

- Gráficas y tablas de tiempo de ejecución y uso de CPU/GPU.
- Comparativa con ejecución nativa.

Contenedores en Mac

Solo CPU

- Gráficas y tablas de tiempo de ejecución y uso de CPU.
- Comparativa con ejecución nativa.

CPU + GPU

- Gráficas y tablas de tiempo de ejecución y uso de CPU/GPU.
- Comparativa con ejecución nativa.

5.1.3. Barrido de Hebras

Ejecución Nativa

Solo CPU

- Gráficas de tiempo de ejecución vs número de hebras.
- Gráficas de uso de CPU vs número de hebras.
- Tablas con datos clave (tiempo, uso de CPU, eficiencia, etc.).

CPU + GPU

- Gráficas de tiempo de ejecución vs número de hebras.
- Gráficas de uso de CPU/GPU vs número de hebras.
- Comparativa entre solo CPU y CPU + GPU.

Contenedores en Nativo

Solo CPU

- Comparativa entre ejecución nativa, Docker y Podman.
- Gráficas y tablas de tiempo de ejecución y uso de CPU.

CPU + GPU

- Comparativa entre ejecución nativa, Docker y Podman con GPU habilitada.
- Gráficas y tablas de tiempo de ejecución y uso de CPU/GPU.

Contenedores en Windows

Solo CPU

- Gráficas y tablas de tiempo de ejecución y uso de CPU.
- Comparativa con ejecución nativa.

CPU + GPU

- Gráficas y tablas de tiempo de ejecución y uso de CPU/GPU.
- Comparativa con ejecución nativa.

Contenedores en Mac

Solo CPU

- Gráficas y tablas de tiempo de ejecución y uso de CPU.
- Comparativa con ejecución nativa.

CPU + GPU

- Gráficas y tablas de tiempo de ejecución y uso de CPU/GPU.
- Comparativa con ejecución nativa.

5.1.4. Comparativa General

- Mononodo vs Multinodo: Comparativa de tiempo de ejecución y uso de CPU. Impacto de la distribución multinodo en el rendimiento.
- Nativo vs Contenedores: Comparativa de tiempo de ejecución y uso de CPU/GPU entre entornos. Análisis de la penalización (si existe) al usar contenedores.
- Solo CPU vs CPU + GPU: Comparativa de rendimiento entre configuraciones. Análisis del impacto de la GPU en el rendimiento.

Capítulo 6

Conclusiones y trabajo futuro

[En este capítulo se presentan las conclusiones obtenidas al llevar a cabo el presente trabajo]

6.1. Contribuciones

[En esta sección se presentan las principales contribuciones del trabajo realizado.]

- Contribución 1 ...
- Contribución 2 ...

6.2. Retos y trabajo futuro

[Exponer aquí los retos y trabajos futuros.]

Capítulo 7

Bibliografía

Bibliografía

[1] J. J. Escobar, "Energy-efficient parallel and distributed multi-objective feature selection on heterogeneous architectures," Ph.D. dissertation, 2020.

[2] J. J. Escobar, J. Ortega, A. F. Díaz, J. González, and M. Damas, "Time-energy analysis of multilevel parallelism in heterogeneous clusters: the case of eeg classification in bci tasks," *The Journal of Supercomputing*, vol. 75, no. 7, pp. 3397–3425, 2019. [Online]. Available: https://doi.org/10.1007/s11227-019-02908-4