Diseño e Implementación del Algoritmo GPU-DBSCAN para la Identificación y Clasificación de Agregados en Imágenes

Rodrigo Lomba Moreno

Tutores: Nazario Félix González
Antonio Díaz Pozuelo

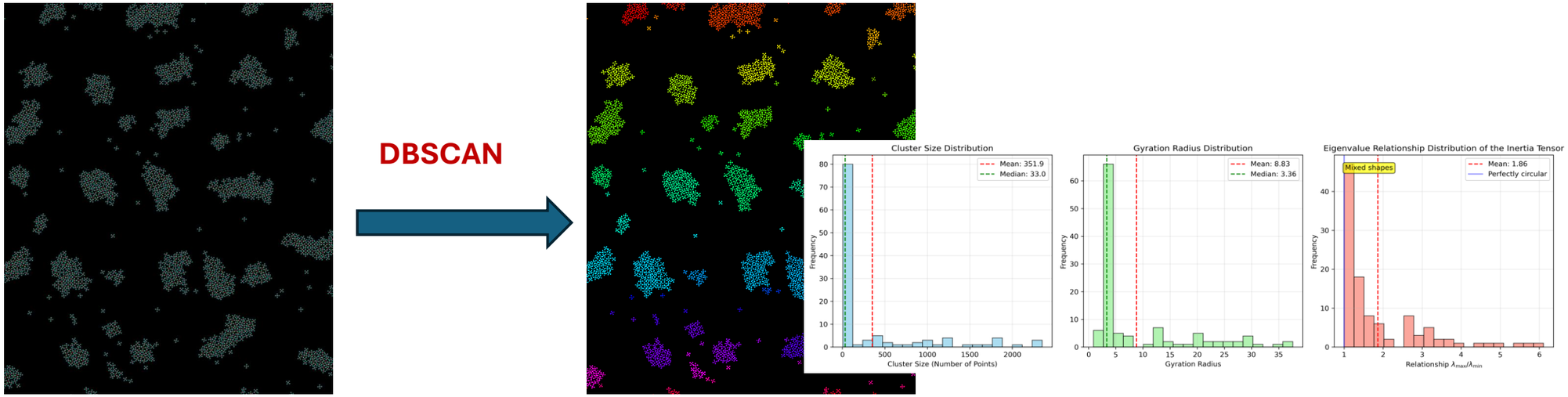
Escuela Técnica Superior de Ingenieros Informáticos

Universidad Politécnica de Madrid

OBJETIVOS

Objetivo general

Implementación de un **algoritmo paralelo** en **GPU** basado en **DBSCAN** para la detección y posterior análisis estadístico de **agregados en conjuntos de datos 2D**, obtenidos directamente o previa digitalización de imágenes



Objetivos específicos

Versión secuencial

Implementar una versión en Python, que sirva para la comprobación y comparación de resultados.

Implementación GPU

Diseñar e implementar una versión paralela del algoritmo DBSCAN optimizada para GPU usando arquitectura CUDA.

Extracción de Características

Desarrollar un módulo para calcular propiedades morfológicas de agregados identificados.

Optimización de Versiones

Mejorar ambas implementaciones analizando el rendimiento y aplicando optimizaciones con el objetivo de reducir el tiempo de ejecución

Evaluación de Rendimiento

Comparar el rendimiento entre implementaciones CPU y GPU.

MARCO TEÓRICO

Algoritmo DBSCAN

Fundamentos

DBSCAN identifica agregados basándose en densidad de puntos, sin necesidad de especificar el número de grupos previamente.

Parámetros clave:

- ε (epsilon): distancia de conectividad
- MinPts: puntos mínimos para formar agregado

Pasos del algoritmo:

1. Inicialización
2. Selección del punto inicial
3. Búsqueda de vecinos
4. Propagación del agregado
5. Iteración a 2 hasta fin
6. Reevaluación del ruido

Tipos de Puntos

- **Núcleo:** ≥ MinPts vecinos dentro de ε
- **Frontera:** dentro de ε de un núcleo
- **Ruido:** no pertenece a ningún agregado

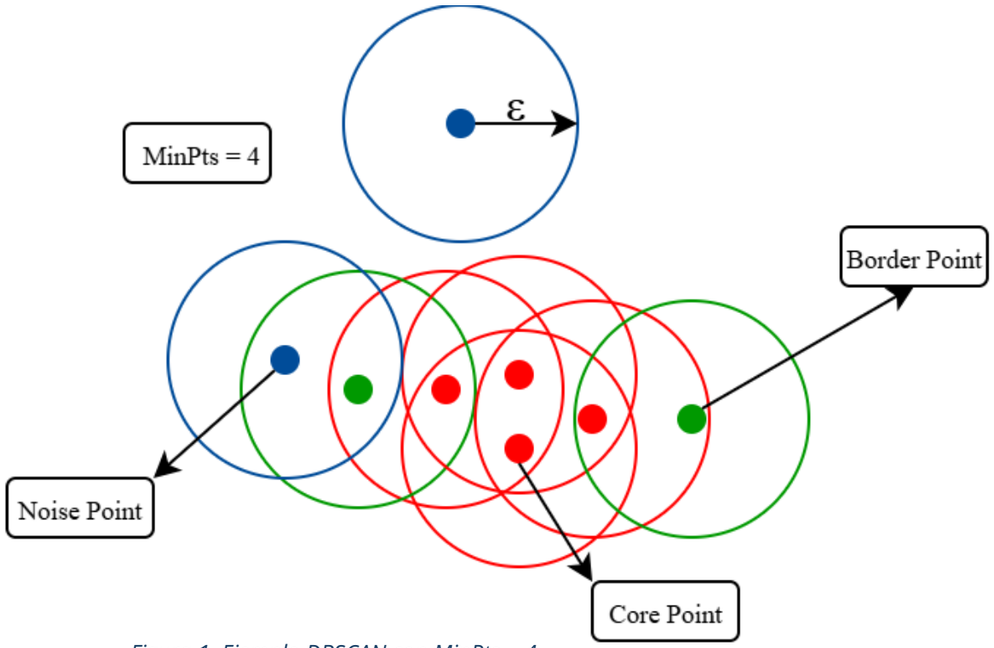


Figura 1. Ejemplo DBSCAN con MinPts = 4.

Extracción de Características Morfológicas

01

Centro de Masas (CDM)

Posición promedio de todos los puntos del agregado.

$$\overrightarrow{R_{\text{CDM}}} = \frac{1}{N_{\text{clus}}} \sum_{i=1}^{N\text{clus}} \vec{r_i}$$

02

Radio de Giro

Medida del tamaño efectivo del agregado respecto al CDM.

$$R_g^2 = \frac{1}{N_{clus}} \sum_{i=1}^{N clus} \left| \vec{r_i} - \overrightarrow{R_{CDM}} \right|^2$$

03

Tensor de Inercia

Caracteriza la forma de los agregados.

$$\mathcal{I} = \begin{pmatrix} \sum y_i'^2 & -\sum x_i' y_i' \\ -\sum x_i' y_i' & \sum x_i'^2 \end{pmatrix}$$

Donde: $\vec{r}'_i = \vec{r}_i - \vec{R}_{CDM}$

04

Valores propios del Tensor de Inercia

Si $\lambda_1 \approx \lambda_2$, la forma es aproximadamente circular.
Si $\lambda_1 \gg \lambda_2$ o $\lambda_1 \ll \lambda_2$, el agregado tendrá una forma alargada

$$\lambda_{1,2} = \frac{\mathcal{I}_{xx} + \mathcal{I}_{yy} \pm \sqrt{\left(\mathcal{I}_{xx} - \mathcal{I}_{yy}\right)^2 + 4\mathcal{I}_{xy}^2}}{2}$$

Ventajas de la Computación GPU en DBSCAN

Aspectos Claves

1. **Aceleración masiva**: Reducción drástica del tiempo de ejecución
2. **Complejidad optimizada**
3. **Paralelismo eficiente:** Un hilo por punto: aprovechando el modelo SIMT (Single Instruction Multiple Thread) y jerarquía CUDA.
4. **Escalabilidad:** Rendimiento estable incluso con conjuntos de datos muy grandes.

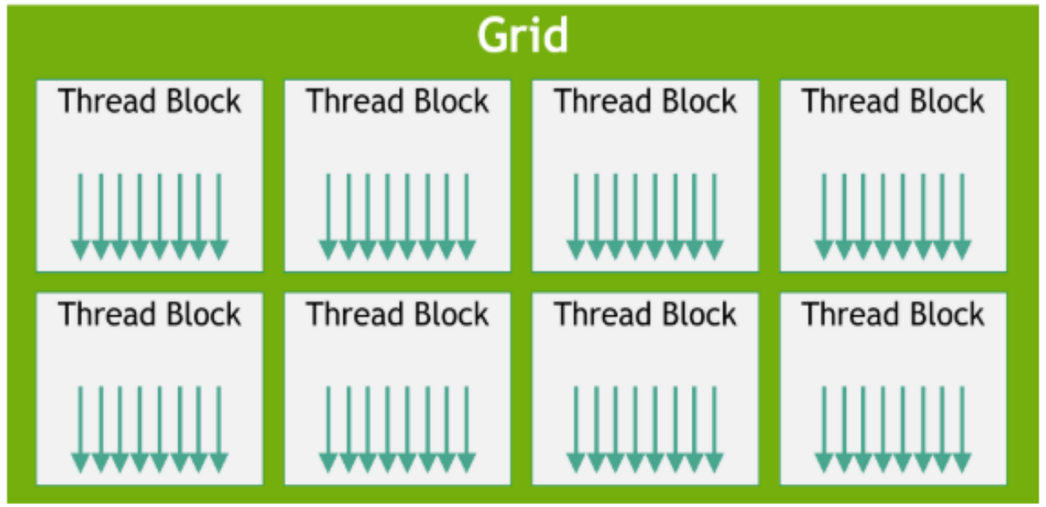


Figura 2. Estructura de una malla (Grid) en CUDA, compuesta por múltiples bloques de hilos (Thread Blocks) [1].

[1] NVIDIA, *CUDA C++ Programming Guide*, versión 13.0, NVIDIA Corporation, 2025. Disponible: https://docs.nvidia.com/cuda/cuda-c-programming-guide/

DISEÑO

Diseño del Código

Arquitectura del código

Procesamiento de Entrada

Conversión de imágenes a coordenadas 2D
Lectura de datos de archivos NetCDF

Selección de Parámetros

Cálculo de ϵ (distancia de conectividad) automático

DBSCAN

Diseño del grafo y propagación de agregados

Extracción de Propiedades

Cálculo paralelo de características morfológicas

Conversión de Imágenes

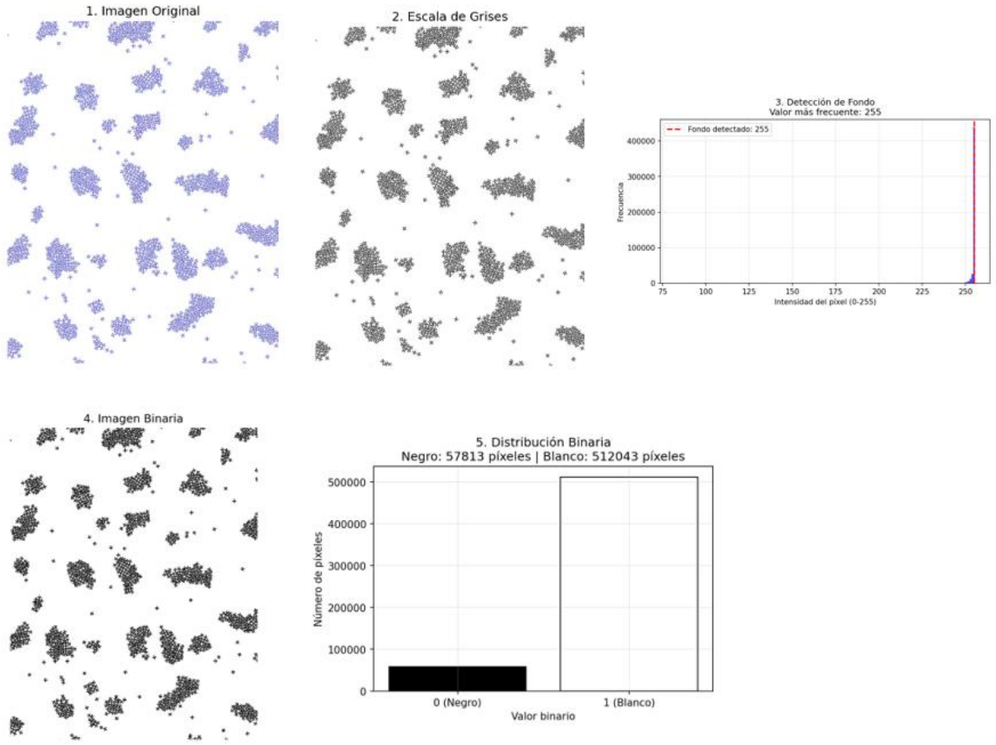


Figura 3. Flujo de Procesamiento de Imagen para Obtención de Puntos. .

Selección de Parámetros y DBSCAN

1. Selección de Parámetros:

MinPts:

$$MinPts = 2 \times dimensión + 1 = 5$$

Ɛ : distancia de conectividad

Método automático basado en las k-distancias (k = MinPts).

$$\varepsilon = \mu_k + \sigma_k \cdot \text{std\_scale}$$

2. Diseño del Grafo

Cada punto un es un vértice y se establece una arista entre dos vértices siempre que la distancia entre ellos sea menor o igual a épsilon.

El grafo se almacena en tres estructuras:

1. Vector de grados.
2. Lista plana de adyacencia o vecinos.
3. Vector de índices de adyacencia.

Complejidad:

CPU: $O(N^2)$

GPU: $\lim_{ncCUDA \to N} O\left(\frac{N^2}{ncCUDA}\right) = O(N)$ siendo blockDim $\cdot\, nSM = ncCUDA$

3. Propagación de Agregados (Breadth-first search)

Se aplica el recorrido en anchura mediante el BFS para recorrer el grafo y descubrir todos los puntos alcanzables desde cada punto núcleo.

Funcionamiento:

1. Recorrer los puntos.
2. Encontrar un punto núcleo no visitado e iniciar agregado.
3. Ejecutar el BFS desde el punto. (Paralelizado en GPU)
4. Cada punto visitado se etiqueta con identificador de agregado.
5. Una vez terminado el BFS se selecciona el siguiente punto núcleo no visitado y se repite el proceso.

IMPLEMENTACIÓN

Implementación y Optimizaciones

Herramientas

Software: Python, CUDA-C++, Numpy, CuPy

Decisiones Clave

256 hilos por bloque, distancias al cuadrado, minimización de transferencias CPU-GPU.

Versiones Desarrolladas

- **CPU Secuencial:** Implementación base en Python con NumPy
- **GPU CUDA:** Paralelización masiva con CuPy
- **CPU Numba:** Optimización JIT con vectorización automática
- **GPU + C:** Propagación optimizada en C

93695 puntos	Tiempo extracción de puntos	Tiempo cálculo de ε	Tiempo construcción de grafo	Tiempo de propagación agregados	Tiempo obtención de propiedades	Tiempo total
CPU	0.638 s	5898.471 s (1 h 38 min 18 s)	10378.591 s (2 h 52 min 59 s)	6.858 s	0.138 s	16284.696 s (4 h 31 min 25 s)
CPU - Numba	0.655 s	10.194 s	14.381 s	0.194 s	0.141 s	25.565 s
GPU	0.186 s	0.022 s	0.031 s	1.352 s	0.016 s	1.607 s
GPU – Propagación en C	0.180 s	0.022 s	0.031 s	0.116 s	0.015 s	0.364 s

Figura 4. Evaluación del rendimiento por etapa del DBSCAN (93 695) puntos entre CPU, CPU optimizada con Numba, GPU y GPU con propagación de agregados optimizada en C. .

VALIDACIÓN

Pruebas Realizadas

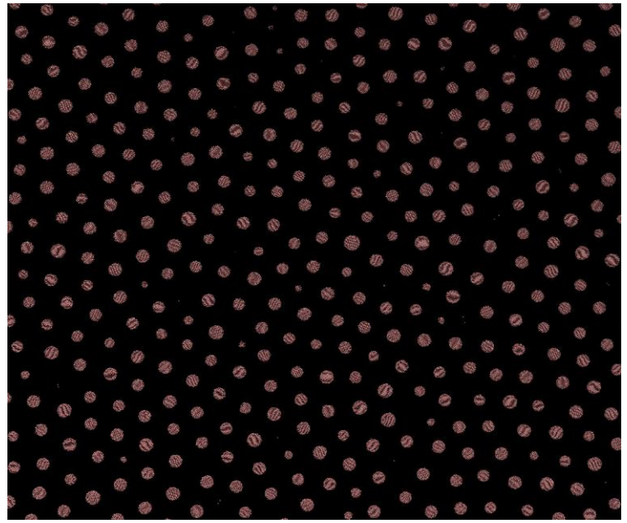


Figura 5. Imagen obtenida representando una configuración especifica de la trayectoria de agregados globulares en sistema con interacciones atractivas de corto alcance y repulsivas del largo alcance (SALR) radiales)

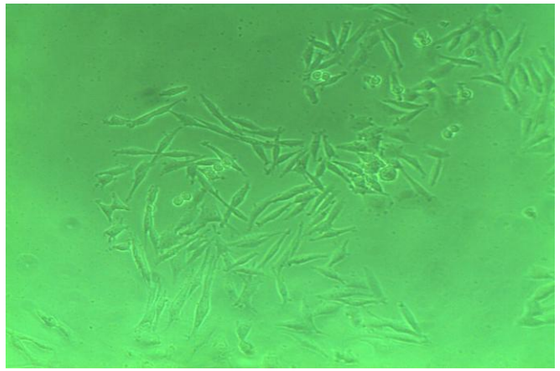


Figura 6. Imagen de microscopía de fluorescencia correspondiente a un cultivo in vitro de neuroglioma

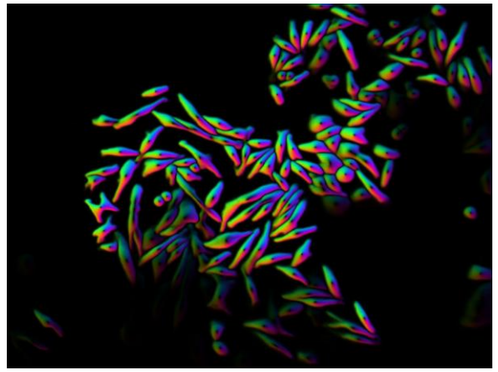


Figura 7. Resultado del proceso de segmentación celular mediante la herramienta Cellpose-SAM aplicado a la imagen de la Figura 6.

Simulaciones de Dinámica Molecular

Datos generados mediante el programa LAMMPS, pertenecientes a estudios del grupo de Modelización y Simulación Molecular del Instituto de Química Física Blas Cabrera.

Enfoques:

1. Análisis directo de las coordenadas en formato NETCDF (https://www.unidata.ucar.edu/software/netcdf).
2. Conversión en imagen mediante VMD (https://www.ks.uiuc.edu/Research/vmd/).

Muestras Biológicas

Fotografías reales de una muestra procedente de un cultivo *in vitro* de neuroglioma, proporcionadas por el Laboratory of Neurochemistry and Cell Biology y el grupo de Statistical Physics and Complex Systems de la Federal University of Bahia (Brasil).

Proceso de segmentación realizado mediante la herramienta Cellpose-SAM.

RESULTADOS

Análisis Tiempos de Cálculo

Análisis de los tiempos de las distintas muestras, todas con un valor de minPts = 5, y con la épsilon calculada por el código.

Figura/Archivo	Figura 17	Figura 18	Figura 21	Archivo de coordenadas	Figura 19
Puntos	40142	93695	107426	160000	176784
Tiempo total CPU (segundos)	3016.101	16284.696	21410.169	46292.948	58473.652
Tiempo total CPU – Numba (segundos)	6.616	25.565	30.946	63.654	83.013
Tiempo total GPU (segundos)	0.804	1.607	1.85	2.859	3.639
Tiempo total GPU - Propagación en C (segundos)	0.26	0.364	0.36	0.802	1.073

Figura 8. Tiempos de ejecución obtenidos para las distintas implementaciones del algoritmo DBSCAN .

Implementación	Ecuación de la recta
CPU	t = **0.4075**N- 18021
CPU – Numba	t = **0.0006**N - 21.778
GPU	t = **0.00002**N- 0.1545
GPU - Propagación en C	t = **0.000006**N - 0.1118

Figura 10. Ecuaciones de las rectas de regresión lineal correspondientes a cada implementación del algoritmo DBSCAN, obtenidas a partir de los tiempos de ejecución

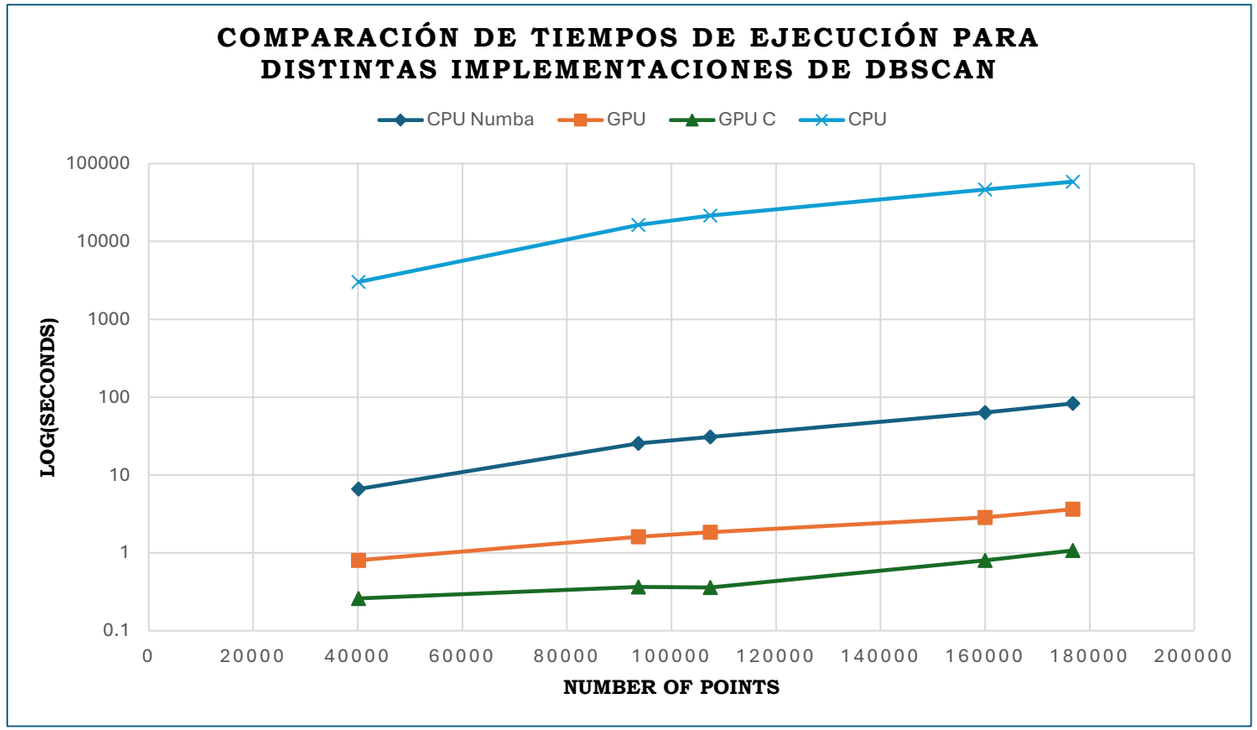


*Figura 9. Comparación de los tiempos de ejecución de las distintas implementaciones del algoritmo DBSCAN en función del número de puntos procesados. Representación **semi-log** para mejorar la visualización.*

CONCLUSIÓN

Conclusiones Generales

Éxito

Se logró una implementación funcional y validada capaz de procesar imágenes masivas en tiempo real.

Rendimiento

Las implementaciones en GPU superan ampliamente a las implementaciones en CPU.

Mejoras Futuras

1. Incorporar un módulo de preprocesado de imágenes.
2. Modelos basados en redes neuronales para una identificación más robusta de formas y estructuras específicas.
3. Implementación de estructuras espaciales como *linked cells,* para determinar los vecinos.

IMPACTO

Análisis de Impacto

Impacto Social y Cultural

Puede contribuir a mejorar procesos en ámbitos como la investigación biomédica, la detección de anomalías en imágenes clínicas o la monitorización de procesos biológicos

Objetivos de Desarrollo Sostenible

1. ODS 9 (Industria, innovación e infraestructura)
2. ODS 12 (Producción y consumo responsables)
3. ODS 3 (Salud y bienestar)
4. ODS 13 (Acción por el clima)

Impacto Económico y Medioambiental

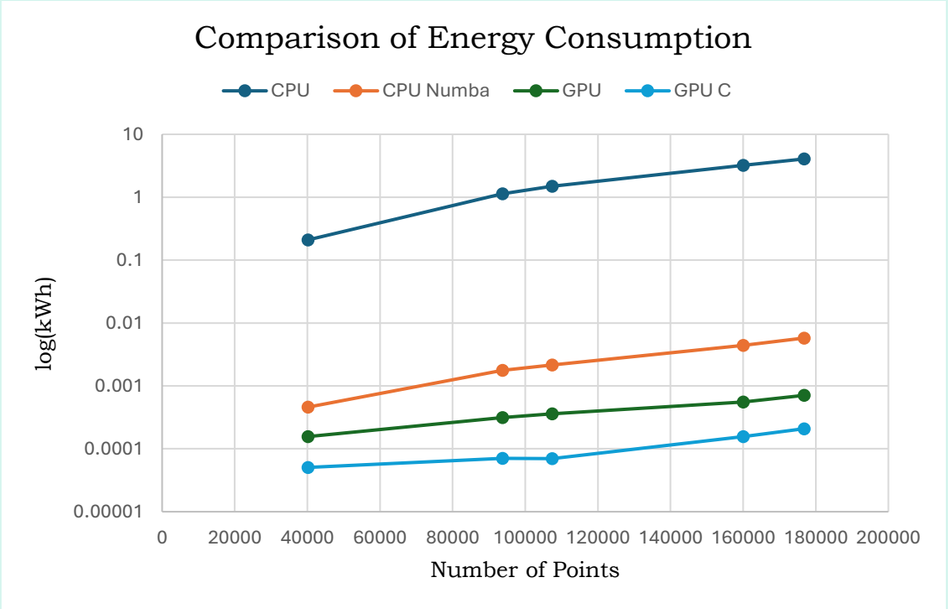


Figura 11. Comparación del consumo energético estimado para distintas versiones del algoritmo. .

¿Preguntas?

Muchas gracias por su atención