UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CARRERA DE TECNOLOGÍAS DE LA INFORMACIÓN

TRABAJO DE INTEGRACIÓN CURRICULAR

TEMA:

” EVALUACIÓN DE CONTRASEÑAS SEGURAS MEDIANTE ALGORITMOS DE FUERZA BRUTA EN ENTORNO DE COMPUTACIÓN DE ALTO RENDIMIENTO (HPC) ”

Trabajo de titulación previo a la obtención del título en Ingeniero en Tocologías de la información

Línea de investigación: Desarrollo, aplicación de software y cyber security (seguridad cibernética)

AUTOR:

Teddy Damian Pujota Rocha

DIRECTOR:

Ing. Marco Remigio Pusdá Chulde, PhD.

Ibarra – Ecuador 2026

UNIVERSIDAD TÉCNICA DEL NORTE
BIBLIOTECA UNIVERSITARIA

AUTORIZACIÓN DE USO Y PUBLICACIÓN A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

1. IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Art. 144 de la Ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

DATOS DE CONTACTO			
CÉDULA DE IDENTIDAD:	1725306318		
APELLIDOS Y NOMBRES:	Pujota Rocha Teddy Damian		
DIRECCIÓN:	San Juan de Calderon		
EMAIL:	tdpujotar@utn.edu.ec		
TELÉFONO FIJO:	2814758	**TELÉFONO MÓVIL:**	0962644415

DATOS DE LA OBRA	
TÍTULO:	EVALUACIÓN DE CONTRASEÑAS SEGURAS MEDIANTE ALGORITMOS DE FUERZA BRUTA EN ENTORNO DE COMPUTACIÓN DE ALTO RENDIMIENTO (HPC)
AUTOR (ES):	Teddy Damian Pujota Rocha
FECHA: DD/MM/AAAA	04/03/2026
SOLO PARA TRABAJOS DE GRADO	
PROGRAMA:	☒ **PREGRADO** ☐ **POSGRADO**
TITULO POR EL QUE OPTA:	Título de ingeniero en tecnologías de la información
ASESOR /DIRECTOR:	**Ing. Marco Pusdá PhD**

2. CONSTANCIAS

El autor (es) manifiesta (n) que la obra objeto de la presente autorización es original y se la desarrolló, sin violar derechos de autor de terceros, por lo tanto, la obra es original y que es (son) el (los) titular (es) de los derechos patrimoniales, por lo que asume (n) la responsabilidad sobre el contenido de la misma y saldrá (n) en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 04 días del mes de Marzo de 2026

EL AUTOR:

Firmado electrónicamente por:
TEDDY DAMIAN PUJOTA ROCHA
Validar únicamente con FirmaEC

.....

Teddy Damian Pujota Rocha

CERTIFICACIÓN DEL DIRECTOR DEL TRABAJO DE INTEGRACIÓN CURRICULAR

Ibarra, a los 04 dias del mes de marzo de 2026

Ing. Marco R. Pusdá, PhD
DIRECTOR DEL TRABAJO DE INTEGRACIÓN CURRICULAR

CERTIFICA:
Haber revisado el presente informe final del trabajo de Integración Curricular, el mismo que se ajusta a las normas vigentes de la Universidad Técnica del Norte; en consecuencia, autorizo su presentación para los fines legales pertinentes.

MARCO REMIGIO PUSDA CHULDE
Firmado digitalmente por MARCO REMIGIO PUSDA CHULDE
Fecha: 2026.03.04 14:44:40 -05'00'

Ing. Marco R. Pusdá, PhD

C.C: 0401200951

APROBACIÓN DEL COMITÉ CALIFICADOR

El Comité Calificado del trabajo de Integración Curricular “ Evaluación de contraseñas seguras mediante algoritmos de fuerza bruta en entorno de computación de alto rendimiento (HPC) ” elaborado por Teddy Damian Pujota Rocha , previo a la obtención del título del Ingeniero en tecnologías de la información , aprueba el presente informe de investigación en nombre de la Universidad Técnica del Norte:

MARCO REMIGIO PUSDA CHULDE

Firmado digitalmente por MARCO REMIGIO PUSDA CHULDE
Fecha: 2026.03.04 14:44:58 -05'00'

Ing. Marco R. Pusdá, PhD

C.C: 0401200951

Firmado electrónicamente por:
DAISY ELIZABETH IMBAQUINGO ESPARZA
Validar únicamente con FirmaEC

Ing. Daisy Imbaquingo, PhD

C.C: 1002873048

DEDICATORIA

Dedico este trabajo a mi familia, por su apoyo constante, su paciencia y la confianza que siempre depositaron en mí. Su respaldo incondicional fue fundamental durante cada etapa de este proceso.

A mis padres, por inculcarme la disciplina y la perseverancia necesarias para afrontar cada desafío académico y personal. Su ejemplo ha sido una guía permanente en mi formación.

También dedico este logro a quienes creyeron en mis capacidades incluso en los momentos de mayor exigencia, recordándome que el esfuerzo sostenido siempre produce resultados.

Teddy Damian Pujota Rocha

AGRADECIMIENTO

Expreso mi sincero agradecimiento a mi tutor y asesora, por su orientación académica, sus observaciones críticas y el acompañamiento constante durante el desarrollo de esta investigación. Sus aportes fueron esenciales para fortalecer la rigurosidad metodológica del estudio.

A la institución y a los docentes que formaron parte de mi proceso académico, por brindarme los conocimientos y herramientas que hicieron posible la realización de este trabajo.

Finalmente, agradezco a todas las personas que, directa o indirectamente, contribuyeron con su apoyo, motivación y confianza para culminar esta etapa profesional.

Teddy Damian Pujota Rocha

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

El aumento sostenido de la capacidad computacional ha modificado el panorama de la seguridad en sistemas basados en contraseñas. En particular, los ataques de fuerza bruta han incrementado su viabilidad mediante el uso de arquitecturas aceleradas como las GPU, lo que hace necesario analizar de manera controlada el impacto del modelo de ejecución sobre el rendimiento del algoritmo. Este trabajo presenta un diseño experimental orientado a comparar CPU y GPU utilizando la misma implementación, métricas homogéneas y condiciones constantes, con el objetivo de aislar el efecto de la arquitectura de cómputo. Los experimentos se desarrollaron en un entorno de nube bajo criterios de repetibilidad y validación estadística. Se evaluaron distintos tamaños de contraseña considerando tanto el tiempo medido como la capacidad real de completar la ejecución. Los hallazgos muestran que la arquitectura determina la factibilidad práctica del ataque, aportando evidencia empírica útil para el diseño de políticas de seguridad más robustas.

Palabras clave: CPU, computación de alto rendimiento, fuerza bruta, GPU, paralelismo, seguridad de contraseñas, validación experimental.

ABSTRACT

The sustained growth of computational capacity has reshaped the security landscape of password-based systems. In particular, brute-force attacks have increased their practical feasibility through the use of accelerated architectures such as GPUs, making it necessary to systematically analyze the impact of the execution model on algorithmic performance. This study presents a controlled experimental design aimed at comparing CPU and GPU architectures using the same implementation, homogeneous metrics, and constant testing conditions in order to isolate the effect of the computing architecture. The experiments were conducted in a cloud environment under repeatability criteria and statistical validation. Different password lengths were evaluated, considering both measured execution time and the actual capability to complete the task. The findings indicate that the underlying architecture determines the practical feasibility of exhaustive attacks, providing empirical evidence to support the design of stronger security policies.

Keywords: brute-force attack, CPU, experimental validation, GPU, high-performance computing, parallelism, password security.

CAPÍTULO I

INTRODUCCIÓN

1.1 Planteamiento del Problema

En la sociedad actualmente, la seguridad digital se ha vuelto un pilar fundamental debido a la creciente dependencia de aplicaciones que manejan datos sensibles, como banca en línea, redes sociales y otros servicios que almacenan información personal. Dentro de este marco, la seguridad informática adquiere especial relevancia, siendo las contraseñas uno de los principales mecanismos de protección para resguardar la privacidad e integridad de los datos. No obstante, en los últimos años se ha observado un aumento significativo de ciberdelitos, lo que evidencia la urgencia de fortalecer las medidas de seguridad existentes y concientizar a los usuarios sobre buenas prácticas digitales.

En el estudio realizado (Palatty) [1] Estados Unidos, sobre las causas más comunes de violaciones de datos, se encontró el uso indebido de privilegios o acción física; el 62 % fue el resultado de credenciales robadas. Dado que las personas usan contraseñas débiles, además de que la reutilización de la misma contraseña hace que facilite el acceso a la información para terceros.

La seguridad informática se ha convertido en un requisito indispensable para la vida contemporánea. Según el Digital 2023 Global Overview Report [2] el usuario promedio gestiona 8.7 cuentas digitales críticas (bancarias, médicas, gubernamentales), cada una protegida por contraseñas que, paradójicamente, siguen siendo el factor más vulnerable.

La Figura 1 muestra los principales factores que afectan la seguridad de las contraseñas, las cuales están divididas en cuatro factores:

1. Humanos: relacionados con la creación y gestión de contraseñas débiles por parte de los usuarios.
2. Técnicos: asociados a la implementación de algoritmos de cifrado obsoletos o deficientes.
3. Organizativos: vinculados a la adopción de políticas internas inadecuadas en la gestión de credenciales.
4. Externos: determinados por el incremento de ciberataques con técnicas cada vez más sofisticadas.

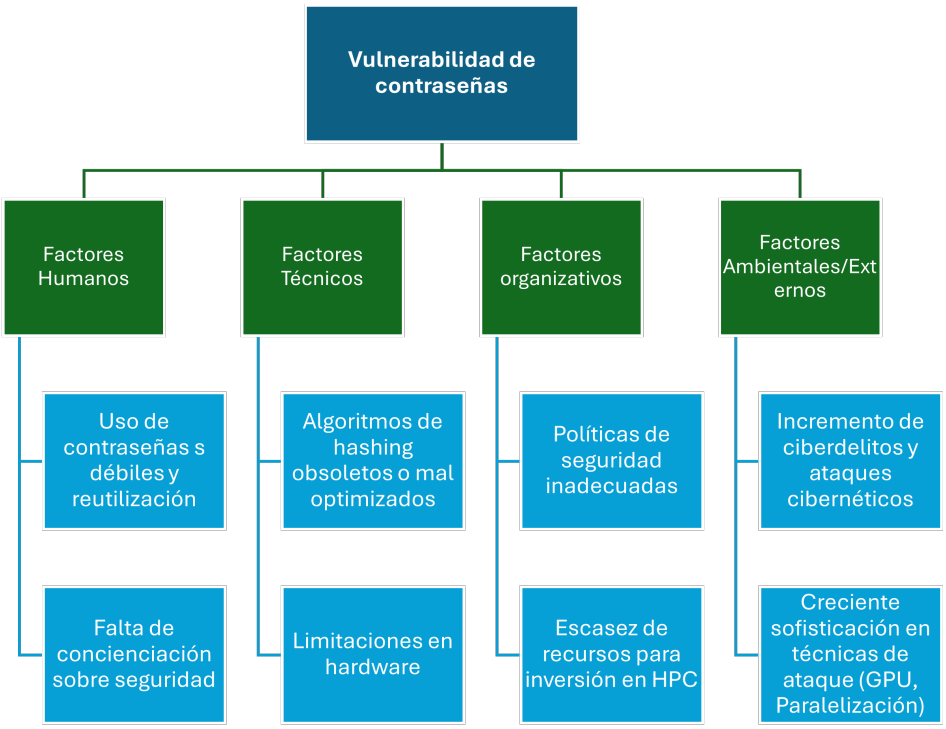

Figura 1.
Factores que provocan la vulnerabilidad de contraseñas

Los factores antes mencionados evidencian que la vulnerabilidad de las contraseñas es un problema multifactorial que requiere una solución integral.

Dicho problema se agudiza por dos factores que están relacionados; estos son el comportamiento humano y el estudio de Kaspersky Lab [3] los usuarios emplean contraseñas que combinan solo letras minúsculas y números, mientras que el 23 % utiliza secuencias simples como “123456”o “password”. Otro factor son los avances tecnológicos, como lo es el HPC, que hace

que una computadora ordinaria [4] pueda tomar la forma de supercomputadoras personalizadas y estas se utilizan para resolver problemas de alta demanda de recursos; para abordar este problema, se utilizarán algoritmos de fuerza bruta, herramientas de paralelización y el uso de HPC.

1.2 Objetivos

1.2.1 Objetivo General

Evaluar la eficiencia de algoritmos de fuerza bruta para la identificación de contraseñas seguras mediante computación de alto rendimiento para fortalecer la seguridad en aplicaciones informáticas.

1.2.2 Objetivos Específicos

- Documentar los principios de seguridad informática y las , herramientas relacionadas con algoritmos de fuerza bruta, enfocados en la evaluación de contraseñas seguras en entornos de computación de alto rendimiento (HPC High-Performance Computing).

- Implementar algoritmos de fuerza bruta utilizando computación de alto rendimiento para evaluar su rendimiento en diferentes escenarios.

- Evaluar la eficiencia y efectividad de algoritmos implementados mediante métricas para la identificación de contraseñas seguras

1.3 Alcance y delimitación

El presente trabajo se enfocó en el estudio y aplicación de algoritmos de fuerza bruta para la identificación de contraseñas seguras en entornos de computación de alto rendimiento (HPC); el proyecto abarcó desde la documentación de conceptos de seguridad y herramientas relacionadas hasta la implementación y evaluación de algoritmos optimizados.

Mediante el uso de unidades de procesamiento gráfico (GPU) y paralelización con herramientas de programación. Por ello se evaluó la eficiencia y efectividad de estos algoritmos utilizando métricas específicas en diferentes escenarios, con el objetivo de proporcionar una comprensión profunda de su rendimiento y mejoras posibles.

Para este estudio se utiliza una base de datos con diferentes contraseñas junto a HPC. Según P. Bordón y F. Crespo, HPC [5] "se entiende por compartir simultáneamente: tener recursos con varias horas de disponibilidad, infraestructuras de redes, seguridad y ejecución de programas computacionales". Se usará un lenguaje de programación paralela de alta velocidad junto a algoritmos de fuerza bruta para demostrar cuál es el tiempo estimado que se tarda en codificar una contraseña de seguridad.

Adicionalmente, se busca cuantificar la mejora en la eficiencia de estos algoritmos mediante el aprovechamiento de hardware especializado como GPUs y la optimización de procesos a través de técnicas de paralelización, lo cual permitirá identificar las limitaciones actuales y proponer soluciones que refuercen la seguridad en sistemas críticos.

1.4 Justificación

En la era digital actual, las contraseñas constituyen el mecanismo fundamental de autenticación para resguardar información confidencial y garantizar el acceso a servicios en línea. Desde transacciones bancarias hasta interacciones en redes sociales, toda esta información se encuentra protegida tras una barrera de seguridad cuya eficacia depende directamente de la robustez de las

credenciales utilizadas [6].

Sin embargo, la creciente sofisticación de los ciberataques y la capacidad de procesamiento de las tecnologías modernas han puesto en evidencia la vulnerabilidad de las contraseñas, especialmente de aquellas que son débiles o predecibles. Según el artículo de auditoría de seguridad [1] “las 10 contraseñas más usadas en 2023 son: “123456”, “123456789”, “qwerty”, “password”. “12345”, “qwerty123”, “1q2w3e”, “12345678”, “111111”, “1234567890, lo cual facilita su compromiso mediante ataques de fuerza bruta. Los ataques de fuerza bruta implican probar exhaustivamente todas las combinaciones posibles hasta encontrar la contraseña correcta; esto se ha vuelto más viable con el uso de hardware especializado, como lo son las Unidades de Procesamiento Gráfico (GPU). “Una unidad de procesamiento gráfico (GPU) es un dispositivo que es eficaz para el procesamiento paralelo de cálculos utilizando muchos subprocesos; esto permite realizar múltiples operaciones en paralelo”[7], acelerando significativamente el proceso de desencriptación.

La computación de alto rendimiento (HPC) ofrece herramientas y técnicas para aprovechar al máximo las capacidades de hardware disponibles, “es la práctica de agregar recursos de procesamiento para obtener un rendimiento mayor que el de una sola estación de trabajo”[4]. En este contexto, la implementación de algoritmos de fuerza bruta optimizados para entornos de HPC es crucial para evaluar la seguridad de las contraseñas y desarrollar una mejor creación de contraseñas seguras.

El presente trabajo se justifica por la necesidad de comprender y mejorar la eficiencia de algoritmos de fuerza bruta en la identificación de contraseñas seguras. Al documentar conceptos de seguridad y herramientas relacionadas, utilizando las técnicas de HCP, junto a herramientas de paralelización, “con esta tecnología se consigue que los desarrolladores puedan explotar el paralelismo en aplicaciones de la forma más eficiente posible”[4].

Finalmente, la evaluación de la eficiencia y efectividad de estos algoritmos mediante métricas específicas proporcionará información valiosa para el desarrollo de sistemas de seguridad más robustos. Los resultados de esta investigación beneficiaron a diversos sectores:

- Académico: Se proporcionó material actualizado y relevante para futuras investigaciones en el área de ciberseguridad, especialmente en el desarrollo de métodos más seguros de autenticación.
- Social: Se concientizó a la sociedad sobre la importancia de utilizar contraseñas seguras y las implicaciones de no hacerlo, promoviendo prácticas más seguras en el uso de la tecnología, "como la importancia de usar la autenticación multifactor"[8].
- Educativo: Se fomentó la educación en ciberseguridad, preparando a profesionales más capacitados para enfrentar los desafíos actuales en la protección de la información.

En esta propuesta se abordan problemas específicos como los siguientes:

- La necesidad de utilizar contraseñas robustas "de tener al menos 14-16 caracteres de longitud y consistir en diferentes letras"[9].
- La necesidad de optimizar algoritmos de fuerza bruta para evaluar la seguridad de contraseñas en entornos HPC, que se refiere en general a la práctica de agregar potencia de cálculo [10].

Desde una perspectiva técnica y teórica, este trabajo aporta:

- Implementación de algoritmos de fuerza bruta utilizando paralelización con lenguajes para computación de alto rendimiento y mejorando su rendimiento con HPC.
- Identificación de prácticas y configuraciones que aumenten la resistencia de las contraseñas ante ataques de fuerza bruta.

Este trabajo proporciona conocimientos prácticos en:

- Desarrollo de habilidades para programar y optimizar algoritmos utilizando herramientas de paralelización y aprovechando las capacidades de las GPU.

- Capacidad para evaluar la robustez de contraseñas seguras identificando vulnerabilidades y proponiendo mejoras.
- Experiencia en el uso de herramientas y técnicas de computación de alto rendimiento.

CAPÍTULO II

MARCO TEÓRICO

En la sociedad actual debemos entender lo que es la seguridad informática, ya que se ha vuelto un pilar importante para garantizar la confidencialidad, integridad y disponibilidad de la información. El aspecto que más ha cobrado relevancia son las contraseñas seguras, por la razón de que es nuestra primera línea de defensa, según Matt Bishop [11] "la seguridad tiene un valor binario: es seguro o no lo es"; para verificar la seguridad de contraseñas, se lo hace mediante una evaluación con algoritmos de fuerza bruta en un entorno de computación de alto rendimiento (HPC).

En este capítulo se establecen los fundamentos teóricos que sustentan la base de la investigación; se abordan conceptos como la seguridad informática y también se abordan las herramientas, como lo son los algoritmos de fuerza bruta, los lenguajes con paralelismo y el funcionamiento de los entornos de computación de alto rendimiento.

2.1 Antecedentes

Una investigación realizada por Fablán Calcedo, [12], abordó la creciente complejidad de los problemas actuales que demandan altos niveles de procesamiento, como las simulaciones climáticas y el análisis de big data en biología molecular, lo que ha impulsado la necesidad de estrategias más potentes en entornos de computación de alto rendimiento (HPC). Además, el estudio tuvo como objetivo explorar de forma integral dichas estrategias; aporta una visión amplia sobre el estado actual de HPC y sus posibles direcciones a futuro.A través de una revisión bibliográfica estructurada por conceptos clave como funcionamiento, utilidad y ventajas, se identificaron enfoques avanzados en HPC capaces de superar las limitaciones de rendimientos actuales, por lo que encontramos entre ellos la computación paralela, distribuida, cuántica, neu-

romórfica y en la nube, por lo que los resultados evidencian que el uso de HPC es esencial para potenciar el avance tecnológico en diversas áreas científicas y técnicas, la investigación concluye que, aunque estas estrategias representan una evolución significativa en el rendimiento computacional, aún enfrentan retos importantes relacionados con la escalabilidad, además de la complejidad y la seguridad, pero que su integración futura podría fortalecer de manera decisiva la eficiencia y confiabilidad en distintos sistemas de alto impacto.

En el trabajo de Pedro Fernández [13], analizó las limitaciones del uso de OpenCL en la herramienta de Hashcat, especialmente en dispositivos de NVIDIA, por lo que propuso como alternativa la implementación de CUDA para mejorar el rendimiento en ataques con algoritmos de hashing. El objetivo de este estudio fue examinar el funcionamiento de Hashcat para evaluar su rendimiento con OpenCL y desarrollar versiones optimizadas utilizando paralelización con CUDA, así comparando luego los resultados obtenidos con los de la versión original. Además de esto, se implementó una metodología SCRUM organizada en sprints y mediante el uso de herramientas como Gprof e Intel Vtune, se modificaron partes claves del código fuente para adaptarlo a la ejecución en CUDA con las configuraciones ajustadas de grid y bloques en GPUs Tesla K20m y K40c. El trabajo permitió implementar diferentes modos de ataque, como fuerza bruta, diccionario y combinatorio, por lo que se lograron mejoras notables en algoritmos como MD5 y SHA3-256 gracias a una mejor distribución del trabajo entre hilos y, como conclusión, se determinó que CUDA puede ofrecer un rendimiento superior al de OpenCL en varios escenarios, aunque esto implica una reestructuración compleja del código y un manejo preciso de las técnicas de paralelización, el estudio recomienda continuar el desarrollo con ajustes más específicos y a un acceso más cercano con el repositorio de Hashcat para facilitar su integración.

El trabajo de SeongJun Choi et al. [7],se centró en el análisis del algoritmo Scrypt, diseñado para resistir ataques mediante hardware especializado como ASICs y GPUs, y en cómo se debe aprovechar de forma eficiente su ejecución en entornos paralelos. El objetivo de este trabajo fue optimizar este algoritmo mediante la implementación de cuatro modelos distintos de cómputo adaptos a la GPU para poder evaluar combinaciones para diferentes valores de parámetro P con el fin de maximizar el rendimiento. Además, la metodología usada en este trabajo consistió en

implementar variaciones del algoritmo, trasladando parcialmente la función PBKDF2 a GPU y aplicando mejoras específicas a scryptROMix, lo que representa la mayor carga computacional adicional, se incorporaron técnicas de uso eficiente de memoria compartida y constante para buscar reducir los conflictos de accesos y mejorar la ejecución paralela, y los resultados mostraron las mejoras del 204.84 % y 168.14 % para los valores p=2 y p=4 respectivamente por lo que, en comparación con una implementación simple de Scrypt en GPU, los aumentos fueron significativos respecto a la ejecución en CPU y con las mejoras superiores al 8000 % y el estudio concluyó que, al modificar la distribución de procesamiento y aprovechar la arquitectura de GPU, es posible escalar de forma eficiente el rendimiento del algoritmo Scrypt, lo que marca un precedente de cómo optimizar el algoritmo en este tipo de entornos.

La investigación de Paola Bordón et al., [14], abordó la ejecución de modelos estadísticos complejos y simulaciones espaciales en investigaciones que manejan grandes volúmenes de datos, como lo es el análisis de más de setenta mil estudiantes mediante modelos logit anidados y la simulación de patrones de protesta utilizando datos georreferenciados extraídos de Twitter, el objetivo principal fue evaluar la capacidad de NHLPC para ejecutar este tipo de estudios de forma eficiente, lo que demuestra que es posible alcanzar estándares internacionales en disciplinas que tradicionalmente no se asocian al uso de la supercomputación, y la metodología que se usó es de Stata para ejecutar modelos de logit con múltiples variables y R para aplicar algoritmos de captura-recaptura espacial en big data. Gracias a los recursos del NLHPC, se logró ejecutar 120 modelos con exactitud en un periodo de pocos días, cada uno con una duración de hasta 72 horas, lo que resultó en un error de predicción inferior al 1 % en los modelos educativos y permitió identificar patrones no lineales en eventos de protesta, se concluyó que el uso de infraestructuras como el NLHPC no solo permite realizar investigaciones de alta complejidad técnica, sino que también facilita la integración de enfoques multidisciplinarios, posicionando a los investigadores en un nivel competitivo a escala internacional.

En el estudio hecho por Alkhwaja et al., [15], se enfocó en la evaluación del rendimiento de distintas técnicas de descifrado de contraseñas mediante ataques de fuerza bruta y ataques de diccionario utilizando programación paralela, por lo que el estudio partió de la necesidad

de evidenciar las vulnerabilidades de contraseñas débiles dentro del campo de la seguridad informática y se propuso analizar cuál enfoque resulta más eficiente al ejecutarse en diferentes configuraciones de hardwarepor lo que la metodología incluyó cuatro experimentos donde se implementaron ataques paralelos tanto en Python como en C++ con OpenMP y se usaron herramientas como Hashcat para comparar el desempeño entre dos GPUs, y los resultados mostraron que la GPU NVIDIA GeForce GTX 1050 TI tuvo un rendimiento significativamente superior al de la GPU Intel HD Graphics 630, alcanzando mejoras de velocidad de hasta 11.5x en contraseñas con caracteres especiales, también se registraron mejoras en el algoritmo de fuerza bruta con seis núcleos y en el ataque de diccionario usando planificación estática con ocho núcleos, y la conclusión principal del estudio fue que el uso de programación paralela incrementa notablemente la velocidad y efectividad de las técnicas de descifrado, por lo que representa una herramienta clave para fortalecer las estrategias de defensa frente a amenazas cibernéticas.

En el trabajo elaborado por Manuel Guiracocha et al., [16], se analizó la factibilidad de utilizar GPU como alternativa para mejorar la eficiencia de algoritmos de optimización metaheurísticos que enfrentan desafíos de rendimiento al aplicarse en problemas de gran complejidad, y el objetivo fue evaluar si herramientas compatibles con Python, como CuPy, que permiten acelerar la ejecución del algoritmo PSO al resolver casos como el problema del viajante de comercio (TSP) y la planificación de expansión de redes eléctricas (TEP), a esto comparando su comportamiento con implementaciones tradicionales en CPU. La metodología incluyó pruebas entre bibliotecas como Numba, Theano y CuPy, así como la implementación de PSO en ambos entornos, midiendo los tiempos de ejecución en función de la cantidad de partículas e interacciones, además de que los resultados indicaron que, en el caso del TSP, se logró una mejora clara en rendimiento al analizar más soluciones en menos tiempo usando GPU, mientras que en el problema TEP no se obtuvieron beneficios debido a las limitaciones en las bibliotecas de Python y al costo de comunicación entre CPU y GPU, por lo que el estudio concluyó que, aunque el uso de GPU puede ser ventajoso en problemas específicos, la efectividad depende del tipo de tarea y del entorno de desarrollo en el cual se realizan, por lo cual se sugiere reestructurar los algoritmos y considerar lenguajes con mayor control del hardware, como C++, para obtener mejores resultados en escenarios complejos.

La investigación realizada por Verma et al. [17], se centró en analizar la resistencia de distintos algoritmos de la familia SHA frente a ataques de fuerza bruta. Ante esta preocupación creciente de que estos métodos sigan evolucionando a pesar de los avances en seguridad criptográfica, el objetivo de este estudio fue calcular el número de combinaciones posibles para cada variante SHA y determinar si sería factible descifrarlos utilizando tecnologías actuales. Para esto se consideraron los fines éticos y destructivos, por lo cual la metodología consistió en un análisis teórico de los algoritmos, por lo que se consideró la longitud de sus hashes, el total de combinaciones posibles y el tiempo requerido para realizar ataques de fuerza bruta, notando incluso mejoras significativas en la velocidad de los procesamientos. Los resultados indicaron que SHA-0 y SHA-1 presentan mayor vulnerabilidad debido a su menor longitud de hash; por otro lado, SHA-256 y especialmente SHA-512 ofrecieron niveles de resistencia significativamente superiores, por lo que se concluyó que la posibilidad de descifrar algoritmos como SHA-512 mediante fuerza bruta es prácticamente nula con los recursos disponibles en la actualidad, por lo cual se respalda su uso como medida sólida para la protección de sistemas informáticos.

En el estudio hecho por Nair et al. [18], propuso una estrategia innovadora para fortalecer la resistencia de los hashes de contraseñas frente a ataques de fuerza bruta, los cuales superan las limitaciones de las funciones adaptativas tradicionales, cuya efectividad está condicionada por la tolerancia del usuario a la latencia, y el objetivo del estudio fue presentar una función de hash de credenciales multifactor (MFCHF) que integra los factores como HOTP, TOTP y OOBA, esto sin la necesidad de modificar el hardware o software de autenticación existentes ni comprometer la experiencia del usuario. Para esto se diseñaron cuatro versiones prácticas de MFCHF y se evaluó su resistencia comparándola con funciones estándar como bcrypt, Argon2 y scrypt, utilizando una aplicación web como prueba de concepto, y los resultados demostraron mejoras sustanciales, en especial las combinaciones con HOTP y TOTP, donde el tiempo estimado para descifrar una contraseña pasó de minutos a varios años. En caso de OOBA o HMAC-SHA1, no se logró romper ninguna clave en 24 horas de ataque, por lo que el estudio concluyó que MFCHF permite una resistencia asimétrica efectiva frente a ataques de fuerza bruta por lo que se logran niveles de seguridad hasta un millón de veces mayores que las funciones adaptativas comunes y manteniendo compatibilidad con herramientas ampliamente utilizadas como Google

Authenticator o YubiKeys.

En la investigación de Beno et al. [19], examinó la efectividad de las políticas mínimas de contraseñas adoptadas por diversos sitios web con el fin de identificar qué tan vulnerables resultan las credenciales generadas bajo esos parámetros frente a ataques con herramientas especializadas. Dicho estudio analizó las políticas de catorce plataformas donde se crearon contraseñas mínimas aceptadas por cada una y posteriormente 58 usuarios generaron contraseñas realistas siguiendo esas mismas políticas y estas contraseñas fueron sometidas a pruebas mediante ataques de fuerza bruta y diccionario utilizando Hashcat en una máquina equipada con cuatro GPUs y se evaluó el nivel de resistencia ante dichos ataques, por lo que los resultados mostraron que todas las contraseñas mínimas fueron vulneradas con facilidad e incluso en cuestión de segundos. Por otro lado, las contraseñas realistas ofrecieron mayor resistencia, pero muchas también fueron descifradas con configuraciones adecuadas, por lo que se concluyó que las políticas débiles favorecen la creación de contraseñas fácilmente vulnerables y se recomienda establecer requisitos más estrictos, como una longitud mínima de ocho caracteres, el uso de alfabetos variados, la prevención de patrones repetitivos y la verificación frente a bases de datos de contraseñas comprometidas para reducir el riesgo de reutilización.

En la investigación de Ciccozzi et al. [20], se orientó a la complejidad del panorama actual de lenguajes de programación orientados a la computación paralela por la razón de la creciente demanda de procesamiento en tiempo real en sistemas intensivos en software, como lo son los vehículos autónomos o aplicaciones médicas, entre unos cuantos ejemplos. Este estudio tuvo como objetivo identificar, clasificar y analizar las principales tendencias, además de los desafíos de los lenguajes de alto nivel utilizando la programación paralela, excluyendo aquellos de bajo nivel o centrados únicamente en verificación formal. Para esto se llevó a cabo una revisión sistemática de la literatura en tres fases, recompilando inicialmente más de tres mil estudios, de los cuales se seleccionaron 225 trabajos primarios. A partir de este análisis se evidenció un aumento sostenido del interés en estos lenguajes desde 1988, en el cual se destaca el uso de extensiones de C/C++ o Java bajo paradigmas imperativos y orientados a objetos, así como un predominio de paralelismo explícito en arquitecturas multi-core y many-core, no obstante, se

identificaron barreras importantes para su adopción, como la falta de estandarización, la escasa madurez de las herramientas y la limitada compatibilidad en arquitecturas heterogéneas, y el estudio concluyó que es necesario fomentar el desarrollo de lenguajes más accesibles y eficientes que incorporen modelos de comunicación avanzados y respondan mejor a las necesidades de los ingenieros de software que trabajan en entornos de computación paralela.

2.2 Contraseñas seguras

2.2.1 Principios de seguridad

La seguridad informática se ha convertido en una necesidad esencial en cualquier entorno donde se gestiona la información digital. El propósito de esto es proteger los datos frente a accesos no autorizados, daños o pérdidas de información que es relevante para el usuario en cuestión. Esta área no solo se limita a proteger archivos, sino que también ayuda a prevenir vulnerabilidades en redes, programas y dispositivos que se usan de manera continua; hoy en día, con el crecimiento constante de la tecnología y la interconexión global, las amenazas se han vuelto mucho más complejas y mucho más frecuentes. En palabras de Jain et al., "el objetivo de la ciberseguridad es garantizar la protección de aplicaciones, redes, computadoras e información crítica contra ataques"[21]. Esta afirmación resalta la importancia de contar con estrategias sólidas de defensa digital.

2.2.2 Tipos de ataques

El mal uso de las contraseñas ha sido una de las causas más recurrentes en los ataques de seguridad; existen muchos tipos de ataques que buscan obtener las contraseñas de los usuarios; entre los más conocidos están los ataques por diccionario, los ataques de fuerza bruta y los ataques por phishing.

Uno de los errores más comunes es utilizar datos personales al momento de la creación de las

contraseñas; en palabras de Tremblay-Savard et al., “es fácil para los atacantes adivinar contraseñas cuando los usuarios eligen información personal como su ID o número de teléfono”[22]. Esto sucede porque los atacantes suelen probar primero con este tipo de combinaciones que tienen información personal, lo cual no está tan protegido. Otro riesgo que se ha evidenciado es la manera en que los usuarios almacenan sus contraseñas; en palabras de los mismos autores, “a veces las escriben en notas adhesivas o las guardan en dispositivos, lo que representa un terrible riesgo de seguridad”[22], estas son prácticas que aumentan la posibilidad de que una contraseña caiga en manos equivocadas, lo que genera grandes riesgos a la información que se trata de proteger.

2.2.3 Políticas de complejidad

Para enfrentar los problemas anteriormente explicados, existen recomendaciones y políticas que ayudan a mejorar la seguridad de contraseñas; de esta manera se obtiene el equilibrio entre la seguridad y la facilidad de uso. Shay et al. afirma que “una política de contraseñas bien escrita puede proporcionar una mayor seguridad, aunque una política demasiado estricta puede generar comportamientos inseguros como escribirlas o compartirlas”[23], lo que demuestra que no solo se trata de exigir contraseñas complejas, sino que los usuarios realmente puedan usar contraseñas seguras, además de que sean fáciles de recordar.

Una solución interesante es la generación automática de contraseñas fuertes usando información personalizada con algún sistema o programa según el enfoque de Tremblay-Savared et al.,“las contraseñas generadas por nuestro sistema son fáciles de recordar, pero también presentan características de alta resistencia frente a ataques de fuerza bruta y diccionario”[22], este tipo de enfoques pueden llegar a ser útiles para resolver el conflicto entre seguridad y memorización; el usar aplicaciones que generen contraseñas es buena práctica, ya que no se usa información personal y las combinaciones que se pueden crear son más robustas y seguras. Además de esto, se ha comprobado que las contraseñas generadas automáticamente pueden alcanzar altos niveles de resistencia frente a ataques automatizados. Según Tremblay-Savard et al., se tiene “94.88 % de efectividad y una resistencia de más de 90 días frente a ataques de fuerza bruta”[22].

2.2.4 Buenas prácticas de almacenamiento

Además, es preocupante que los usuarios reutilicen contraseñas, como lo señalan Shay et al., "incluso cuando los usuarios cumplen con requisitos más estrictos, tienden a reutilizar contraseñas o a modificarlas ligeramente en diferentes plataformas"[23], lo que da paso a que, si una contraseña se ve comprometida en un sistema, puede poner en riesgo a las demás cuentas, las cuales tienen cambios pequeños. Este hábito da pie a un punto débil importante a considerar dentro de cualquier análisis que sea de seguridad.

Una de las soluciones propuestas para este problema es evitar por completo la repetición de claves. Esto se puede lograr apoyándose en generadores automáticos de contraseñas fuertes; como sostienen Tremblay-Savard et al., "la generación sistemática de contraseñas fuertes permite al usuario no reutilizar claves previas y reduce la necesidad de almacenarlas manualmente"[22].

Existen varias estrategias para la protección de contraseñas almacenadas y esta técnica es conocida como salting; consiste en añadir una cadena aleatoria que sea única a cada contraseña antes de que se aplique un algoritmo hash. Esto evita que dos contraseñas iguales produzcan un único valor cifrado; además, eso es muy importante para prevenir los ataques basados en tablas precalculadas como las rainbow tables. En palabras de Hallett et al., "agregar un salt hace que cada hash de contraseña sea único, incluso si dos usuarios eligen la misma contraseña"[24]. Esta práctica hace difícil que los atacantes comparen hashes y se mejora notablemente la protección de las bases de datos.

Otra medida útil para la seguridad de las contraseñas consiste en categorizar las cuentas según su nivel de sensibilidad, por lo que se pueden asignar diferentes tipos de estrategias de almacenamiento dependiendo de la información que maneje el usuario. Como argumenta Hamzha Hussain "una buena estrategia que los usuarios pueden usar es dividir sus cuentas en cuentas de alta sensibilidad y baja sensibilidad"[25], esta separación puede permitir al usuario priorizar aquellas contraseñas que requieren una mayor protección sin verse saturado por todas las contraseñas que debe recordar.

El uso de administradores de contraseñas es otra práctica aceptada y sobre todo recomendada porque este tipo de sistemas permiten gestionar contraseñas complejas sin que el usuario tenga que recordarlas. Como lo menciona Hamzha Hussain “los administradores de contraseñas cifran tus contraseñas y protegen tu información; además, cada cuenta utilizada tendrá una contraseña única de más de 16 caracteres generada automáticamente que el usuario no necesita memorizar ni siquiera ver”[25], este tipo de estrategias reduce la reutilización de contraseñas y el riesgo de filtraciones. Sin embargo, este tipo de gestores también influyen en la seguridad general, por lo que los gestores locales ofrecen un mayor control sobre las contraseñas, pero los gestores en la nube presentan beneficios de sincronización, lo que ayuda al usuario. Por otro lado, el uso de la nube también tiene un riesgo “los gestores de almacenamiento local pueden operar completamente fuera de línea y, por lo tanto, no son susceptibles a ataques de red; esto te convierte en un objetivo mucho menos para los hackers”.[25]. Además de esto, se puede usar la protección por dos pasos o también conocido como 2FA.

Además de las recomendaciones individuales que encontramos sobre la creación y el uso de las contraseñas, es fundamental considerar los marcos normativos que nos ofrecen los lineamientos más amplios para la seguridad de la información. Los estándares internacionales como la familia ISO/IEC 27000 proporcionan guías estructuradas para gestionar de forma segura los archivos digitales y esto incluye las contraseñas que usamos para el mismo propósito. De acuerdo con Tamimi et al. “Los estándares ISO/IEC 27001 y 27002 permiten establecer políticas de seguridad para proteger los activos de información, construir, implementar, monitorear y mejorar la eficacia del sistema de gestión de seguridad de la información (ISMS)”[26]. Este enfoque no solo establece medidas técnicas, también establece técnicas operativas y administrativas que aseguran la confidencialidad, integridad y disponibilidad de los datos; asimismo, se promueve la adopción de controles organizacionales para evitar las vulnerabilidades comunes, como accesos no autorizados y fallas en la autenticación, e incorpora estas buenas prácticas dentro de una política institucional que fortalece significativamente la postura de seguridad de cualquier entorno digital.

2.3 Algoritmos de Fuerza Bruta

2.3.1 Fundamentos y funcionamiento

El ataque de fuerza bruta se fundamenta en la generación secuencial y exhaustiva de posibles soluciones, esto sin necesidad de conocimiento previo al valor o, en este caso, de la contraseña buscada. Como lo explican Alkhwaja et al., "un algoritmo de fuerza bruta genera todas las posibles combinaciones de caracteres en un rango y longitud especificados"[15], esto lo convierte en una técnica aplicable incluso en escenarios donde no se dispone de ninguna información previa sobre la clave que se quiere descifrar.

Este tipo de ataques es exhaustivo por naturaleza y precisamente por eso es efectivo en sistemas donde las contraseñas son débiles, pero su efectividad también puede disminuir drásticamente a medida que aumenta la complejidad de la combinación y del conjunto de caracteres que está buscando.

La estimación del tiempo requerido para descubrir contraseñas mediante ataques sistemáticos permite dimensionar de manera más clara su nivel de vulnerabilidad, por lo que este tipo de análisis resulta clave al momento de identificar patrones débiles en contraseñas aparentemente robustas. Autores como Saputra et al. plantean que "la simulación estima el tiempo requerido para adivinar una contraseña, dependiendo del nivel de complejidad"[27].

Además del enfoque tradicional, también se pueden aplicar técnicas optimizadas para diferentes modos de ataque, lo que permite evaluar no solo todas las combinaciones posibles, sino también hacerlo en menos tiempo, dependiendo del modo a usar. Fernández-Arruti explica que "los modos de ataque que permite Hashcat son: por diccionario, por máscara, híbrido, combinaciones y fuerza bruta. Cada uno tiene sus ventajas y se usan según el contexto de la clave"[13].

2.3.2 Tipos de algoritmos

En el contexto de los ataques de fuerza bruta, se han desarrollado diferentes algoritmos que nos sirven para reforzar o vulnerar contraseñas mediante diferentes métodos; estos algoritmos tienen distintas complejidades, eficiencia y nivel de resistencia frente a intentos de descifrado.

Uno de los más utilizados es PBKDF2, el cual está diseñado para aumentar la dificultad de los ataques sistemáticos mediante funciones hash y múltiples iteraciones. Como sostienen Ertaul et al., "PBKDF2 trabaja sobre una función pseudorandom (PRF) con un número fijo de iteraciones, toma una sal, una contraseña definida por el usuario y la longitud deseada de la clave de salida como entrada"[28], esta arquitectura permite ralentizar de forma involuntaria el proceso de derivación de claves, lo cual dificulta los ataques paralelos.

Otro algoritmo usado es Bcrypt; tiene una estructura de subclaves que aumenta el costo computacional de cada intento. Tal como lo señalan Ertaul et al., "Bcrypt tiene una configuración de clave costosa y es una utilidad de cifrado multiplataforma; por lo tanto, es compatible con cualquier sistema operativo. Bcrypt utiliza EBC (Electronic Code Block) y divide los datos de entrada en subclaves, comenzando luego el cifrado por bloques de dichas subclaves"[28], este enfoque permite ajustar de una forma dinámica la carga computacional y esto resulta útil frente a mejoras de hardware por parte de atacantes.

En cuanto a Scrypt, este algoritmo agrega un consumo intensivo de memoria al proceso de derivación, conforme a lo dicho por Ertaul et al., "Scrypt es un algoritmo de derivación de claves que hace uso intensivo de la CPU y la memoria. También se lo conoce como cifrado simétrico basado en contraseñas. Scrypt genera grandes vectores de cadenas de bits pseudorandom almacenadas en memoria RAM, lo cual dificulta los ataques debido a la necesidad de grandes recursos computacionales"[28], este diseño lo convierte en una opción sólida para entornos donde se requiere resistencia frente a ataques distribuidos o con hardware especializado.

Adicionalmente, se han desarrollado enfoques que manipulan variantes de contraseñas y las combinan con técnicas de salting para generar múltiples versiones de hashes, esto con el objetivo

de aumentar la probabilidad de éxito en ataques por diccionario. Como lo explican Hussian y Al-shareefi, “el algoritmo SHA-1 se emplea para cifrar contraseñas, y cuando se utiliza una sal, las contraseñas y la sal simplemente se concatenan como sal + contraseña”[29], este procedimiento permite aplicar transformaciones a la contraseña original y generar múltiples vectores de ataque.

En palabras de Hussian y AL-Shareefi, “las 10 versiones posibles de hash incluyen: SHA1 (password), SHA1 (adoprssw), SHA1 (wssrpoda), SHA1 (drowssap), SHA1 (psswrd), SHA1 (salt + password), SHA1 (salt + drowssap), SHA1 (salt + adoprssw), SHA1 (salt + wssrpoda), y SHA1 (salt + psswrd)”[29], este tipo de variaciones busca anticiparse a patrones comunes usados por los usuarios, lo que demuestra la necesidad de considerar estos enfoques en el análisis de algoritmos de ataque.

Estas variantes no solo permiten elegir el método más adecuado según el entorno; lo que esto refleja es la evolución técnica de los algoritmos de fuerza bruta desde enfoques tradicionales hasta estrategias más adaptativas y específicas.

Fuera de los métodos clásicos, existen las técnicas híbridas que combinan ambos enfoques para mejorar los resultados a obtener, y estas técnicas suelen comenzar con contraseñas comunes y luego aplican variaciones como reemplazos de caracteres o números al final. Como lo explica Ertaul et al., “una técnica híbrida puede iniciar con un diccionario de contraseñas comunes y luego aplicar permutaciones como números añadidos o sustituciones de letras”[28]. Este enfoque permitió explorar un mayor número de combinaciones sin incrementar en exceso el tiempo de procesamiento, lo que es útil para atacar contraseñas mal diseñadas pero parcialmente complejas.

2.3.3 Herramientas

El avance del hardware y el software ha permitido el desarrollo de herramientas especializadas para ejecutar ataques de fuerza bruta de una forma que sea más eficiente. Una de las herramientas más utilizadas es Hashcat; esta aplicación puede aprovechar GPUs modernas para realizar millones de intentos por segundo. Como lo señala Laatansa et al., “la fuerza bruta des-

cifró 770.884 contraseñas cortas (de 6 a 7 caracteres)"[30], lo cual demuestra su eficacia para contraseñas que tienen una longitud pequeña.

Además de Hashcat, también existen más herramientas relevantes como John the Ripper, Hydra o Medusa. Estas herramientas permiten hacer ataques en diferentes contextos, los cuales pueden ser contraseñas locales, redes y archivos protegidos, porque pueden combinar ataques de fuerza bruta, diccionario y máscaras, haciendo un trabajo mucho más versátil.

2.4 Computación de alto rendimiento (HCP)

Un entorno HPC está formado por varios elementos, como los nodos de cálculo interconectados, además del sistema de almacenamiento rápido, las redes de baja latencia y planificadores de tareas que organizan los procesos; estos componentes nos permiten distribuir y ejecutar tareas de forma paralela. Como lo señala Netto et al., "los clústeres pueden manejar problemas computacionales complejos y son gestionados por planificadores por lotes"[31].

2.4.1 Arquitecturas paralelas

Las arquitecturas paralelas constituyen una alternativa eficaz para afrontar tareas que demandan elevados volúmenes de procesamiento, como lo es el caso del descifrado de contraseñas, por lo que este enfoque permite distribuir la carga de trabajo entre múltiples núcleos o dispositivos, lo que mejora notablemente el tiempo de ejecución, tal como lo plantea Ciccozzi et al., "el paralelismo en software, donde múltiples cálculos se ejecutan al mismo tiempo, se ha convertido en la forma más común de proporcionar niveles más altos de poder computacional"[20].

2.4.2 Modelos de programación

- MPI (Message Passing Interface): Es un estándar ampliamente adoptado para arquitecturas con memoria distribuida, por lo cual permite que procesos independientes se comu-

niquen entre sí a través del envío de mensajes. Como lo menciona Skillicorn, “es especialmente útil para sistemas con memoria distribuida, ya que permite una comunicación explícita entre procesos mediante el envío y recepción de datos”[32].

- OpenMP (Open Multi-Processing): Se emplea comúnmente en sistemas con memoria compartida, lo que facilita la paralelización de bloques de código mediante directivas insertadas en lenguajes como C, C++ o Fortran. Skillicorn señala que “las variables pueden declararse como compartidas o privadas y el programador tiene control sobre el modo en que se distribuyen las tareas entre hilos”[32].
- CUDA (Compute Unified Device Architecture): Desarrollado por NVIDIA, nos permite ejecutar tareas masivamente paralelas en GPUs, por lo que es altamente eficiente para tareas como la fuerza bruta, donde se requieren realizar millones de combinaciones por segundo. Como Ciccozzi et al. lo mencionan, “CUDA ofrece un modelo de ejecución masivamente paralelo orientado al procesamiento gráfico, pero adaptable a tareas computacionales generales”[20].
- OpenCL (Open Computing Language): Proporciona portabilidad entre distintos dispositivos como CPU, GPU y FPGA; su modelo de ejecución basado en kernels permite implementar tareas heterogéneas de forma eficiente.

2.4.3 Aplicaciones en seguridad informática

Dentro de la seguridad informática o HPC, se emplea para tareas exigentes como el descifrado de datos, detección de patrones sospechosos en redes o pruebas de fuerza bruta para evaluar contraseñas. Estas actividades requieren millones de operaciones en paralelo, por lo cual se hace imprescindible contar con plataformas potentes. Como lo mencionan Netto et al., “El uso de recursos computacionales bajo demanda proporciona una mejora al obtener resultados a problemas grandes en un tiempo aceptable”[31].

Además, existe la opción del uso de plataformas en la nube como recurso complementario para poder optimizar este tipo de procesos. Según Parra-Gonzáles et al., “la computación en la

nube permite el procesamiento paralelo y distribuido, lo que acelera el tiempo de respuesta y reduce el costo total del procesamiento de datos”[33].

2.5 Métricas de evaluación

2.5.1 Tiempo promedio

Uno de los aspectos más relevantes en pruebas de rendimiento es el tiempo que se tarda en completarse un proceso, especialmente en sistemas paralelos, donde pequeñas demoras pueden amplificarse, y en tareas como el descifrado de contraseñas, contar con una baja latencia y un procesamiento continuo resulta determinante para lograr resultados confiables. Parra-Gonzáles et al. señalan que “la variación en los retardos y el rendimiento inestable de la red tienen un fuerte impacto negativo en las aplicaciones científicas”[33].

2.5.2 Tasa de éxito

La eficacia de un ataque por fuerza bruta puede medirse por la proporción de contraseñas descubiertas con éxito frente al total de intentos realizados, por lo que esta métrica permite evaluar la eficiencia de las herramientas utilizadas o de los algoritmos implementados, además de la vulnerabilidad de dichas contraseñas. Autores como Alkhwaja et al. plantean que “en las pruebas se pudo recuperar con éxito un 32 % de las contraseñas al utilizar fuerza bruta distribuida con OpenMP”[15].

Además de esto, el uso de simulaciones permite verificar qué tan efectivas son estas estrategias de ataque bajo diferentes entornos de ejecución, lo que mejora la comprensión y el comportamiento de dichos algoritmos, conforme a lo expuesto por Saputra et al. “la simulación estima el tiempo requerido para adivinar una contraseña, dependiendo del nivel de complejidad”[27]

2.5.3 escalabilidad

La posibilidad de ampliar el número de procesos sin perder eficiencia representa una ventaja significativa en los entornos de HPC; además, en el caso de algoritmos de fuerza bruta, que se requiere ejecutar múltiples combinaciones en paralelo, la escalabilidad garantiza que el rendimiento se mantenga incluso con un aumento de nodos o cargas de trabajo. Como lo menciona Parra-Gonzáles et al., “las latencias de arranque altas y los anchos de banda limitados limitan severamente la escalabilidad de programas intensivos en comunicación”[33].

2.5.4 Consumo de recursos

En pruebas computacionales intensivas, no solo es importante la velocidad, sino también la cantidad de recursos que se requieren para alcanzar un resultado. El consumo eficiente de memoria y procesadores permite ejecutar tareas complejas sin saturar los sistemas; por ello, evaluar este aspecto permite anticipar los posibles cuellos de botella y optimizar la arquitectura paralela. Como lo explican Parra-Gonzáles et al., “los sistemas de HPC en la nube deben garantizar un acceso rápido y eficiente a la memoria de alta velocidad y proporcionar herramientas para optimizar el uso de la memoria y minimizar la latencia de acceso”[33].

CAPÍTULO III

MATERIALES Y MÉTODOS

3.1 Introducción

A partir del estudio conceptual que se ha realizado en los capítulos anteriores, se da paso al diseño e implementación de un algoritmo propio que permita evaluar la resistencia de contraseñas mediante el uso de fuerza bruta. Para esto, se empleó el entorno de desarrollo Visual Studio Code, donde se elaboró el código base utilizando el lenguaje de programación Python por la versatilidad y compatibilidad con bibliotecas de procesamiento, lo que lo convierte en una herramienta útil para este tipo de tareas.

Con el fin de aprovechar un entorno de ejecución que simule condiciones propias de la computación de alto rendimiento, por esto se decidió trasladar el proyecto a Google Colab, porque esta plataforma ofrece el soporte para aceleración mediante GPU y permite realizar pruebas sin la necesidad de contar con infraestructura física especializada. Esta elección facilitó el acceso a recursos paralelos para validar el comportamiento del algoritmo.

Adicionalmente, se integró la tecnología CUDA, la cual es desarrollada por NVIDIA, esto con el objetivo de optimizar la ejecución del algoritmo a través del procesamiento paralelo, lo que permitió que las combinaciones generadas por el ataque de fuerza bruta se distribuyeran de manera más eficiente en los núcleos de la tarjeta gráfica, lo que redujo el tiempo total de procesamiento y mejoró el rendimiento en comparación con la implementación secuencial tradicional.

En este capítulo se detallan los pasos seguidos para la creación del algoritmo, además de la configuración del entorno de pruebas y las estrategias de paralelización implementadas, así

como también los criterios utilizados para evaluar su efectividad en distintos escenarios.

3.2 Escenarios de pruebas

Para validar el sistema desarrollado, se establecieron varios escenarios de prueba, tomando en cuenta tanto la longitud como la complejidad de las contraseñas, ya que cada variación implica un costo computacional distinto. La dificultad de un ataque por fuerza bruta aumenta de manera exponencial conforme crece el número de caracteres y el conjunto de símbolos, lo que incide directamente con el tiempo requerido y en la carga de procesamiento. Con esto en mente, se incorporaron criterios de generación de contraseñas basados en las recomendaciones del estándar 27001, donde se indica que una clave robusta debe "combinar letras mayúsculas, minúsculas, números y caracteres especiales.ªdemás, se debe evitar "palabras de diccionario, patrones comunes y cualquier información personal predecible"[26], dado que este tipo de elementos facilita su descubrimiento mediante técnicas automatizadas y estas directrices fueron consideradas para construir los rangos de prueba utilizados en los experimentos, asegurando escenarios realistas y acordes con las prácticas actuales de seguridad.

1. Contraseñas cortas

   Se clasificaron como contraseñas cortas a aquellas con una longitud entre 1 y 4 caracteres; la razón es que presentan una complejidad casi nula y un espacio de búsqueda demasiado bajo; en este escenario permite validar el correcto funcionamiento del sistema en tiempos de respuesta que son mínimos.

2. Contraseñas medianas

   Se clasificó como contraseñas medianas a aquellas con una longitud entre 5 y 8 caracteres; en este rango, el número de combinaciones posibles aumenta en una forma considerable, además de que la complejidad es más fuerte, y este escenario resulta útil para observar cómo el rendimiento se ve afectado cuando se aumenta la longitud de las contraseñas a evaluar.

3. Contraseñas largas

   Se clasificaron como contraseñas largas aquellas con una longitud de 9 caracteres o incluso con más caracteres; el crecimiento exponencial del espacio de búsqueda ya genera un nivel de complejidad más alto; de esta manera, el tiempo que tarda en descifrar la contraseña se vuelve más considerable y estas pruebas permiten medir la capacidad del algoritmo frente a escenarios de una dificultad de mayor grado.

4. Datasets

   Además de las pruebas con contraseñas generadas de manera propia, se empleó un dataset de contraseñas comúnmente usadas; específicamente, se utilizó el conjunto de datos RockYou2024.txt [34], en donde se encuentran aproximadamente 180 millones de contraseñas clasificadas entre fuertes y débiles. Con este conjunto de datos se puede evaluar la eficacia del algoritmo bajo un entorno controlado.

3.3 Entorno de desarrollo

El desarrollo del algoritmo de fuerza bruta se llevó a cabo empleando un conjunto de herramientas y plataformas que permitieron disponer de un entorno flexible, accesible y adecuado para la experimentación. En la primera etapa, las pruebas iniciales se realizaron en un equipo de escritorio con Windows 11, esto con un procesador Intel Core i3-9100f de cuatro núcleos y 12 GB de memoria RAM, lo cual permitió implementar la versión secuencial y validar el comportamiento básico del algoritmo en un entorno físico y controlado.

Posteriormente, el proyecto fue migrado a Google Colab con el objetivo de evaluar el rendimiento en un entorno de computación en la nube. Para la etapa de pruebas en CPU se utilizó la instancia estándar proporcionada por Colab, la cual ofrece un procesador de dos núcleos, lo que es suficiente para ejecutar y comparar la versión secuencial y la versión paralelizada del algoritmo.

Finalmente, para simular escenarios de mayor demanda computacional y aprovechar el pa-

ralelismo masivo, se habilitó el entorno de Colab Premium con acceso a la GPU NVIDIA A100, además de la posibilidad de activar la alta capacidad de memoria RAM para manejar combinaciones de gran tamaño y estructuras de datos más complejas.

3.3.1 Selección de herramientas y justificación técnica

Para la codificación del algoritmo se eligió el lenguaje de Python por la simplicidad y la disponibilidad de bibliotecas que facilitan el manejo de datos, además de la ejecución de procesos complejos. Por otro lado, el entorno que se escogió es Visual Studio Code; es una herramienta accesible y multiplataforma que ofrece funcionalidades avanzadas de depuración y de edición, lo cual resultó esencial durante la etapa de desarrollo inicial, además se consideró que el objetivo del proyecto es evaluar el rendimiento del algoritmo bajo un enfoque de computación de alto rendimiento y se seleccionó Google Colab como plataforma de ejecución porque proporciona acceso a GPUs de alto rendimiento y permite realizar pruebas de procesamiento paralelo sin requerir infraestructura física especializada y, por último, la elección de CUDA como tecnología de paralelización se justificó por su compatibilidad con las GPUs disponibles en Google Colab, así como por su efectividad en la aceleración de tareas que requieren gran capacidad de cómputo, como es el caso de fuerza bruta que se usa.

3.3.2 Visual Studio Code como entorno de codificación principal

Para el desarrollo del proyecto se eligió Visual Studio Code como entorno de programación por su compatibilidad con diversos lenguajes y la facilidad que ofrece para integrar extensiones que optimizan el proceso de desarrollo, y esta herramienta resultó ideal para la escritura y depuración junto a la ejecución del algoritmo de fuerza bruta gracias al soporte para Python y la capacidad de personalización en la interfaz de trabajo.

En este entorno se configuraron los archivos principales del proyecto; de esta forma se implementó el algoritmo encargado de generar las combinaciones de contraseñas, además de realizar

las comparaciones correspondientes, y la estructura del código se organizó en bloques y funciones. Esto permitió mantener una lógica clara y modular durante la etapa de desarrollo; cada sección del programa cumplía una función clara y específica, como lo es la definición del conjunto de caracteres a usarse, la generación de combinaciones, la validación de la contraseña y, sobre todo, la medición del tiempo de ejecución que toma descifrar la contraseña a evaluar.

Además, Visual Studio Code facilitó la integración de herramientas externas que permitieron que la ejecución se realice de forma correcta y óptima junto al análisis de rendimiento de distintos entornos, y la consola integrada del editor se usó para visualizar los resultados en tiempo real, mostrando las combinaciones que se generaron. Gracias a esta configuración, el entorno de desarrollo ofreció un espacio práctico y ordenado que permitió controlar todas las etapas de codificación y evaluación del presente proyecto, permitiendo la reproducibilidad de los resultados en posteriores pruebas y adaptaciones del algoritmo.

3.3.3 Configuración del entorno de ejecución en Google Colab

Una vez concluida la etapa de desarrollo local, el proyecto fue migrado a Google Colab utilizando la suscripción premium, con el fin de acceder a recursos de cómputo más potentes y adecuados para la ejecución del algoritmo en paralelo y, como se muestra en la Figura 2, esta modalidad permitió trabajar con la GPU NVIDIA A100, un proceso gráfico de alto rendimiento orientado a cargas intensivas de cómputo, además de habilitar la opción de memoria RAM ampliada, necesaria para manejar conjuntos de combinaciones más extensos y estructuras de datos de mayor tamaño.

Durante la migración fue necesario adaptar parte del código para garantizar su compatibilidad con el entorno de ejecución en la nube con la arquitectura de la A100; esto incluyó ajustes en la inicialización de los kernels, la administración de memoria y la definición de bloques e hilos para aprovechar de manera adecuada el paralelismo masivo de la tarjeta gráfica. Esta configuración permitió llevar a cabo experimentos con cargas de trabajo significativamente más exigentes y analizar cómo el algoritmo respondía bajo un entorno cercano al cómputo de alto

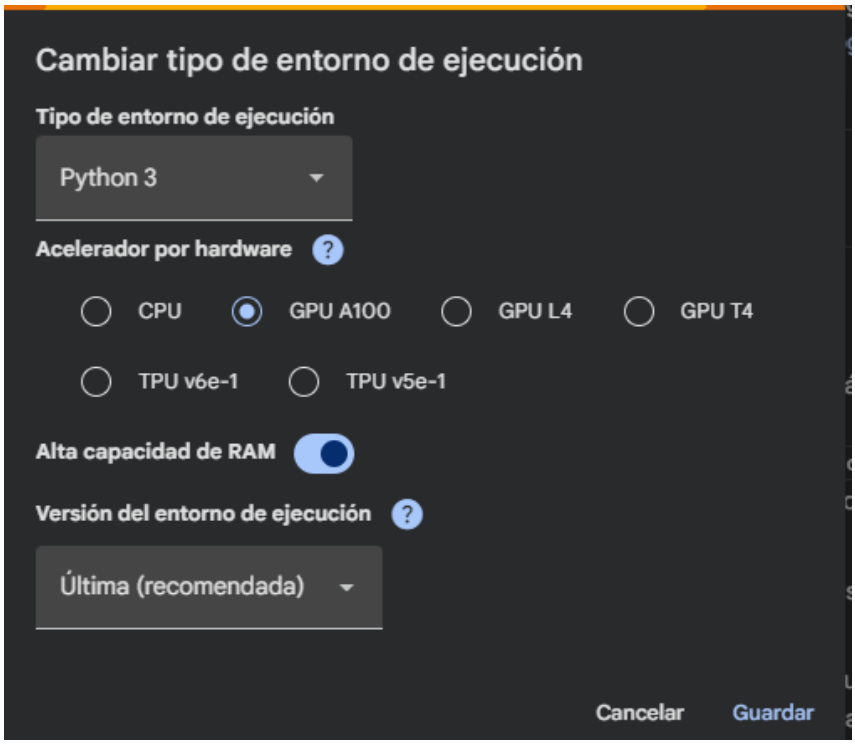

Figura 2.
Configuración del entorno de Google Colab seleccionando GPU como acelerador de hardware

rendimiento.

3.3.4 Integración de CUDA para procesamiento paralelo

Para la optimización del rendimiento del código de fuerza bruta, se integró CUDA (Compute Unified Device Architecture) como plataforma de procesamiento paralelo. Esta tecnología desarrollada por NVIDIA ayuda a que las tareas computacionales con una gran carga de trabajo se distribuyan en los núcleos de las GPUs, lo que es importante en los procesos donde se necesitan millones de combinaciones en un corto período de tiempo. Además, la integración de CUDA en el entorno de Google Colab se realizó de manera transparente, aprovechando las capacidades del hardware disponible. Esto para verificar la correcta activación de CUDA; se utilizó el código !nvcc –version dentro de la celda de código, dándonos el resultado la versión 12.5 de CUDA que está activa en el entorno, como se puede observar en la Figura 3.

Esta configuración garantizó que las operaciones de generación y la validación de combinaciones se ejecutaran de una forma paralela para que se reduzca de manera significativa el tiempo

de procesamiento respecto a la ejecución en CPU. Además, la integración de esta herramienta fue importante para simular un entorno de computación de alto rendimiento y permitió evaluar el comportamiento del algoritmo bajo condiciones reales de carga computacional sin la necesidad de disponer de un clúster físico.

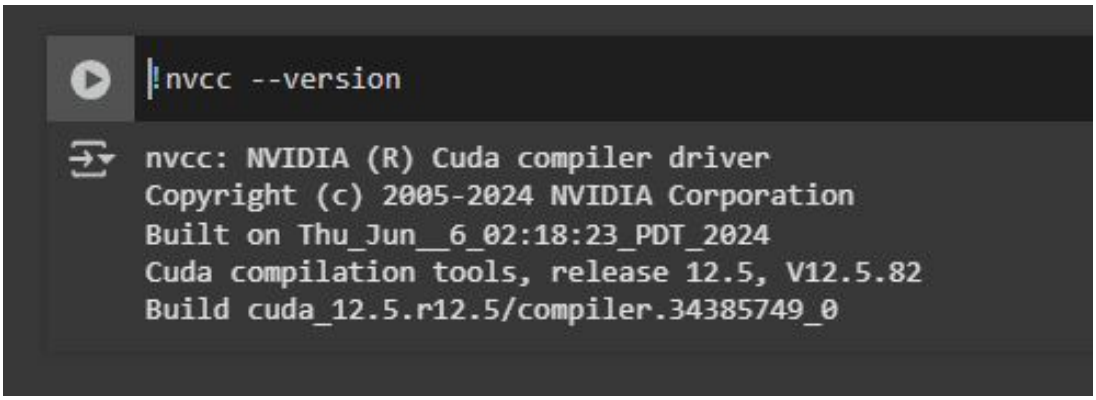

Figura 3.
Verificación de la integracón de CUDA en Google Colab

Además para permitir la ejecución del código en la GPU ya migrado en Google Colab, es necesario integrar PyCUDA, que es una biblioteca que facilita la interacción entre Python y CUDA; esto nos da paso a poder compilar y ejecutar kernels de manera directa desde el entorno de desarrollo mencionado anteriormente.

La instalación se la realizó mediante el gestor de paquetes pip, de manera que se ejecutó el comando mostrado en la Figura 4, el cual descarga y además instala la última versión estable de PyCUDA junto a sus dependencias necesarias, y en esta etapa es importante verificar que la instalación se haya realizado de forma correcta, ya que cualquier incompatibilidad o fallo impedirá que se pueda compilar el código CUDA.

En la Figura 4 se aprecia cómo la instalación se finalizó de una manera correcta, lo cual confirma que el entorno ya está listo para la ejecución de código paralelo sobre la GPU, y este paso es esencial para que en las fases posteriores se puedan cargar y ejecutar los kernels de CUDA de una forma más optimizada para la tarea de fuerza bruta.

Figura 4.
Instalación de PyCUDA en Google Colab y confirmación de instalación exitosa

3.4 Diseño lógico del algoritmo de fuerza bruta

3.4.1 Lógica general del algoritmo y estructura de búsqueda

El algoritmo se desarrolló con el enfoque de fuerza bruta; esto consiste en generar de manera sistemática todas las combinaciones posibles de caracteres hasta que se encuentra aquella que coincida con la contraseña a encontrar. Para poder lograr esto, se diseñó una función llamada fuerza_bruta(), la que se encarga de recorrer de una manera exhaustiva todas las posibles combinaciones que se pueden formar a partir de un conjunto determinado de caracteres, y este conjunto también incluye letras, mayúsculas y números; esto permite cubrir un rango extenso de posibilidades.

La estructura general del algoritmo sigue un flujo lógico en el cual primero se establecen los parámetros iniciales, como el rango de caracteres y la longitud de la contraseña, y el programa genera combinaciones de manera iterativa; así compara cada una de estas contraseñas generadas con la contraseña objetivo. En caso de que coincida, el proceso se detiene y se registra el tiempo total que tomó, y esto permite evaluar la eficiencia del algoritmo frente a contraseñas de diferentes complejidades.

Para optimizar la búsqueda, este algoritmo usa estructuras de control que permiten mantener

un seguimiento ordenado de las combinaciones que se generan, lo que reduce los tiempos del procesamiento y evita las repeticiones que son innecesarias. Además, se incorporó una función de registro que muestra en consola el avance que tiene el proceso, de modo que se pueda observar en tiempo real la evolución del intento de descifrado.

El propósito de este diseño es lograr comprobar el comportamiento del algoritmo de fuerza bruta frente a diferentes escenarios y analizar cómo varía el rendimiento dependiendo de la longitud y complejidad de la contraseña a encontrar. Gracias a este enfoque fue posible establecer una base clara para las pruebas comparativas posteriormente entre las versiones secuenciales, paralelas y en GPU.

3.4.2 Entrada y validación de datos

Para la ejecución del algoritmo fue necesario implementar el mecanismo de entrada, lo cual permite que el usuario pueda definir la contraseña objetivo a evaluar. Esto se realiza de forma directa desde la consola del entorno de desarrollo, donde el programa solicita al usuario que ingrese una contraseña alfanumérica y de preferencia con una combinación que tenga una complejidad media o alta.

Antes de iniciar el proceso de búsqueda, el algoritmo valida que la contraseña ingresada cumpla con ciertos criterios mínimos, como lo son la presencia de letras y números junto a la longitud que se encuentre en un rango permitido, y en caso de que esto no sea respetado, el programa notifica al usuario y solicita un nuevo ingreso. Este paso es fundamental para evitar errores de ejecución y asegurar que las pruebas se realicen bajo condiciones controladas y realistas.

El proceso de validación también verifica que no existan espacios vacíos ni caracteres fuera del rango establecido, ya que esto podría interferir con la generación de combinaciones, y una vez que la contraseña pasa el filtro de validación, se almacena temporalmente en memoria y se utiliza como referencia en la función principal del algoritmo.

Gracias a esta etapa, el flujo del programa logra mantener un control adecuado sobre los datos que se ingresen y esto garantiza una ejecución estable y precisa durante las pruebas de rendimiento; además, ayuda a analizar el comportamiento del algoritmo frente a contraseñas válidas y con distintos niveles de complejidad, simulando escenarios similares a los que se encuentran en sistemas reales.

3.4.3 Consideraciones sobre seguridad y rendimiento

El desarrollo de este algoritmo no solo busca demostrar el funcionamiento de un ataque de fuerza bruta; también busca evidenciar cómo la longitud, además de la complejidad de las contraseñas, influye directamente en el tiempo necesario para que sean decifradas. Las contraseñas cortas y compuestas únicamente por letras y números son particularmente vulnerables porque la cantidad de combinaciones es más reducida y esto facilita el ataque, lo que resulta en que sea descubierta en un tiempo reducido, en esta situación se resalta la importancia de implementar políticas de creación de contraseñas que sean exigentes con el uso de caracteres especiales y que se aumente la longitud; esto ayuda a la dificultad de que un ataque sea exitoso.

Desde el punto de vista del rendimiento, este algoritmo fue diseñado para ejecutarse en un entorno de cómputo de alto rendimiento para aprovechar la capacidad de procesamiento paralelo y para acelerar la generación, además de la verificación de combinaciones. Aunque esta estructura es básica, el algoritmo sigue la lógica secuencial y la implementación modular facilita la migración hacia plataformas que trabajen con GPU, como lo es el caso de Google Colab. Además, se puede usar la integración de CUDA, lo que permite ejecutar múltiples combinaciones de una manera simultánea. Esta adaptación es muy importante para que se puedan reducir los tiempos de ejecución cuando se trabaja con contraseñas de mayor complejidad o cuando se realizan pruebas a una mayor escala.

3.5 Implementación técnica del algoritmo

Para facilitar la reproductibilidad del entorno experimental y garantizar la coherencia entre las diferentes fases del desarrollo, se documentaron las versiones de las herramientas utilizadas tanto en el entorno local como en Google Colab Premium, como se muestra en la tabla 1.

Herramienta	**PC Local**	**Google Colab Premium**
Python	3.11.9	3.12.x
CUDA Toolkit	13.0	12.5.82
PyCUDA	No aplica	2025.1.2
NumPy	No especificado	1.26.x

Tabla 1.
Versiones de las herramientas utilizadas en el entorno local y en Google Colab Premium

3.5.1 Implementación secuencial

Para la primera aproximación del algoritmo de fuerza bruta, se implementó de manera secuencial lo que se ejecuta en un solo núcleo de la CPU y, en este esquema, el proceso consiste en generar de forma exhaustiva todas las posibles combinaciones de caracteres hasta que se encuentre una coincidencia con la contraseña a descifrar.

El alfabeto utilizado incluye letras mayúsculas, minúsculas y dígitos, lo que incrementa el espacio de búsqueda en función de la longitud de la contraseña, y la complejidad del algoritmo crece de manera exponencial, ya que el número de combinaciones posibles se calcula con la ecuación 1:

$$C = N^L \tag{1}$$

Donde:

- N = Tamaño del alfabeto (son los caracteres posibles)
- L = Longitud de la contraseña
- C = Número total de combinaciones a evaluar

El procedimiento secuencial se muestra en la figura 5 e inicia con la solicitud de una contraseña al usuario y posteriormente genera las combinaciones necesarias en orden usando la librería itertools; además, en cada intento es comparado con la contraseña objetivo y, en caso de coincidir, el programa finaliza mostrando el número de iteraciones realizadas y el tiempo que transcurrió.

Este enfoque es correcto para contraseñas cortas; sin embargo, debido a la naturaleza exponencial del problema, se vuelve ineficiente para contraseñas medianas y largas, y esto justifica la necesidad de explorar enfoques paralelos en CPU y GPU.

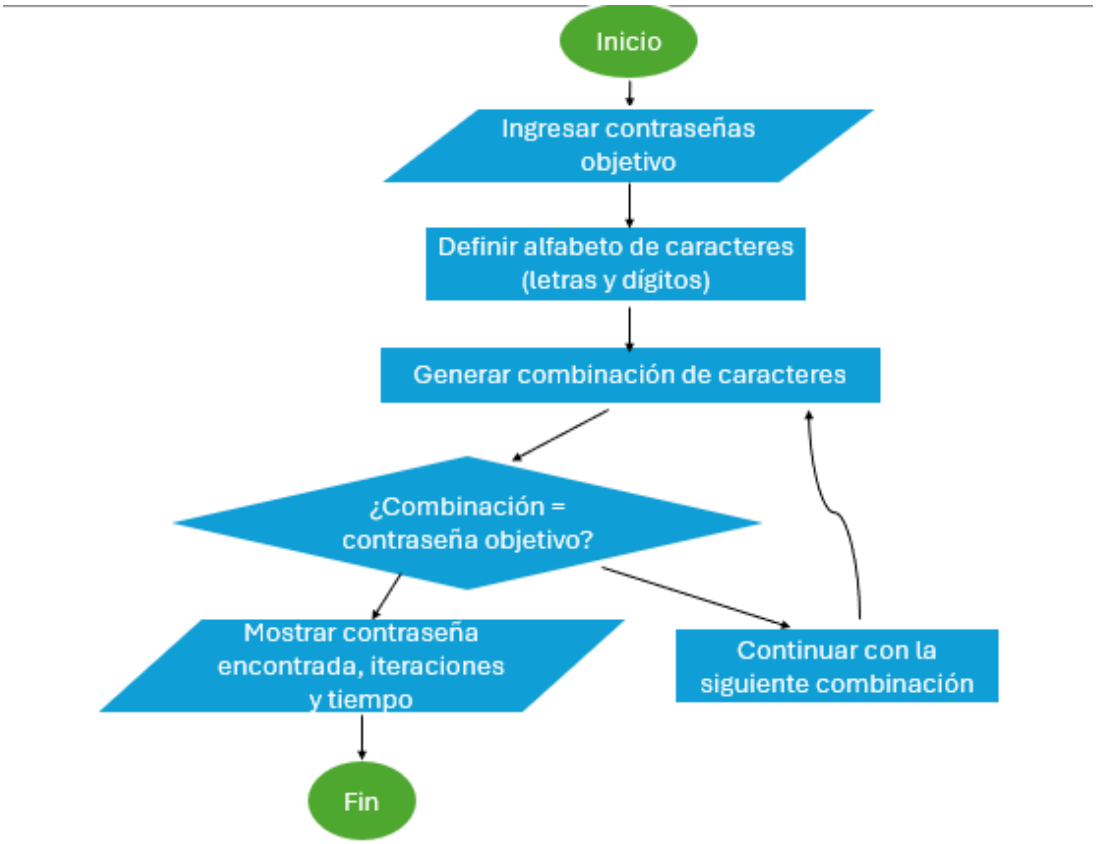

Figura 5.
Flujograma de la implementación secuencial

3.5.2 Implementación paralela multicore

El algoritmo en su versión multicore se ejecutó asignando el número de procesos de acuerdo con la cantidad de núcleos disponibles en cada entorno de prueba. En el equipo de escritorio, que dispone de un procesador de cuatro núcleos, la ejecución paralela se configuró mediante el parámetro de n_procesos = 4, lo que permitió lanzar cuatro procesos simultáneos para distribuir el espacio de búsqueda. Cada proceso opera sobre un subconjunto distinto de combinaciones utilizando un mecanismo de turnos definido por los parámetros start y step, donde step corresponde al número total de procesos. De esta manera, cada proceso evalúa únicamente una de cada cuatro combinaciones generadas, evitando soplamientos y equilibrando la carga entre los cuatro núcleos del procesador físico.

En Google Colab, la instancia utilizada proporciona una CPU virtual donde se tienen dos núcleos disponibles, por lo que la versión paralelizada del algoritmo se ajustó para ejecutarse con n_procesos = 2. El algoritmo mantiene el mismo esquema de distribución basado en turnos, de modo que cada proceso explora la mitad del espacio total de combinaciones. Esta adaptación permite comparar el rendimiento entre ambos entornos bajo las limitaciones de su respectivo hardware, garantizando que el reparto del trabajo siempre refleja la cantidad real de núcleos disponibles.

Este enfoque permite aprovechar arquitecturas multicore modernas donde es posible ejecutar múltiples hilos de manera concurrente; sin embargo, la escalabilidad está limitada por el número de núcleos que estén disponibles en la CPU y por la sobrecarga de comunicación entre procesos.

Como se muestra en la Figura 6, el flujograma ilustra cómo el espacio de cada combinación es dividido en rangos y es asignado a múltiples procesos porque cada proceso genera sus combinaciones y, en caso de que se encuentre la contraseña, se actualiza la bandera global para que se detenga la ejecución del resto.

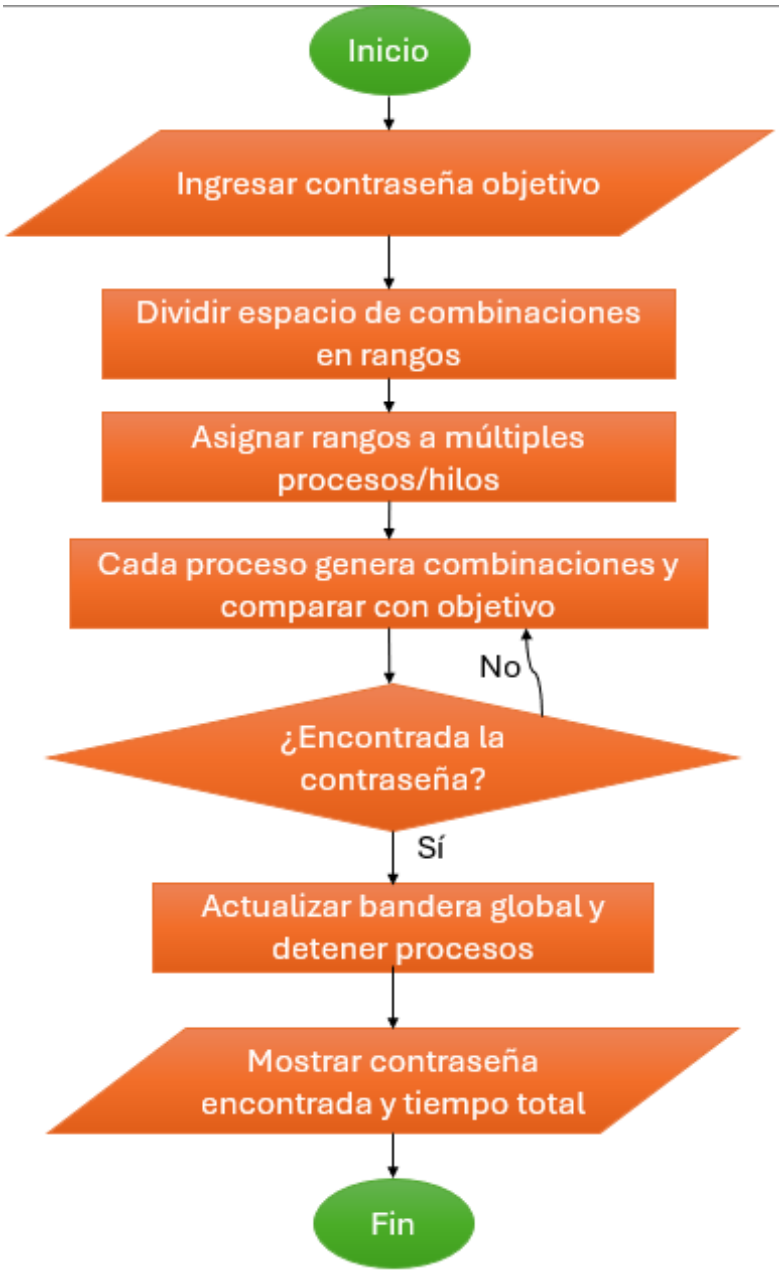

Figura 6.
Flujograma de paralelización multicore

3.5.3 Implementación en GPU (CUDA)

La versión del algoritmo diseñada para GPU aprovecha la arquitectura masivamente paralela de CUDA, donde miles de hilos pueden ejecutar operaciones independientes de manera simultánea. Esta implementación hace que cada hilo genere y evalúe un intento distinto dentro del espacio de búsqueda, lo que permite procesar grandes volúmenes de combinaciones en tiempos

significativamente menores en comparación con la CPU.

El kernel utilizado se estructura alrededor de tres componentes clave: el uso de memoria optimizada, la distribución del trabajo entre bloques e hilos y la sincronización mediante operaciones atómicas.

- Uso de memoria constante y compartida:

  El conjunto de 62 caracteres utilizado para generar las combinaciones se almacenan en memoria constante, lo cual reduce la latencia de acceso al ser un arreglo consultado de manera uniforme por todos los hilos. de la esta forma la contraseña objetivo se copia en memoria compartida dentro de cada bloque, permitiendo que la comparación se realice desde un espacio de memoria de baja latencia. Esta estructura minimiza dependencias con la memoria global y mejora el tiempo de respuesta del kernel.

- Configuración de hilos por bloque y número de bloques:

  El kernel utiliza los parámetros estándar de CUDA blockDim.x y gridDim.x para distribuir el trabajo. En el entorno configurado, la ejecución se optmizo utilizando 256 hilos por bloque, un valor que se ajusta bien al tamaño típico de warp y al balance entre registros disponibles y la latencia de memoria. El número de bloques se calcula dinámicamente en función del tamaño del batch de combinaciones que debe procesarse lo cual ppermite saturar los multiprocesadores disponibles en la GPU y con esto, la GPU puede asignar múltiples bloques en paralelo, manteniendo siempre hilos activos mientras existan combinaciones por evaluar.

- Distribución del espacio de búsqueda (overlap control):

  Cada hilo obtiene un identificador global (gid), calculado a partir del bloque y del índice del hilo dentro del bloque. A partir de este identificador se determina el primer intento a procesar y un stride correspondiente al número total de hilos activos en la cuadrícula. Este esquema garantiza que cada hilo procese índices que no se superponen con los de los otros hilos lo que evita soplamientos es decir overlaps en la búsqueda. De esta manera,

toda la cuadrícula avanza sobre el espacio de combinaciones de forma equilibrada, con saltos regulares y sin duplicación de trabajo entre hilos.

- Generación de intentos:

  Para cada índice asignado, el hilo construye la cadena candidata utilizando la conversión aritmética en base 62. Se realiza luego la comparación carácter por carácter con la contraseña objetivo almacenada en memoria compartida. Si se detecta una coincidencia el hilo actualiza una bandera global mediante atomicExch, lo cual detiene a otros hilos en iteraciones posteriores y esto reduce el gasto innecesario de cómputo.

- Sincronización y acumulación de métricas:

  El kernel utiliza operaciones warp-level (__shfl_down_sync) esto para acumular el número de intentos procesadors y, en el caso de coincidencias, el numero de aciertos. La actualización de estas métricas se realiza mediante operaciones atómicas en memoria global, garantizando la coherencia de los valores sin generar bloqueos prolongados entre los hilos.

La elección de utilizar 256 hilos por bloque responde a un criterio de equilibrio entre el paralelismo disponible y el uso eficiente de los recursos internos de la GPU. Este valor coincide con un múltiplo completo del tamaño de un warp (32 hilos) y esto evita la creación de warps parciales y reduce la ociosidad dentro de cada bloque. En conjunto, esta configuración de 256 hilos por bloque, múltiples bloques distribuidos según el tamaño del batch y un esquema de stride sin superposición permite que la GPU ejcute millones de combinaciones por segundo, aprovechando de forma eficiente la arquitectura multiprocesadores de la tarjeta utilizada.

Como se observa en la Figura 7 el flujograma describe el proceso de ejecución del algoritmo en GPU cuando la contraseña objetivo es copiada a la memoria compartida después cada hilo genera un intento diferente y lo compara con la contraseña a descifrar y en caso de que coincida se actualizará la bandera global mediante una operación atómica y se guarda el resultado.

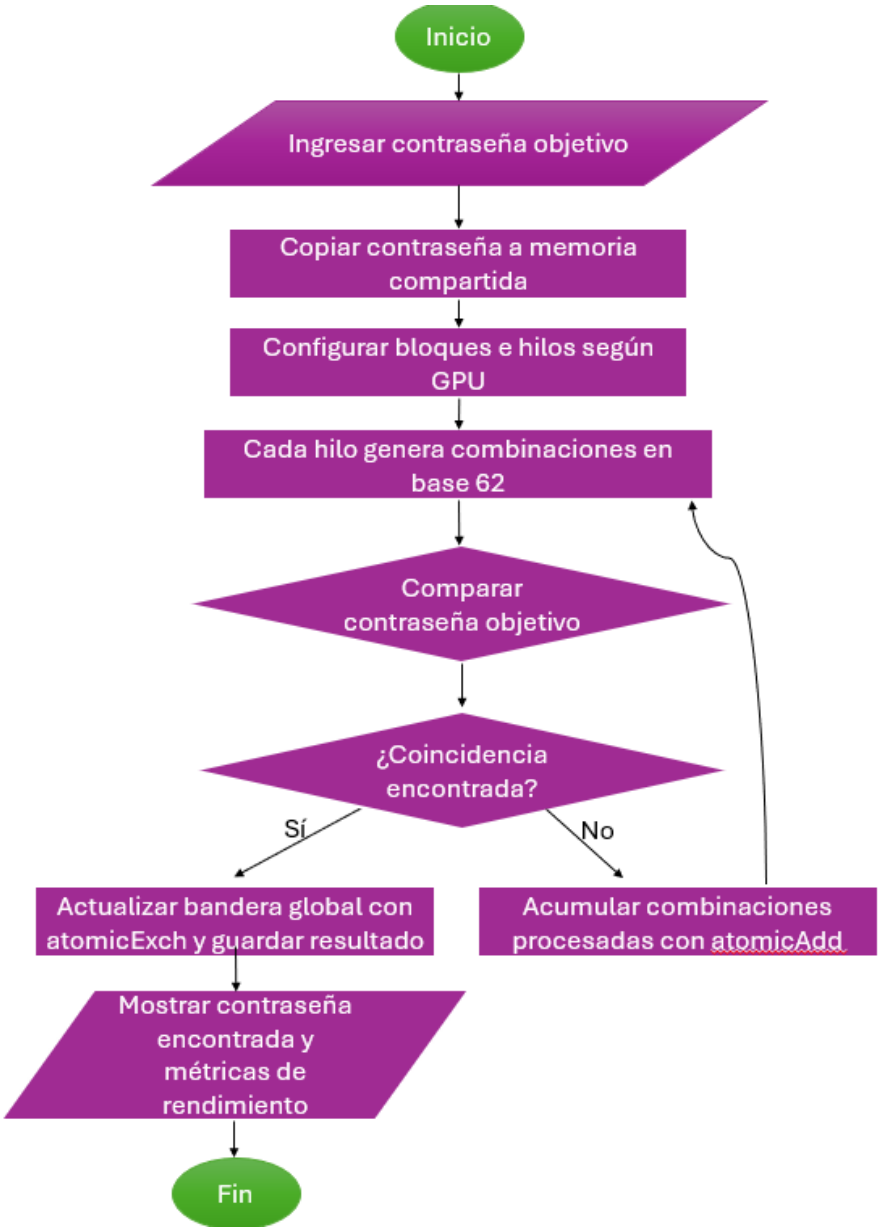

Figura 7.
Flujograma de la implementación en la GPU

3.5.4 Codificación en Python desde Visual Studio Code

En el desarrollo inicial del algoritmo de fuerza bruta, se realizó usando la herramienta de Visual Studio Code, un entorno de programación por la capacidad de ofrecer un espacio de trabajo flexible además de organizado. Esto permitió que se pueda gestionar de manera eficiente la escritura y la depuración del código, y durante esta fase también se diseñó la estructura básica del algoritmo que se va a usar, esto en lenguaje de Python y se enfocó en crear una función que sea

capaz de generar todas las combinaciones posibles de caracteres alfanuméricos y compararlas de forma secuencial hasta encontrar la contraseña que es correcta.

Dentro del entorno, cuando el algoritmo se ejecuta de una forma directa, la herramienta permite visualizar los resultados en tiempo real y, durante las pruebas, se mostraban en pantalla el proceso de generación de combinaciones, la comparación con la contraseña a evaluar, además del tiempo total que tardó la ejecución una vez que se encontraba dicha contraseña, y este proceso permitió comprobar el correcto funcionamiento del algoritmo antes de aplicar las modificaciones necesarias para su ejecución paralela.

El flujo general del programa inicia con la definición de los parámetros principales, como lo son el conjunto de caracteres y la contraseña a evaluar. Después de esto, se realiza la iteración exhaustiva de combinaciones mediante el uso de funciones, las cuales controlan el número de intentos junto a la correspondiente validación. Finalmente, una vez encontrada la contraseña correcta, el sistema muestra un mensaje, el cual confirma la contraseña y el tiempo total de la búsqueda.

El uso de esta estructura permitió que sea posible validar la lógica del algoritmo en un entorno controlado, además de garantizar que los resultados fueran reproducibles y sirvieran de base para las versiones posteriores orientadas al procesamiento paralelo y la ejecución en GPU.

3.5.5 Adaptación del código para ejecución paralela

En el código inicial implementado en Python, se realizaba el proceso de fuerza bruta de una forma secuencial donde se empleaba la librería itertools para que se generen todas las combinaciones posibles de caracteres. Este enfoque es funcional, pero tiene una limitación importante cuando se refiere rendimiento, ya que, al ser secuencial, se usa la CPU y esta procesa las combinaciones una por una.

Para optimizar el tiempo de ejecución, se migró el algoritmo a PyCUDA y esto permitió aprovechar la capacidad de procesamiento masivo que ofrece Google Colab con la GPU, y esta

adaptación implicó que se realizaran cambios estructurales en la lógica del programa base, que son los siguientes:

- Se realizó la sustitución de la generación secuencial de combinaciones por un esquema que se basa en índices en base de 62, lo que permite que cada hilo de la GPU pueda calcular directamente la combinación que le corresponde sin depender de estructuras iterativas pesadas.
- Definición de constantes y variables en la memoria de la GPU para poder almacenar el conjunto de caracteres, la contraseña objetivo y el estado en el cual se encuentra el proceso.
- Se añadió el módulo de CUDA (kernel) compilado en tiempo de ejecución, esto usando PyCUDA, el cual es el encargado de distribuir el trabajo entre los miles de hilos y bloques en los que se hace la ejecución.
- Se optimizó el uso de memoria con la implementación de memoria compartida (shared memory); esto para almacenar la contraseña objetivo en cada bloque, lo que reduce el acceso a memoria global y así se puede mejorar el rendimiento del algoritmo.

En cuanto al flujo general después de la adaptación consiste en lo siguiente:

- Se preparan los datos para después transferirlos a la memoria de la GPU utilizada.
- Se calcula el número óptimo de bloques e hilos por bloque según las características del dispositivo a usarse, es decir, la capacidad de multiprocesadores, el máximo de hilos por bloque, etc.
- Se lanza el kernel, el cual distribuye las combinaciones entre los hilos que se encuentran activos.
- Se recuperan los resultados y las métricas de rendimiento desde la GPU al entorno utilizado.

Esta modificación no solo permitió reducir drásticamente el tiempo de la ejecución; también ayuda a escalar el algoritmo para probar contraseñas de mayor longitud, lo que es algo inviable en el enfoque original debido a las limitaciones que tiene la CPU.

3.5.6 Uso de kernels Cuda para distrubución de tareas

Una vez que se adaptó el algoritmo para que se aproveche el procesamiento paralelo de la GPU, esto fue necesario para implementar el kernel CUDA, que es el encargado de distribuir todas las combinaciones posibles de contraseñas entre los diferentes hilos de ejecución, y este kernel fue desarrollado en el lenguaje de C para CUDA y es compilado de forma dinámica desde Python mediante la librería usada PyCUDA.

Para esto debemos hacer énfasis en las siguientes características:

- La definición de memoria constante (__constant__), se usa para el conjunto de caracteres (charset) esto con la finalidad de acelerar el acceso repetitivo a este dato por parte de todos los hilos usados.
- La función idx_to_str_ull se usó para convertir un índice numérico en una cadena de texto en base 62, lo que optimiza la generación de combinaciones sin tener la necesidad de usar estructuras iterativas con un coste computacional elevado.
- El uso de la memoria compartida (__shared__) se empleó para almacenar la contraseña objetivo dentro de cada bloque usado y esto reduce la latencia de acceso con respecto a la memoria global.
- El cálculo del identificador global del hilo (grid) y el stride permite que cada hilo pueda procesar múltiples combinaciones de una forma ordenada y que lo haga sin solapamientos.
- La comparación directa en cada intento con la contraseña a encontrarse utilizó operaciones atómicas (atomicExch, atomicAdd); de esta manera se controló la sincronización, además del registro de resultados sin condiciones de carrera.

La lógica implementada garantizó que, una vez encontrada la contraseña, todos los hilos detengan la ejecución lo antes posible y así se optimiza el tiempo total del proceso. Adicional a esto, se incluye un contador de combinaciones procesadas para poder calcular las métricas necesarias de rendimiento, como lo son la velocidad por intentos por segundo.

3.5.7 Interaccón entre CPU y GPU durante el proceso

Antes de que el procesamiento paralelo pueda comenzar, la CPU desempeña un papel importante, ya que prepara el entorno de trabajo de la GPU, y en esta fase inicial el programa realiza la verificación básica de la contraseña que se ingresó para asegurarse de que se cumplan las restricciones de longitud establecidas. También se define el conjunto de caracteres permitidos y se almacena en la memoria constante de la GPU, lo que permite que todos los hilos puedan acceder a la información de una manera rápida y de manera uniforme durante la ejecución.

Durante la ejecución del algoritmo, la interacción entre CPU y GPU desempeña un papel esencial para que se pueda garantizar el correcto flujo de información, ya que, una vez que la CPU prepare los datos necesarios, estos son transferidos hacia la memoria de la GPU, donde cada hilo del kernel accede a ellos para que se pueda iniciar el procesamiento paralelo.

Dicho proceso de transferencia debe realizarse de una manera controlada, ya que la comunicación entre ambos componentes depende de la correcta gestión de la memoria y, en esta etapa, la CPU actúa como un coordinador, lo que asegura que los datos lleguen al dispositivo de forma ordenada y que no existan pérdidas. Una vez completada la ejecución en la GPU, los resultados son devueltos al procesador principal para que se analicen y pueda mostrar la información.

Para finalizar, la CPU ajusta la configuración de la ejecución de la GPU, lo que determina cuántos hilos y cuántos bloques se lanzarán. Este ajuste se hace en función de la arquitectura disponible; así se evita que hilos queden inactivos y se asegura que la capacidad de cómputo de la GPU se use de la manera más eficiente posible.

3.6 Paralelización con Cuda en entorno HPC

La ejecución del algoritmo en un entorno de computación de alto rendimiento se realizó empleando la GPU NVIDIA A100 disponible en la versión Premium de Google Colab. Esta tarjeta esta diseñada para cargas de trabajo intensivas, lo que permitió aprovechar un nivel de paralelismo considerablemente mayor al alcanzado en equipos de escritorio o en instancias de GPU estándar. Antes de iniciar el proceso, se usó una configuración basada en 256 hilos por bloque, un tamaño que permite mantener una alta ocupación sin comprometer la disponibilidad de registros ni limitar la ejecución concurrente de otros bloques. A partir de este valor, el programa calcula también el número de bloques en total, determinado por el tamaño del lote de combinaciones asignadas al kernel. Esta estrategia asegura que el grid size sea lo suficientemente grande como para mantener todos los multiprocesadores de la A100 trabajando de forma continua.

La distribución del espacio de búsqueda se organiza para evitar soplamientos entre los hilos. Cada bloque recibe un rango distinto del total de combinaciones y, dentro de ese rango, cada hilo avanza de forma ordenada sobre los índices que le corresponden. Esto garantiza que ningún hilo repita el trabajo asignado a otro y que la GPU avance de forma uniforme sobre el conjunto de posibilidades. Gracias a esta partición explícita, el algoritmo mantiene una carga de trabajo bien distribuida incluso cuando el volumen de combinaciones es muy elevado.

La información mostrada durante la inicialización incluye el modelo de la GPU, la cantidad de bloques generados y la longitud de la contraseña objetivo, así como el total de combinaciones a evaluar en el lote actual. Estos parámetros son esenciales en entornos de HPC, porque permiten ajustar la configuración de ejecución según la capacidad real del hardware y evitar tanto la subutilización de recursos como la sobresaturación de hilos. Al automatizar esta configuración, el sistema es capaz de aprovechar de manera eficiente el potencial de cómputo de la A100 y mantener un nivel de paralelismo sostenido durante todo el proceso.

3.6.1 Principios básicos del modelo de ejecución CUDA

El modelo de ejecución que usa CUDA se basa en una arquitectura jerárquica que organiza entre hilos, bloques y grillas (grids en inglés). Esto permite la distribución de grandes volúmenes de trabajo de una manera paralela en la GPU usando kernels. De acuerdo con NVIDIA [35], "un kernel es ejecutado por una malla de bloques de hilos. Cada hilo ejecuta el kernel sobre diferentes datos".

En palabras de NVIDIA [35], "cada hilo tiene memoria local provada. Cada bloque de hilos tiene memoria compartida visible para todos los hilos del bloque y con la misma duración que el bloque. Finalmente, todos los hilos tienen acceso a la misma memoria global", esta estructura facilita la asignación de tareas a diferentes unidades de cómputo, lo que aprovecha la arquitectura paralela de las GPUs modernas.

En la Figura 8 se presenta un esquema donde se ilustra el modelo de la ejecución que realiza CUDA; muestra cómo los hilos se agrupan en bloques y estos a su vez en grids, además de la relación con los diferentes niveles de memoria con los que se trabaja, y este diagrama es importante para comprender cómo se distribuye la carga de trabajo en una GPU que usa el modelo de CUDA.

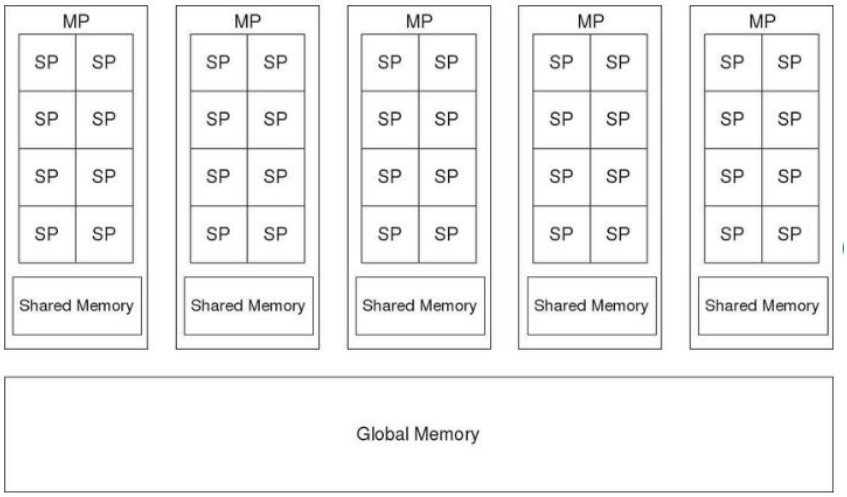

Figura 8.
Modelo de ejecución de CUDA con la jerarquía de hilos, bloques y grids. Adaptado de *GPGPU Kernel Implementation using an Embedded Language: a Status Report [36]

3.6.2 Organización de hilos y bloques en GPU

En cuanto a la disposición de hilos y los bloques, constituyen un elemento que es importante para maximizar el rendimiento de la programación que se ha empleado con CUDA. Tal como lo señala NVIDIA [35], “los hilos se agrupan en bloques, y estos bloques forman una cuadrícula; cada hilo ejecuta el mismo código, pero trabaja con un conjunto diferente de datos”. Esta estructura hace posible que el hardware distribuya el trabajo de una forma masiva, además de una forma paralela, lo que permite aprovechar al máximo la capacidad de cómputo que se tiene disponible.

Para optimizar la ejecución del kernel, además de garantizar el uso adecuado de los recursos de la GPU, se implementó una función que se encarga de determinar de forma automática la cantidad ideal de hilos y bloques según las características del dispositivo. Esta función es capaz de analizar la capacidad de cómputo disponible junto al número máximo de hilos por bloque y la cantidad de multiprocesadores activos y lo ajusta de una forma dinámica antes de lanzar el kernel. De esta manera, el algoritmo se puede adaptar a diferentes entornos de hardware que sean compatibles con esta dinámica, además de que no necesita una intervención manual y la detección automática de parámetros reduce los errores asociados a configuraciones estáticas y permite que la carga de trabajo sea distribuida de una forma equilibrada.

Una vez que se determinaron los valores iniciales, el código ejecuta el ajuste dinámico o, en inglés, auto-tune, y se verifica si el número total de combinaciones que se van a procesar es menor que la capacidad máxima de hilos posibles. Esto se realiza con blocks * threads_per_block. En caso de que esto sea afirmativo, se procede a reducir el número de bloques para evitar la creación de hilos que están inactivos; así se puede optimizar la utilización de los recursos y se garantiza que cada hilo ejecutado tenga un trabajo real que tiene que procesar, tal como explica NVIDIA [35] “una configuración de ejecución cuidadosamente diseñada puede marcar la diferencia entre un uso eficiente o ineficiente de la GPU”.

Para terminar, cuando se tienen todos los parámetros correctamente ajustados, se procede a

lanzar el kernel fuerza_bruta_opt; así se especifica el tamaño del bloque y el número de bloques, además de la memoria compartida que es necesaria para la ejecución, y este paso es importante porque se determina la forma en que la carga de trabajo se distribuirá entre los procesadores de la GPU.

3.6.3 Ventajas del procesamiento paralelo sobre CPU tradicional

El procesamiento paralelo que se usa mediante la GPU ofrece una serie de beneficios cuando se compara con la ejecución simple de la CPU, especialmente en tareas que necesitan la manipulación intensiva de datos y cálculos repetitivos, como lo plantea NVIDIA [35], "el modelo de programación CUDA permite distribuir la carga de trabajo entre miles de hilos concurrentes, lo que resulta en un incremento significativo del rendimiento en comparación con arquitecturas secuenciales". Esta capacidad se debe a que las GPUs se diseñaron con un gran número de núcleos simples y además son eficientes, están optimizadas para operaciones de punto flotante y procesamiento masivo de datos paralelo.

Una de las principales ventajas que se logra obtener es la capacidad de escalado; así como lo señala NVIDIA [35], "la arquitectura de las GPU es altamente escalable, permitiendo que el mismo código pueda ejecutarse en dispositivos con un distinto número de núcleos sin modificaciones sustanciales". Esto no solo simplifica la migración del código, sino que también garantiza que el rendimiento se pueda aprovechar con las mejoras del hardware a futuro.

En términos energéticos, el procesamiento paralelo también presenta beneficios importantes. De acuerdo con Smith et al. [37], "para cargas de trabajo que involucran procesamiento de datos a gran escala, la GPU logra completar las tareas con un menor consumo energético por operación, en comparación con una CPU, debido a su especialización en cómputo vectorial", y esto implica que, además de reducir los tiempos de ejecución, se puede optimizar el uso de energía, lo que resulta clave en entornos de alto rendimiento.

Otra ventaja que también es importante resaltar es la reducción de la latencia en operaciones

repetitivas. Como se mencionó anteriormente, NVIDIA explica que [35], “la proximidad de los hilos a la memoria compartida dentro de cada bloque permite tiempos de acceso considerablemente menores que en el modelo de memoria principal de la CPU”; esto es crítico en algoritmos como lo son en el caso de fuerza bruta; aquí el acceso rápido a datos recurrentes puede marcar una gran diferencia entre el tiempo de cómputo que es aceptable y uno que no es viable.

La combinación de mayor rendimiento, mejor eficiencia energética, escalabilidad, además del acceso optimizado a la memoria, posiciona a las GPUs como una herramienta ideal para abordar los problemas donde se requiere un alto grado de paralelización, y esto hace que supere a los aspectos de las CPUs tradicionales.

3.6.4 Comparación técnica entre PyCUDA y Numba

La elección entre PyCUDA y Numba depende del nivel de control requerido sobre la GPU y del balance entre productividad y acceso directo a las primitivas de CUDA. PyCuda se caracteriza por ofrecer un enfoque de bajo nivel que expone de forma directa el modelo de programación de CUDA. Como lo señala Akulapally, PyCUDA “permite integrar código CUDA con scripts de Python y automatiza la administración de memoria y la compilación de código directamente desde Python”[38]. Esta integración le permite al desarrollador definir manualmente los parámetros de ejecución, gestionar la memoria y optimizar kernels sin las restricciones propias de un compilador de alto nivel.

Adicionalmente, Akulapally indica que PyCUDA “brinda una interfaz directa para usar kernels escritos en CUDA C junto con la potencia y flexibilidad del entorno de programación Python”[38]. Esto confirma que PyCUDA está orientado a escenarios donde se requiere un control fino sobre la arquitectura CUDA, ya que los kernels se describen íntegramente en C/CUDA y se compilan dinámicamente, ofreciendo acceso directo a hilos, bloques, memoria compartida y optimizaciones específicas del hardware.

Por otro lado, Numba adopta un enfoque de abstracción más alto. Rao et al. señala que Num-

ba “permite que el código Python se ejecute en hardware GPU mediante el decorador @cuda.jit, compilando funciones Python en kernels CUDA sin necesidad de escribir código CUDA explícito”[39]. Esta característica hace que Numba sea más accesible para usuarios que no están familiarizados con el lenguaje CUDA C, ya que las funciones se escriben en Python y es el compilador de Numba el que se encarga de traducirlas a instrucciones para GPU.

Sin embargo, esa simplicidad viene acompañada de ciertas limitaciones. Rao et al. señala que Numba es útil “para comenzar a trabajar con GPUs gracias a la simplicidad de su sintaxis”[39], pero también implica que el desarrollador tiene menos control sobre aspectos como gestión avanzada de memoria, sincronización entre hilos o uso de optimizaciones específicas del hardware. Por tanto, aunque permite desarrollar kernels de manera rápida, Numba está mejor orientado a aplicaciones donde la prioridad sea la productividad y no la optimización extrema.

3.6.5 Desafíos técnicos durante la implementación Google Colab

Durante la fase inicial de desarrollo de la migración del código a Google Colab, se empleó Numba como la herramienta principal para aprovechar la aceleración de la GPU, pero al momento de la ejecución en el entorno surgió una incompatibilidad crítica entre la versión de CUDA Toolkit instalada y la versión que es soportada por Numba, como se muestra en la Figura 9 indica el mensaje de error que arroja el sistema en donde se observa la excepción CudaAPIError: Call to cuLinkAddData results in CUDA_ERROR_UNSUPPORTED_PTX_VERSION. Esto es causado por la discrepancia entre la versión PTX generada, que para esta ocasión es la 8.5, y la versión que soporta el entorno de Colab, que es la 8.4.

Esta limitación impedía la correcta compilación, además de que no se podía realizar la ejecución de los kernels. Por esta razón, se tuvo que replantear la herramienta que se utilizara. Por este contexto, se optó por migrar el código a PyCUDA porque ofrece un control más directo sobre la API CUDA y no depende de un compilador intermedio como Numba, y de esta manera se evitan los problemas de compatibilidad en la generación del código para la GPU.

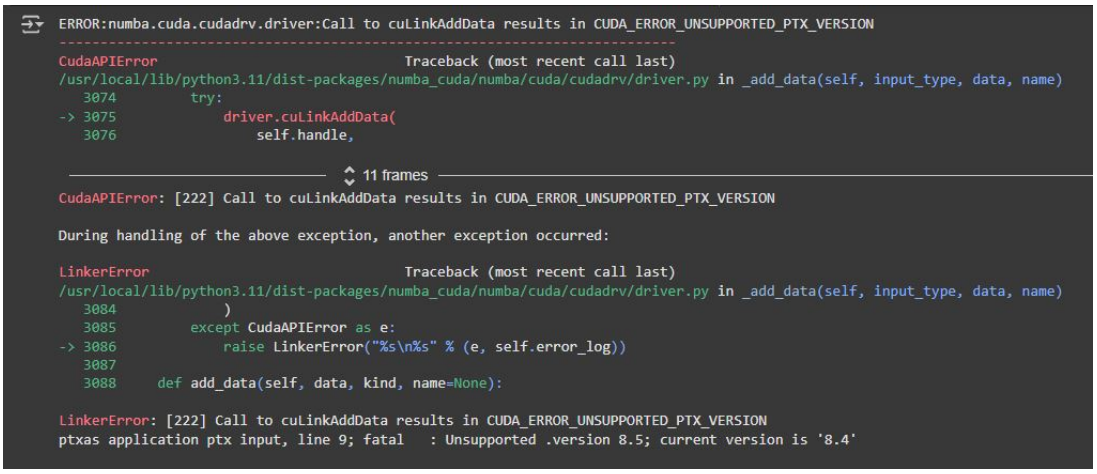

Figura 9.
Error de incompatibilidad de versión PTX al ejecutar código con Numba en Google Colab

Como lo explican Akulapally et al. [38] en GPU Exploration using PyCUDA, "PyCUDA permite a los programadores ejecutar código CUDA desde Python y ofrece un control explícito sobre la gestión de la memoria del dispositivo y el lanzamiento de kernels". Esta flexibilidad fue un paso importante para superar las limitaciones encontradas y así se pudo garantizar que el código pueda ejecutarse de una manera óptima en la infraestructura de Google Colab.

La adopción de PyCUDA no solo resolvió el problema que se tenía; además, facilitó la optimización del código para poder aprovechar al máximo las capacidades de la GPU y permitió un control más detallado sobre la configuración de hilos y los bloques que se van a usar, así como la gestión de la memoria global, compartida y los registros.

3.6.6 Desafíos técnicos durante la implementación GPU local

Durante la implementación del proyecto se presentaron varios desafíos al momento de intentar ejecutar CUDA en la GPU local. El objetivo que se buscaba era comparar el rendimiento entre un entorno local con las características de una NVIDIA GTX 1060 de 3 GB y el entorno en la nube que es Google Colab con la GPU A100. La ejecución local presentó varios fallos que impidieron hacer pruebas locales, lo que reveló limitaciones de compatibilidad entre hardware, sistema operativo, además de librerías de uso paralelo.

Aunque la tarjeta gráfica fue reconocida correctamente por el sistema operativo local con Compute Capability, el controlador 581.15 y CUDA Toolkit 13.0, se presentaron errores al momento de utilizar PyCuda, CuPy y Numba. En el caso de PyCUDA, la inicialización falló por conflictos entre el compilador de Visual C++ y el runtime de la herramienta de CUDA.

En CuPy, los módulos precompilados no eran compatibles con la versión de CUDA 13.0, lo que generó fallos por librerías ausentes, y Numba logró detectar el dispositivo, pero no logró habilitarlo. Esto se debe al soporte limitado de CUDA en arquitecturas 6.1 de Windows.

Tras múltiples intentos, se llegó a la conclusión de que los problemas estaban relacionados con la antigüedad del hardware, además de las incompatibilidades de las librerías con la herramienta de CUDA en su versión 13.0 y las restricciones que tiene Windows para la gestión de dependencias dinámicas.

Por esta razón, la ejecución en la GPU se realizó solo en el entorno de Google Colab utilizando una NVIDIA A100, la cual ofreció un entorno estable y totalmente compatible con las herramientas que se decidieron usar. Aunque la limitación de la GPU local restringió las comparaciones, ayudó a comprender con mayor profundidad la importancia de mantener las versiones de hardware, librerías y sistemas operativos actualizados para proyectos de computación de alto rendimiento, es decir, HPC.

3.7 Métricas de evaluación del rendimiento

En el contexto de procesamiento paralelo con GPU, la evaluación del rendimiento es algo importante para poder determinar la eficiencia y la escalabilidad de una implementación. Para este proceso se requiere medir parámetros clave como lo son el tiempo de ejecución, la tasa de procesamiento y el uso de recursos de hardware disponibles. Como lo señala Akulapally [38] “PyCUDA no solo permite acceder de manera directa a las capacidades de la GPU, sino también instrumentar el código para medir y optimizar el rendimiento”.

En aplicaciones donde la carga de cómputo es intensa, las métricas de rendimiento ayu-

dan a establecer comparaciones objetivas entre distintas configuraciones de ejecución y entre arquitecturas heterogéneas; así lo señala Akulpally [38], “una parte crucial del desarrollo con PyCUDA es la posibilidad de recopilar datos de tiempo y rendimiento sin interferir con la ejecución normal del kernel”; esto resulta importante para identificar los cuellos de botella, además de la optimización que se le asignó a los hilos y los bloques; de esta forma se puede garantizar que el sistema está operando cerca de todo su potencial.

En este apartado se presentan y se describen las métricas que se usaron para evaluar el rendimiento de la implementación propuesta, por lo que se tocarán tres temas importantes: tiempo de ejecución, speed-up y la eficiencia. Con estas métricas se pudo analizar el comportamiento del sistema desde una perspectiva complementaria y se puede proporcionar una visión integral del rendimiento que se obtuvo al momento de usar la GPU.

3.7.1 Tiempo de ejecución

El tiempo de ejecución es una métrica importante para el análisis del rendimiento de aplicaciones que hacen uso de la herramienta de paralelización este indicador refleja la duración total que se necesitó desde el inicio al fin de la ejecución del kernel y esto permite evaluar la eficiencia en la distribución de tareas y el aprovechamiento de los recursos que se tienen disponibles de hardware, la fórmula para definir dicho parámetro es la siguientes Ecuaciónes 3 2:

$$T_{\text{secuencial}} = t_{\text{fin}} - t_{\text{inicio}} \tag{2}$$

$$T_{\text{paralelo}} = \frac{T_{\text{secuencial}}}{N} \tag{3}$$

donde $T_{\text{secuencial}}$ corresponde al tiempo que toma la ejecución en un solo núcleo, T_{paralelo} representa el tiempo estimado bajo un modelo ideal de paralelización y N es el número de núcleos o procesos empleados en la ejecución. Si bien en la práctica existen sobrecostos y sincronizaciones

que impiden alcanzar una reducción perfectamente lineal, estas expresiones permiten establecer una referencia clara para analizar el comportamiento del algoritmo en distintos entornos de hardware.

3.7.2 Throughput

El throughput representa la cantidad de combinaciones procesadas por segundo, esto durante la ejecución del algoritmo, y para calcularlo se emplea la siguiente ecuación: 4:

$$Throughput = \frac{\text{Iteraciones procesadas}}{\text{Tiempo de ejecución}} \tag{4}$$

En las pruebas realizadas, los resultados que se mostraron en la GPU alcanzaron rendimientos del orden de miles de millones de combinaciones por segundo, mientras que en la CPU, sobre todo en contraseñas medias y largas este rendimiento disminuyó de forma considerable, lo que refleja la diferencia de varios órdenes de magnitud entre ambos.

3.7.3 Complejidad

La complejidad del algoritmo de fuerza bruta se expresa de manera general mediante la Ecuación 5:

$$O(|\Sigma|^n) \tag{5}$$

Donde $|\Sigma|$ corresponde al tamaño del alfabeto a usarse; en este caso se usará el de 62 caracteres. n es la longitud de la contraseña y esta relación evidencia el crecimiento exponencial en el número de combinaciones posibles al incrementar la longitud de la clave, lo que se reflejó directamente en los tiempos de ejecución, sobre todo en el entorno secuencial.

3.7.4 Tasa de éxito

La tasa de éxito permite evaluar la proporción de ejecuciones en las cuales el algoritmo logra encontrar la contraseña dentro del espacio de búsqueda definido. Esta métrica se expresa como el porcentaje de ejecuciones exitosas en relación con el total de ejecuciones realizadas y se calcula mediante la siguiente Ecuación 6:

$$\text{Tasa\_exito} = \left(\frac{N_{\text{éxitos}}}{N_{\text{total}}} \right) \times 100 \tag{6}$$

donde $N_{\text{éxitos}}$ corresponde al número de ejecuciones en las que el algoritmo encontró la contraseña y N_{total} representa el total de pruebas realizadas. Esta métrica es útil para cuantificar la confiabilidad del proceso en diferentes configuraciones de hardware y bajo distintos volúmenes de combinaciones, permitiendo verificar que la distribución del trabajo en CPU y GPU asegure la exploración completa del espacio de búsqueda definido en cada experimento.

En la implementación que se desarrolló, el cálculo del tiempo se obtuvo mediante las funciones de sincronización y la medición de intervalos antes y después de la ejecución del kernel usado, lo que garantizó que los resultados reflejaran únicamente el tiempo consumido por las operaciones de la GPU. Como lo explican Akulapally et al., “PyCUDA también proporciona mecanismos convenientes para medir el tiempo de ejecución de los kernels utilizando eventos de CUDA”[38] , con esta herramienta se puede garantizar la precisión en la evaluación de la eficiencia del código paralelo porque permite obtener mediciones más consistentes y confiables.

Para medir el rendimiento del algoritmo en GPU, se empleó un método que logra calcular el tiempo que transcurrió desde el inicio hasta el final de la ejecución del kernel, y este procedimiento se implementó directamente en el programa mediante funciones que son capaces de registrar el instante del inicio y fin del proceso completo, lo que permite obtener una lectura precisa sobre el tiempo, y los resultados obtenidos fueron fundamentales para comparar la eficiencia entre las versiones secuenciales, multicore y GPU.

3.7.5 Speed-UP: Comparación entre ejecución secuencial y paralela

El concepto de speed-up es una medida que permite comparar el tiempo que tarda un algoritmo en ejecutarse de forma secuencial frente a una versión paralelizada. En este proyecto es especialmente útil ya que el ataque de fuerza bruta requiere procesar un espacio muy amplio de combinaciones, y la ejecución en múltiples núcleos o en una GPU puede reducir significativamente el tiempo total.

Para comprender los límites reales de esta aceleración es necesario considerar la Ley de Amdahl como lo señala Gilberto Días "todo programa consta de una parte que puede paralelizarse y otra que debe ejecutarse de manera secuencial"[40]. Esta distinción es importante ya que incluso si se incrementa de manera indefinida el número de procesadores, la parte secuencial termina imponiendo un límete al rendimiento que se puede alcanzar.

La aceleración obtenida con N procesadores se hace con la ecuación 7

$$S = \frac{1}{(1-P) + \frac{P}{N}} \tag{7}$$

Tal como lo expone Gilberto en su explicación matemática sobre laa relación entre la parte paralela P y la parte secuencial del programa señala que "la aceleración máxima viene dada por la parte que no puede paralelizarse"[40] y esto se expresa como la siguiente ecuación 8

$$S_{\text{max}} = \frac{1}{1-P} \tag{8}$$

Estas ecuaciones permiten interpretar de manera realista el comportamiento del algoritmo pero gran parte del proceso de generación de combinaciones y verificación de las mismas puede distribuirse entre varios hilos o núcleos, existen pasos que continúan siendo secuenciales, como lo es la inicialización, la coordinación entre procesos, el manejo de memoria o la verificación final y precisamente estos elementos explican por qué la aceleración no crece de forma

proporcional al número de recursos disponibles.

Tal como lo señala NVIDIA [35] “CUDA está diseñado para admitir diversos niveles de aceleración a través de su jerarquía de hilos, bloques y cuadrículas, lo que permite que miles de hilos ligeros se ejecuten de manera concurrente”. Esta afirmación refleja que la arquitectura de CUDA no solo se centra en la ejecución; también proporciona un marco más flexible; de esta manera se pueden aprovechar las concurrencias de los hilos.

Con esto en claro, el speed-up no debe interpretarse únicamente como una reducción en el tiempo que se realizó la ejecución, sino como un indicador de la capacidad de la GPU utilizada para resolver los problemas con una gran escala de eficiencia, lo que es difícil lograr con una CPU secuencial, y como lo dice NVIDIA [35], “Las aplicaciones que exponen un alto grado de paralelismo pueden lograr aceleraciones notables al asignar los cálculos a la arquitectura GPU”

3.7.6 Eficiencia: análisis del aprovechamiento de los recursos paralelos

La eficiencia es un indicador complementario al speed-up porque permite valorar en qué medida los recursos paralelos de la GPU están siendo aprovechados durante la ejecución del algoritmo. En términos generales, la eficiencia se define como la relación entre el speed-up obtenido y el número de unidades de procesamiento usadas, de acuerdo con NVIDIA [35], “La arquitectura de CUDA distribuye los hilos en bloques y cuadrículas, lo que permite un control explícito de la utilización de los multiprocesadores y de la memoria compartida”. Esta organización se convierte en un aspecto determinante porque una asignación errónea de hilos o una gestión que es deficiente en cuanto a la memoria puede causar una subutilización de recursos, lo que afecta directamente a la eficiencia.

Como se observa en la Ecuación (7), la eficiencia (E) se calcula dividiendo el *speedup* (S) entre el número de *procesadores* (P), y de esta manera un valor cercano a 1 indica que se aprovechan de forma óptima los recursos paralelos, mientras que valores bajos reflejan pérdidas de rendimiento debido a que se sobrecarga la comunicación, la sincronización o las limitaciones

inherentes al algoritmo.

$$E = \frac{S}{P} = \frac{T_{secuencial}}{P \times T_{paralelo}} \tag{7}$$

NVIDIA [35] también enfatiza que “El rendimiento óptimo depende de mantener a los multiprocesadores ocupados con suficientes bloques de hilos para ocultar las latencias de memoria”; esto implica que la eficiencia no se va a alcanzar únicamente por el hecho de disponer de miles de núcleos; se va a alcanzar por la correcta planificación del paralelismo y la sincronización entre hilos.

El análisis de eficiencia en este trabajo se orienta a evaluar cómo las configuraciones entre hilos y bloques logran influir en el rendimiento del algoritmo de fuerza bruta, lo que determina si el hardware de la GPU se está aprovechando de una forma correcta o si existen factores de subutilización.

3.8 Limitaciones del entorno de pruebas y ejecución

El uso de Google Colab Pro permitió disponer de una GPU NVIDIA A100 y una capacidad de procesamiento superior a la que se ofrece en entornos HPC; este entorno aún presenta ciertas limitaciones que deben ser consideradas en investigaciones de carácter técnico. A pesar de que el rendimiento obtenido durante las pruebas fue estable y aceptable para la implementación del algoritmo de fuerza bruta, la plataforma continúa imponiendo restricciones relevantes relacionadas con la gestión de recursos, la disponibilidad del hardware y la compatibilidad de algunas librerías empleadas en el desarrollo.

Durante la migración del proyecto hacia un entorno de computación acelerada, surgieron dificultades asociadas a la incompatibilidad entre Numba y la versión de CUDA instalada en Colab Pro. Estas incompatibilidades provocaron errores persistentes durante la compilación y ejecución de kernels, lo que obligó a reemplazar la implementación inicial por PyCUDA. Esta

librería ofreció un control más directo sobre la memoria, la compilación del kernel y el manejo de recursos del dispositivo, lo que permitió asegurar la estabilidad del algoritmo y continuar con las pruebas sin interrupciones. Estos inconvenientes evidencian que las limitaciones no provienen únicamente del código desarrollado, sino también de la infraestructura del entorno y la forma en que gestiona las dependencias del ecosistema CUDA.

3.8.1 Restricciones impuestas por Google Colab

Aunque la versión de pago de Colab ofrece acceso prioritario a hardware más avanzado, como lo es la GPU NVIDIA A100 y opciones de mayor capacidad de memoria RAM, el usuario no puede controlar la disponibilidad del recurso durante la asignación. Colab puede asignar diferentes configuraciones de hardware en función de la carga del sistema, lo que introduce un grado de variabilidad en las pruebas y requiere ajustar la ejecución dependiendo del dispositivo asignado en cada sesión.

Además, las sesiones de Colab Pro, si bien son más extensas que en la versión gratuita, siguen teniendo límites de tiempo y desconexión por inactividad, y esto puede interrumpir ejecuciones prolongadas o pruebas que requieren múltiples iteraciones extensas. Estos factores no impidieron el desarrollo del proyecto, pero representan un aspecto operativo importante cuando se intenta simular condiciones propias de entornos HPC con control total de hardware.

3.8.2 Consideraciones sobre escalabilidad en entornos simulados

El acceso a la GPU NVIDIA A100 permitió evaluar el comportamiento del algoritmo en un escenario más cercano al procesamiento de alto rendimiento, especialmente en tareas con un grado elevado de paralelización. Sin embargo, Colab sigue siendo un entorno simulado y un sistema HPC completo, por lo que existen limitaciones respecto a la libertad para escalar los recursos; por ejemplo, trabajar con varias GPUs en paralelo no es posible, modificar directamente configuraciones avanzadas del driver o elegir arquitecturas específicas para optimizar kernels

CUDA según la familia exacta del GPU.

Estas restricciones impiden analizar el rendimiento del algoritmo en configuraciones más complejas o distribuidas, como las que se encuentran en clústeres reales que emplean múltiples nodos o varias GPU A100 trabajando de manera cooperativa.

3.8.3 Observaciones técnicas sobre uso de recursos

Durante la implementación se evidenciaron diferencias importantes entre Numba y PyCUDA en cuanto a su adaptación al entorno de Colab Pro. Numba presentó fallos recurrentes derivados de incompatibilidades con la versión de CUDA preinstalada, lo que limitó la posibilidad de ejecutar el kernel de manera estable. En contraste, PyCuda proporcionó un nivel de control más detallado sobre la memoria del dispositivo, el uso de memoria constante y compartida, la gestión de operaciones atómicas y la configuración del kernel. Estas características fueron esenciales para garantizar un uso adecuado de la GPU NVIDIA A100 y evitar errores durante la ejecución.

Aunque el hardware disponible fue adecuado para el desarrollo del proyecto, la experiencia mostró que las restricciones del entorno no necesariamente están relacionadas con la potencia computacional, sino con la gestión interna del software, las versiones preinstaladas y la capacidad limitada de personalizar componentes críticos del ecosistema CUDA. Esta situación refuerza la necesidad de considerar cuidadosamente las dependencias técnicas cuando se desarrolla código orientado a GPU y resalta la relevancia de elegir herramientas más flexibles y robustas, como PyCUDA, para asegurar la continuidad del trabajo experimental.

CAPÍTULO IV

RESULTADOS Y ANÁLISIS

4.1 Introducción

En este capítulo se presentan los resultados obtenidos tras ejecutar el algoritmo de fuerza bruta en los diferentes entornos de prueba como lo son la CPU Local en modo secuencial y multinúcleo así mismo el entorno de la nube mediante Google Colab, donde se empleó la GPU NVIDIA A100 para evaluar el rendimiento en un escenario de cómputo de alto rendimiento.

Para garantizar la coherencia entre las pruebas, la generación de las contraseñas utilizadas siguió los liniamientos de la norma ISO/IEC 27001, que recomienda el uso de las combinaciones con niveles adecuados de complejidad y entropía [26]. Los experimentos abarcan contraseñas cortas, medias y largas lo que permite observar cómo el tiempo de cómputo crece conforme aumenta el espacio de búsqueda. Si bien las contraseñas largas demandaron tiempos demasiados altos, se realizaron mediciones hasta donde el entorno lo permitió, tanto en la CPU como en la GPU, lo que ofreció una perspectiva clara del comportamiento del algoritmo en escenarios de alta complejidad.

Con el fin de facilitar el análisis, los resultados se presentan de forma consolidada mediante tablas de promedios y gráficos comparativos por longitud y arquitectura de ejecución. Las métricas analizadas son: tiempo de ejecución, throughput, speed-up, eficiencia y escalabilidad, lo que permite constrastar el dendimiento entre procesamiento secuencial, paralelismo multinúcleo y la aceleracion que ofrece la GPU A100 en un entorno HPC.

4.2 CPU local

Los resultados muestran que, para contraseñas cortas, la versión secuencial obtiene un mejor rendimiento que el procesamiento multinúcleo y como se muestra en la tabla 2 el tiempo promedio secuencia fue de 1.29 segundos, mientras que el multinúcleo aumentó a 2.52 segundos debido al overhead de paralelización, que incluyen la creación de hilos, sincronización y división del trabajo. Como resultado, el speed up fue de apenas 0.50 y la eficiencia alcanzo un 12.88 %, indicando que los cuatro núcleos no se utilizan de manera efectiva en cargas de trabajo pequeñas. Esto confirma que, en problemas de baja complejidad, el costo de paralelizar supera los beneficios.

Tabla 2.
Promedios de rendimiento para contraseñas cortas (4 caracteres) en CPU local

Métrica	**Secuencial**	**Multinúcleo (4 núcleos)**
Tiempo promedio (s)	1.29	2.52
Throughput promedio (iter/s)	14.46 M/s	7.59 M/s
Speedup	–	0.52
Eficiencia	–	12.88 %

Aunque el espacio de búsqueda es mayor, el paralelismo tampoco mejora el rendimiento para contraseñas de seis caracteres y el tiempo secuencial se muesrta en la tabla 3 el promedió es de 3953.62 segundos, mientras que el multinúcleo incrementó hasta 10067.09 segundos, mostrando que el paralelismo no logró reducir la carga computacional efectiva. El speedup fue de 0.41 y la eficiencia de 10.30 %, valores incluso menores que los obtenidos en contraseñas cortas evidenciando que el algoritmo presenta poca escalabilidad y que la sobrecarga del paralelismo continúa siendo dominante incluso cuando el problema crece

Tabla 3.
Promedios de rendimiento para contraseñas medias (6 caracteres) en CPU local

Métrica	Secuencial	Multinúcleo (4 núcleos)
Tiempo promedio (s)	3953.62	10067.09
Throughput promedio (iter/s)	7.36 M/s	4.20 M/s
Speedup	–	0.41
Eficiencia	–	10.30 %

4.2.1 Tiempo de ejecución

En el grafico 10 se logra evidenciar una diferencia significativa entre la ejecución secuencial y la paralela multinúcleo. Para contraseñas de 4 caractereses, ambos tiempos son bajos, pero la versión multinúcleo aumenta de manera notable. Esto demuestra que el paralelismo introduce una sobrecarga mayor que el trabajo real, lo que provoca un rendimiento inferior en problemas de baja complejidad.

Figura 10.
Promedio del tiempo de ejecución del algoritmo secuencial y multinúcleo

Para contraseñas de 6 caracteres, la tendencia se mentiene: la ejecución secuencial requiere 3952.62s, mientras que la versión múltinucleo asciende a 10067.09 s, más del doble. Esto confirma que, incluso cuando el espacio de búsqueda crece, el algoritmo en CPU multinúcleo no

logra distribuir de forma eficiente la carga, y la sincronización entre hilos penaliza el rendimiento. En conjunto, el gráfico muestra que la CPU, en este caso escala negativamente al añadir más núcleos, lo cual es indicativo de una implementación que no se beneficia del paralelismo para un algoritmo de fuerza bruta.

4.2.2 Throughput

El throughput mide cuántas iteraciones por segundo puede procesar el sistema y en el grafico 11 se demuestra que, tanto para contraseñas cortas como medias la ejecución secuencial muestra un mayor througput que la paralela:

- 4 Caracteres: Secuencial - 14.45 M iter/s Multinúcleo - 7.59 M iter/s
- 6 Caracteres: Secuencial - 7.36 M iter/s Multinúcleo - 4.20 M iter/s
- 9 caracteres Las contraseñas de mayor longitud no se lograron evaluar de forma secuencial y multinúcleo ya que el proceso computacional es demasiado elevado por lo que no se obtuvieron resultados dentro de los parametros permitidos.

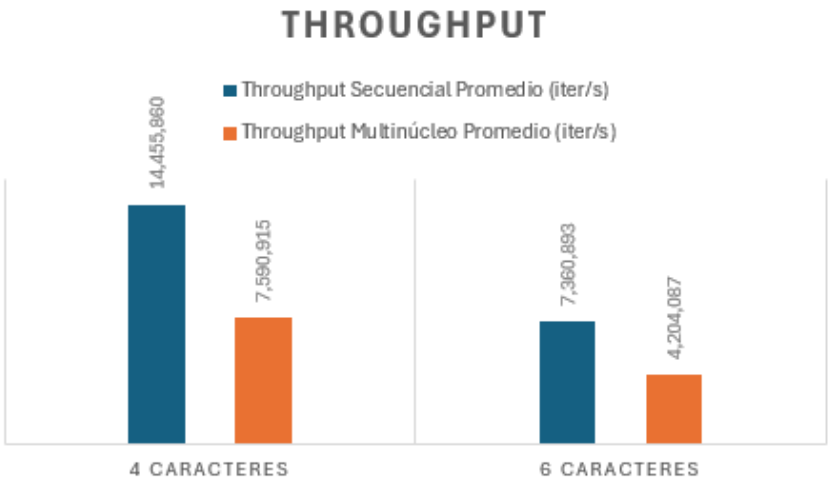

Figura 11.
Promedio del Throughput Secuencial y Multinúcleo (iter/s)

Esto indica que el paralelismo no solo aumenta el tiempo total, sino que reduce la tasa de procesamiento. La causa principal es que el algoritmo debe coordinar múltiples hilos que com-

parten estructuras y actualizan estados de forma dependiente, generando cuellos de bottela y mayor latencia por sincronización.

4.2.3 Speedup (solo para multinúcleo)

El gráfico 12 de speedup muestra valores inferiores a 1 tanto para contraseñas de 4 caracteres es de 0.52 y cpara 6 caracteres es de 0.41, esto implica que la versión paralela es mas lenta que la secuencial para ambas longitudes.

Un speed up menor a 1 revela que la sobrecarga de coordinar varios núcleos supera los beneficios de dividir el trabajo. Además, la caída del speedup al aumentar la longitud refleja que el algoritmo no escala favorablmente a medida que el espacio de búsqueda crece, lo cual es un compartamiento contrario y esto se puede dar al usar un procesador de generación pasada ya que no estan diseñados para soportar cargas de trabajo tan excesivas.

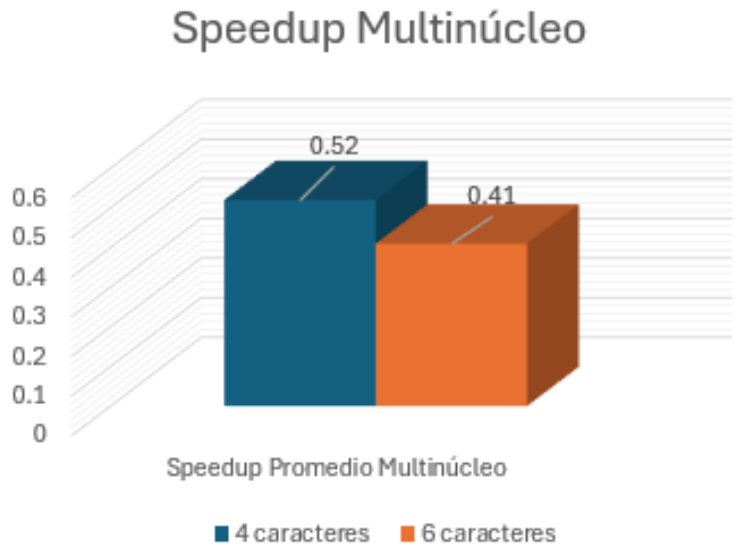

Figura 12.
Speedup Promedio Multinúcleo en 4 nucleos

4.2.4 Eficiencia

El grafico 13 refleja que tan bien se aprovecha el hardware disponible entonces: para los escenarios donde las contraseñas fueron de 4 caracteres se obtuvo el 12.88 % de eficiencia y en el escenario de las contraseñas de 6 caracteres se obtuvo una eficiencia de 10.30 % por esto en ambos casos baja la eficiencia cuando el problema aumenta y el algoritmo solo aprovecha los recursos de forma efectiva alrededor del 10-13 % del total.

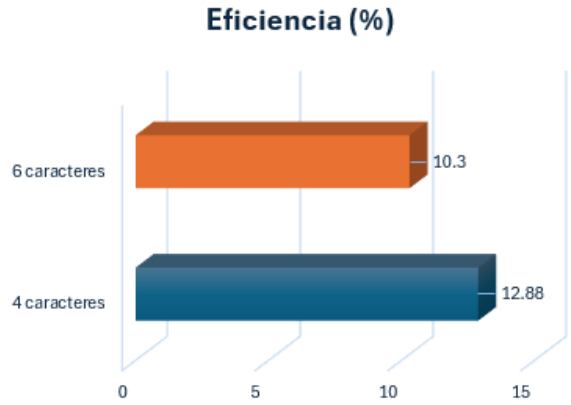

Figura 13.
Porcentaje de la eficiencia de recursos secuenciales y multinúcleos.

La eficiencia baja porque existen factores donde se usan más recursos como lo es la sincronización entre hilos, la administración de recursos, los repartos de desigualdad del espacio de búsqueda y esperas activas o bloqueos, entonces el algoritmo presenta pérdidas sustanciales de rendimiento al intentar paralelizarlo, y que la arquitectura multinúcleo no resulta adecuada para este tipo de búsqueda exhaustiva en CPU.

Una vez analisados los cuatro gráficos se puede decir que la versión secuencial supera consistemente al procesamiento multinúcleo en todos los indicaroes, el algoritmo de fuerza bruta no escala adecuadamente en una CPU Intel Core I3 y presenta un costo de paralelización que supera los beneficios potenciales y para obtener mejoras reales, se requiere una arquitectura más adecuada como una GPU donde la paralelización es masiva y se adapta mejor a algoritmos dependientes del espacio de búsqueda.

4.3 Google Colab

En las contraseñas cortas como se observa en la tabla 4 el tiempo de búsqueda es reducido para todas las arquitecturas, en el modo secuencial ofrece resultados rápidos, pero el uso de dos núcleos no aporta una mejora real y de hecho, el tiempo aumenta ligeramente, lo que sugiere que el volumen de trabajo es tan pequeño que la coordinación entre esos dos núcleos termina añadiendo más carga que beneficio.

La GPU, por otro lado, logra una diferencia muy clara. En su ejecución es casi inmediata y el número de operaciones procesadas por segundo es muy superior a la CPU. Esto se refleja en un speedup elevado y en una eficiencia que indica un uso favorable de los recursos del dispositivo. En resumen, para contraseñas de esta longitud, la GPU es la opción más efectiva, mientras que el multinúcleo no ofrece ventajas frente a la ejecución secuencial.

Tabla 4.
Rendimiento para contraseñas cortas (4 caracteres) en Google Colab

Métrica	Secuencial	Multinúcleo (2 núcleos)	GPU A100
Tiempo promedio (s)	1.44143	2.53750	0.00127
Throughput (iter/s)	6,877,005.40	3,810,565.26	4,031,244,837.83
Speedup	– no aplica	0.568	1134.96
Eficiencia	– no aplica	0.284	10.51
Escalabilidad	– no aplica	0.554	586.06

Como se observa en la tabla 5 en las contraseñas de longitud media aparece una diferencia mucho mas marcada entre las arquitecturas. El tiempo secuencial aumenta considerablemente y el modo multinúcleo no logra compensar esa carga, y su speedup es menor a uno y la eficiencia es baja, lo que indica que el trabajo adicional de coordinación entre núcleos afecta más de lo que ayuda.

La GPU mantiene un comportamiento muy diferente. Aunque los tiempos son mayores que las contrseñas cortas aun así sigue ofrenciendo un rendimiento muy superior a la de la CPU. Su

throughput aumenta de forma significativa y el speedup obtenido frente al modo secuencial es bastante elevado y en conjunto, estos resultados muestran que, a medida de que la complejidad crece, la GPU es la única arquitectura que realmente puede escalar y logra mantener tiempos competitivos en este tipo de pruebas.

Tabla 5.
Rendimiento para contraseñas medias (6 caracteres) en Google Colab

Métrica	Secuencial	Multinúcleo (2 núcleos)	GPU A100
Tiempo promedio (s)	5150.85	13801.24	1.39801
Throughput (iter/s)	5,536,200.41	992,812.57	29,444,887,480.98
Speedup	– no aplica	0.373	3682.40
Eficiencia	– no aplica	0.186	34.10
Escalabilidad	– no aplica	0.179	5318.60

En las contraseñas largas se aprecia un cambio drástrico respecto a las longitudes anteriores. Ni en el modo secuencial ni el multinúcleo lograron completar las pruebas dento de un tiempo razonable, lo que refleja el crecimiento exponencial del espacio de búsqueda al aumentar la cantidad de caracteres. Incluso en la GPU, que en los casos anteriores mostró un rendimiento muy superior, solo logro resolver tres contraseñas y con tiempos bastante elevados.

Tabla 6.
Rendimiento para contraseñas largas (9 caracteres) en Google Colab

Métrica	Secuencial	Multinúcleo (2 núcleos)	GPU A100
Tiempo promedio (s)	– tiempo excedido	– tiempo excedido	14032.51
Throughput (iter/s)	– sin datos	– sin datos	8,585,056.72
Speedup	– no aplica	– no aplica	– no aplica
Eficiencia	– no aplica	– no aplica	– no aplica
Escalabilidad	– no aplica	– no aplica	– no aplica
Contraseñas resueltas	0 de 9	0 de 9	3 de 9

Aunque el trhougput de la GPU sigue siendo considerable, el tiempo total necesario condir-

ma que, a partir de longitudes de 9 o más caracteres, la fuerza bruta se vuelve impráctica incluso con hardware especializado. Esto evidencia que el aumento en la complejidad no solo afecta al rendimiento, si no que puede llevar a que el problema quede fuera del alcance de los recursos disponibles, aun cuando se utilice arquitecturas altamente paralelas.

4.3.1 Tiempo de ejecución

La figura 14 muestra con claridad cómo la arquitectura influye en la capacidad real de romper contraseñas mediante fuerza bruta. En las contraseñas cortas, tanto en el modo secuencial como el multinúcleo se consiguen tiempos muy reducidos, esto se da por el espacio de búsqueda ya que es relativamente pequeño pero la GPU sobresale al completar todas las pruebas en cuestion de milisegundos, lo que demuestra que incluso las tareas simples se ven favorecidas por el paralelismo masivo.

En contraseñas de Longitud media, el rendimiento del multinúcleo cae de forma notable debido al overhead involucrado en la coordinación entre hilos, lo cual evidencia que no siempre más núcleos implica mejoras directas en fuerza bruta. La GPU por otro lado, mantiene tiempos bajos pese al incremento del espacio búsqueda, dejando claro que su arquitectura está mejor adaptada a este tipo de cargas.

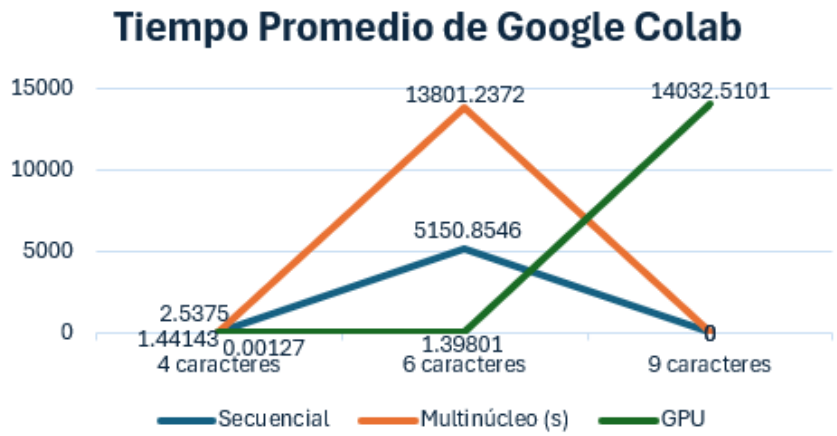

Figura 14.
Tiempo Promedio de las ejecuciones en Google Colab

Finalmente, en contraseñas largas, solo la GPU logro completar algunos casos dentro del límete de tiempo aceptable, mientras que las versiones secuenciales y multinúcleo no lograron finalizar ninguna prueba y este resultado es clave para esta investigación, ya que demuestra que apartir de cierta complejidad, la capacidad de un sistema para romper contrasñas dependen casi exclusivamente de plataformas HPC como las GPU modernas.

4.3.2 Throughput

La figura 15 permite observar cuántas combinaciones por segundo puede procesar cada arquitectura. En contraseñas cortas, aunque el secuecial mantiene cifras aceptables, la GPU supera amplimante al resto, porque procesa millones de combinaciones por segundo incluso en sus casos menos exigentes.

A medida de que la longitud aumenta, la tasa del secuencial y del multinúcleo disminuyen y esto muestra que su capacidad no escala adecuadamente frente a un espacio de búsqueda creciente. Por lo contrario, la GPU mantiene valores significativamente altos, lo que reafirma su potencial como herramienta principal para evaluar la fortaleza de contraseñas en un entorno HPC.

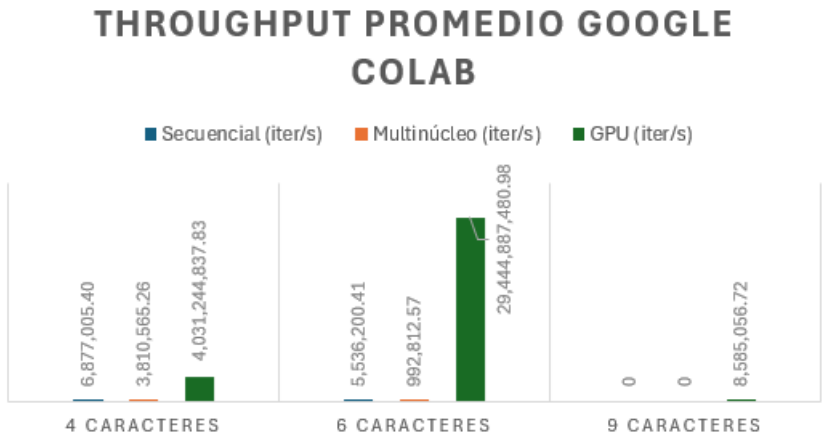

Figura 15.
Throughput promedio obtenido de las pruebas en Google Colab

Finalmente este comportamiento es importante ya que demuestra que la resistencia real de una contraseña no depende únicamente de la longitud o complejidad implementada, también

depende del tipo de hardware que se dipone para poder atacar las contraseñas.

4.3.3 Speedup

En la figura 16 se muestra con claridad cuánto mejora el rendimiento al utilizar la tecnica de paralelismo. Para contraseñas cortas, el multinúcleo obtiene beneficios mínimos, lo que confirma que el paralelismo tradicional en CPU tiene limitaciones frente a los problemas cuya carga no justifica la creación de múltiples hilos.

La GPU, por su lado, logra mejoras enormes desde las primeras pruebas, esto se debe a la capacidad de ejecutar miles de hilos en paralelo. En contraseñas de longitud media, su peedup se dispara aún más lo que indica que el incremento del espacio de búsqueda favorece la eficiencia del paralelismo masivo.

Debido a estos resultados podemos concluir que la GPU no solo acelera el proceso, sino que vuelve viable romper contraseñas que en una CPU serían prácticamente inaccesibles, lo que hace que se redefina el concepto páctico de contraseña segura.

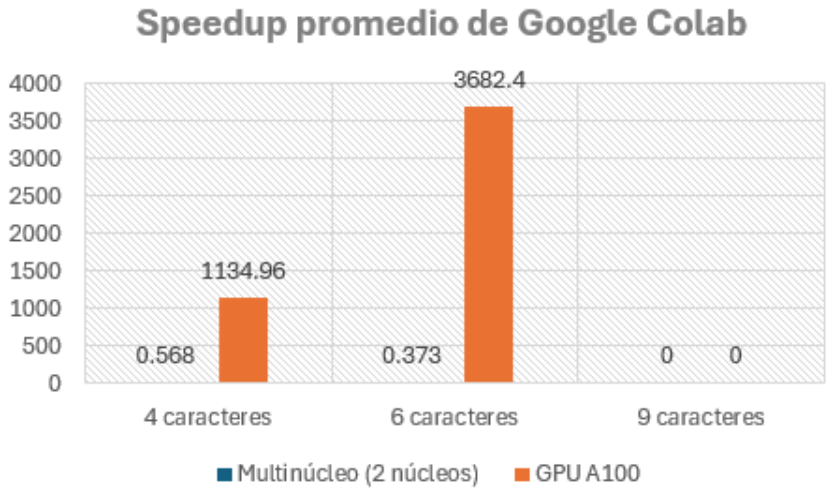

Figura 16.
Speedup promedio obtenido de las pruebas realizadas en Google Colab

4.3.4 Eficiencia

En la figura 17 de muestra la eficiencia, en multinúcleo se mantiene baja en todos los escenarios y esto demuestra que la arquitectura de CPU no escala tan bien frente a problemas altamente paralelizables. La sincronización, el reparto del trabajo y la latencia entre hilos consumen buena parte del tiempo total, algo que limita su utilidad cuando se busca evaluar contraseñas de alta compplejidad mediante fuerza bruta.

Por otro lado, la eficiencia de la GPU aumenta conforme crece la longitud de la contraseña. Esto confirma que la arquitectura de miles de núcleos es especialmente adecuada para manejar espacios de búsqueda masivos, ya que la cantidad de trabajo disponible permite ocupar un mayor número de unidades de procesamiento simultaneamente.

Este comportamiento respalda que los sistemas HPC permiten evaluar de una forma realista el costo computacional que requiere romper contraseñas complejas, y esa evaluación no es posible con un hardware convencional.

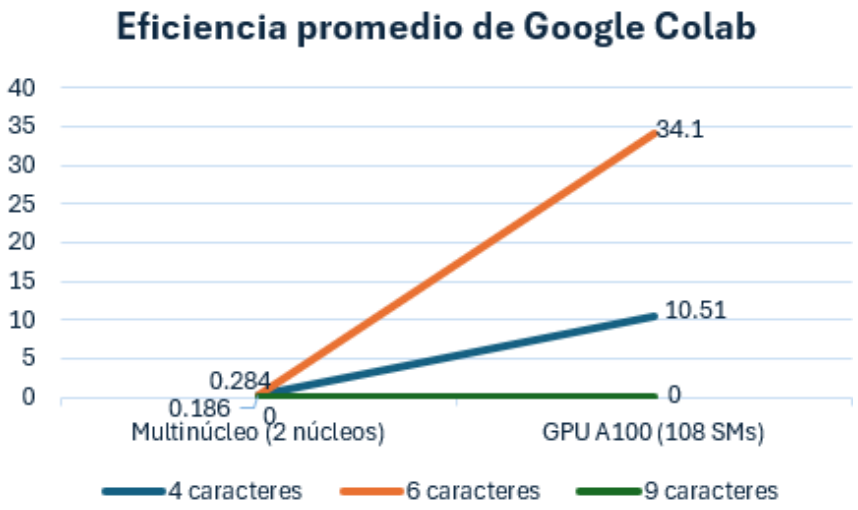

Figura 17.
Eficiencia promedio obtenido despues de las pruebas realizadas en Google Colab

4.3.5 Escalabilidad

En la figura 18 se muestra qué tan bien se aprovechan los recursos adiconales de cada arquitectura. En contraseñas cortas, la GPU ya presenta una ventaja clara, pero es en las contraseñas de longitud media donde se logra observar el verdadero potencial: la escalabilidad se multiplica por miles, lo que demuestra que la arquitectura aumenta el rendimiento a medida que el volumen de trabajo crece.

El multinúcleo presenta una escalabilidad limitada, lo que confirma que su modelo no es el más adecuado para los problemas masivos como lo es la fuerza bruta. En contraseñas largas no puede calcularse la escalabilidad por la razon que las pruebas con contraseñas largas son demasiadas exigentes para una CPU tradicional y esto refuerza que solo la GPU ofrece la capacidad de escalar lo suficiente como para lograr afrontar espacios de búsqueda gigantescos.

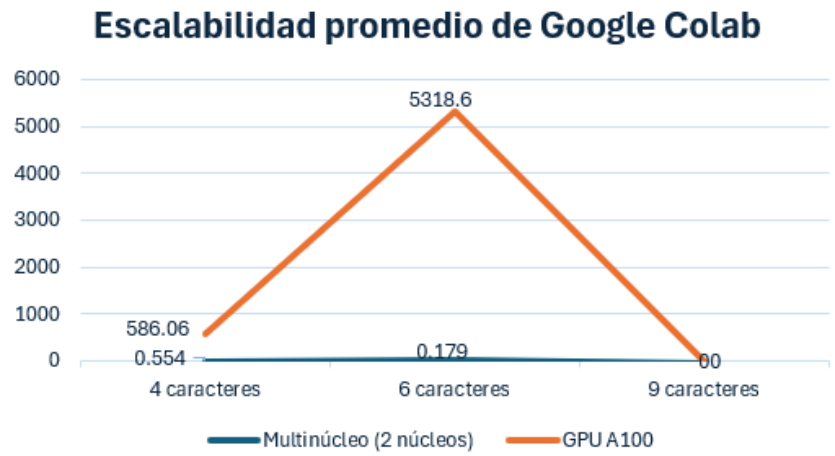

Figura 18.
Escalabilidad promedio tras las pruebas realizadas en Google Colab

4.4 Discusión general de resultados

Los resultados obtenidos en las distintas arquitecturas permiten establecer una visión clara sobre el impacto que tienen los recursos de cómputo en la posibilidad real de romper contraseñas mediante fuerza bruta. En primer lugar, las pruebas demuestran que el rendimiento no depende únicamente de la longitud o complejidad de la contraseña, sino que también depende

del hardware empleado. Mientras que las contraseñas cortas pueden ser resueltas sin mayores dificultades incluso en un modo secuencial, este comportamiento cambia radicalmente cuando se incrementa el espacio de búsqueda.

En Google Colab, las contraseñas de 4 caracteres fueron resueltas por todas las arquitecturas, con tiempos que van desde milisegundos en la GPU hasta segundos en el modo secuencial. Lo que confirma que, para combinaciones pequeñas, la fuerza bruta es completamente ciable incluso sin tener recursos HPC disponibles. Sin emabrgo, al pasar a 6 caracteres la situación cambia; aunque tanto el modo secuanl como la GPU logran completar todas las pruebas, el multinúcleo presenta tiempos significativamente más altos, lo que eviencia que no todo paralelismo mejora el rendimiento y aun así, la GPU mantiene una ventaja contundente, procesando millones de combinaciones por segundo y reduciendo drástricamente el tiempo necesario para romper este tipo de contraseñas.

El comportamiento más relevante se observa en las contraseñas de 9 caracteres. En este caso, ni el procesamiento secuencial ni el multinúcleo pudieron completar la búsqueda dentro del tiempo límite. Solo la GPU A100 logró resolver tres casos y, aun así, rquirió entre 12.769 y 16.199 segundo lo que se traduce al rededor de 3.5 y 5 horas para encontrarlos. Este resultado es especialmente importante porque demuestra que, a pesar de contar con un entorno HPC, la fuerza bruta sigue teniendo límetes prácticos cuando el espacio de búsqueda crece de una forma exponencial. Es decir, incluso con hardware especializado, romper contraseñas largas con fuerza bruta continúa siendo un proceso costoso.

Apartir de estos hallazgos, pueden establecerse varias recomendaciones sobre la creación de contraseñas seguras. En primer lugar, las contraseñas de 4 a 5 caracteres deben considerarse completamente inseguras, pues se pueden resolver en segundos incluso sin hardware especializado. Las contraseñas de 6 caracteres ofrecen una resistencia ligeramente mayor, pero continúan en la linea de vulnerabilidad frente a una GPU moderna que puede procesar millones de combinaciones por segundo. Por tanto, su nivel de seguridad es insuficiente frente a aun atacante que cuente con acceso a recursos HPC.

Las contraseñas de 9 caracteres representan un punto de inflexión interesante porque aunque algunas pueden ser descubiertas mediante GPU, el tiempo requerido crece a varias horas o incluso en la mayoría de casos ni siquiera pudieron completarse dentro del límete trazado. Esta diferencia evidencia que, a parir de los 9 caracteres, la resistencia frente a ataques de fuerza bruta comienza a volverse significativa, especialmente si se combinan mayúsculas, minúsculas, dígitos y símbolos.

En este contexto, la recomendación más adecuada es utilizar contraseñas de al menos 10 o 12 caracteres, empleando un conjunto de caracteres amplio. Con estas longitudes, incluso una GPU de alto rendimiento tardaría días o semanas en avualar el espacio de búsqueda completo. Esto refuerza la idea de que, aunque la potencia de los sistemas HPC ha aumentado considrablemente, la complejidad de las contraseñas puede seguir creciendo de forma que mantenga una ventaja significativamente frente a los ataques de fuerza bruta.

Finalmente, los resultados confirman que los entonros HPC son una herramienta valiosa para evaluar la fortaleza real de las contraseñas. Asimismo, demuestran que la seguridad efectiva depende de una combinación adecuada entre la longitud, diversidad de caracteres y prácticas de gestión, más que de mecanismos teóricos. La disponibilidad de hardware especializado obliga a evaluar el estándar mínimo de seguridad, ya que lo que ntes requería días de cómputo ahora puede resolverse en cuestión de horas.

CONCLUSIONES

Los resultados mostraron con claridad que el rendimiento del ataque de fuerza bruta cambia de forma norable según el hardware que se utilice. La CPU, tanto en secuencial como multi-núcleo, ofreció mejoras limitadas frente al crecimiento del espacio de búsqueda. En cambio, la GPU A100 utilizada en Colab Premium redujo de forma significativa los tiempo de prueba. Lo que confirma que la computación de alto rendimiento es clave para acelerar este tipo de tareas.

La experimentación dejó en evidencia que las contraseñas cortas o poco variadas pueden romperse en tiempos muy reducidos, incluso sin hardware especializado. A partir de los 9 caracteres, la dificultad aumenta de forma considerable, especialmente cuando se combinan distintos tipos de símbolos. Esto refuerza la importancia de adoptar claves extensas y bien estructuradas. Las pruebas permitieron comprobar de manera práctica cómo la longitud influye directamente en la resistencia ante ataques.

La GPU A100 mostró un rendimiento muy superior frente a la CPU, logrando acelerar el proceso de búsqueda de manera contundente. Su capacidad para manejar miles de hilos de forma simultanea permitió explorar combinaciones a un ritmo que la CPU no puede igual. Además, mantuvo un comportamiento estable incluso en escenarios de mayor complejidad.

El uso de PyCUDA permitió aplicar paralelización directa sobre la GPU, distribuyendo la carga entre miles de hilos de forma eficiente. Esta estrategia redujo de manera evidente los tiempos de ejecución y permitió escalar el algoritmo conforme aumentaban las combinaciones posibles. La implementación mostró que la GPU aprovecha mejor el paralelismo que la CPU, sobre todo en tareas repetitivas como la fuerza bruta.

RECOMENDACIONES

En cuanto a las recomendaciones se plantean las siguientes:

- Aumentar la complejidad mínima recomendada de contraseñas. Con base en los experimentos realizados, se recomienda utilizar contraseñas de al menos 10 a 12 caracteres con mezcla de mayúsculas, minúsculas, números y símbolos.
- Expandir las pruebas a contraseñas de mayor longitud y combinaciones más amplias. Es conveniente incluir contraseñas de 12, 14 y 16 caractere.
- Integrar librerías y frameworks alternativos para paralelización. Aunque PyCUDA funcinó correctamente, se recomienda evaluar entornos como Numba, CuPy o CUDA C/C++, los cuales pueden ofrecer otras mejoras o incluso reducir el costo de ejecución en GPU.
- Ejecutar futuras pruebas en entornos HPC con hardware más potente. El uso de GPUs es de gama superior como las RTX permitirá observar comportamientos más estables, valores de throughput superiores y mejoras sustanciales en escalabilidad.
- Aplicar el enfoque paralelo a otros problemas de seguridad. El descifrado por fuerza bruta solo es un caso dentro de múltiples aplicaciones intensivas. Este enfoque puede extenderse a análisis criptográficos, verificación de hashes, cifrado simétrico, simulaciones, entre otros, lo que permite ampliar la relevancia del trabajo.

BIBLIOGRAFÍA

[1] N. J. Palatty. "30+ Password Statistics You Need To Know In 2025," visitado 12 de mar. de 2025. dirección: https://www.getastra.com/blog/security-audit/password-statistics/.

[2] S. Kemp. "Digital 2023: Global Overview Report," visitado 11 de mar. de 2025. dirección: https://datareportal.com/reports/digital-2023-global-overview-report.

[3] Kaspersky. "Kaspersky demuestra que el 59 % de las contraseñas se descifrarían en menos de una hora," visitado 20 de mar. de 2025. dirección: https://surl.lu/nuufnr.

[4] Google Cloud. "¿Qué es la computación de alto rendimiento?" Visitado 17 de mar. de 2025. dirección: https://cloud.google.com/discover/what-is-high-performance-computing?hl=es-419.

[5] C. Bordón, "NLHPC: La experiencia de trabajar usando recursos de computación de alto desempeño," *Observatorio Económico*, vol. —, pág. 166, 2022.

[6] Alapsi. "La Importancia de las Contraseñas Seguras: Cómo Crearlas y Administrarlas," visitado 17 de mar. de 2025. dirección: https://www.alapsi.org/ciberconcienciencia/ciberseguridad-para-el-trabajo/la-importancia-de-las-contrasenas-seguras-como-crearlas-y-administrarlas.

[7] S. Choi, D. Kim y S. C. Seo, "Parallel implementation of scrypt: A study on GPU acceleration for password-based key derivation function," *Journal of information and communication convergence engineering*, vol. 22, n.º 2, págs. 98-108, 30 de jun. de 2024, ISSN: 2234-8883. DOI: 10.56977/jicce.2024.22.2.98. visitado 5 de mar. de 2025. dirección: https://jicce.org/journal/view.html?doi=10.56977/jicce.2024.22.2.98.

[8] N. d. Balboa. "La importancia del uso de contraseñas seguras," visitado 20 de mar. de 2025. dirección: https://www.itdigitalsecurity.es/actualidad/2024/05/la-importancia-del-uso-de-contrasenas-seguras.

[9] L. B. P. Arbesú. "¿Cómo fortalecer las contraseñas ante la nueva era de ataques?" Visitado 18 de mar. de 2025. dirección: https://www.computerweekly.com/es/consejo/Como-fortalecer-las-contrasenas-ante-la-nueva-era-de-ataques.

[10] L. Bonilla. "Qué es la Computación de Alto Rendimiento (HPC) y por qué es el futuro," visitado 18 de mar. de 2025. dirección: https://www.datacentermarket.es/dcm-xl/que-es-la-computacion-de-alto-rendimiento-hpc-y-por-que-es-el-futuro/.

[11] M. Bishop, "What is computer security?" *IEEE Security & Privacy*, vol. 1, n.º 1, págs. 67-69, ene. de 2003, ISSN: 1540-7993, 1558-4046. DOI: 10.1109/MSECP.2003.1176998. visitado 22 de mayo de 2025. dirección: https://ieeexplore.ieee.org/document/1176998/.

[12] F. L. Caicedo Goyes, "Exploración de estrategias avanzadas en computación de alto rendimiento: Un Análisis Integral y Perspectivas Emergentes," *REVISTA ODIGOS*, vol. 5, n.º 2, págs. 9-32, 10 de jun. de 2024, ISSN: 2697-3405. DOI: 10.35290/ro.v5n2.2024.1174. visitado 4 de mar. de 2025. dirección: https://revista.uisrael.edu.ec/index.php/ro/article/view/1174.

[13] P. Fernández-Arruti Gallego, "Paralelización en CUDA de varios modos de ataque a algoritmos de hashing," ago. de 2020. visitado 24 de jun. de 2025. dirección: http://hdl.handle.net/2183/27336.

[14] P. Bordón y F. Crespo, "NLHPC: La experiencia de trabajar usando recursos de computación de alto desempeño," *Observatorio Económico*, n.º 166, págs. 14-15, 2 de mayo de 2022, ISSN: 0719-9597, 0719-9597. DOI: 10.11565/oe.vi166.448. visitado 12 de mar. de 2025. dirección: https://www.observatorioeconomico.cl/index.php/oe/article/view/448.

[15] I. Alkhwaja et al., "Password cracking with brute force algorithm and dictionary attack using parallel programming," *Applied Sciences*, vol. 13, n.º 10, pág. 5979, 12 de mayo de 2023, ISSN: 2076-3417. DOI: 10.3390/app13105979. visitado 28 de feb. de 2025. dirección: https://www.mdpi.com/2076-3417/13/10/5979.

[16] M. Guiracocha, "Análisis de factibilidad del uso de GPU para mejorar la eficiencia de los algoritmos de optimización de metaheurísticas," *NOVASINERGIA REVISTA DIGITAL DE CIENCIA, INGENIERÍA Y TECNOLOGÍA*, vol. 6, n.º 1, págs. 50-64, 16 de ene. de 2023, ISSN: 26312654. DOI: 10.37135/ns.01.11.04. visitado 28 de jun. de 2025. dirección: https://novasinergia.unach.edu.ec/index.php/novasinergia/article/view/379.

[17] R. Verma, N. Dhanda y V. Nagar, "Enhancing security with in-depth analysis of brute-force attack on secure hashing algorithms," en *Proceedings of Trends in Electronics and Health Informatics*, M. S. Kaiser, A. Bandyopadhyay, K. Ray, R. Singh y V. Nagar, eds., vol. 376, Singapore: Springer Nature Singapore, 2022, págs. 513-522, ISBN: 9789811688256 9789811688263. DOI: 10.1007/978-981-16-8826-3_44. visitado 21 de mayo de 2025. dirección: https://link.springer.com/10.1007/978-981-16-8826-3_44.

[18] V. Nair y D. Song, "Multi-Factor Credential Hashing for Asymmetric Brute-Force Attack Resistance," en *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*, Delft, Netherlands: IEEE, jul. de 2023, págs. 56-72, ISBN: 9781665465120. DOI: 10.1109/EuroSP57164.2023.00013. visitado 29 de jun. de 2025. dirección: https://ieeexplore.ieee.org/document/10190544/.

[19] R. Beno y R. Poet, "Hacking passwords that satisfy common password policies: Hacking passwords," en *13th International Conference on Security of Information and Networks*, Merkez Turkey: ACM, 4 de nov. de 2020, págs. 1-3, ISBN: 9781450387514. DOI: 10.1145/3433174.3433616. visitado 29 de jun. de 2025. dirección: https://dl.acm.org/doi/10.1145/3433174.3433616.

[20] F. Ciccozzi, L. Addazi, S. A. Asadollah, B. Lisper, A. N. Masud y S. Mubeen, "A comprehensive exploration of languages for parallel computing," *ACM Computing Surveys*, vol. 55, n.º 2, págs. 1-39, 28 de feb. de 2023, ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3485008. visitado 27 de mayo de 2025. dirección: https://dl.acm.org/doi/10.1145/3485008.

[21] M. Jain, A. Sinha, A. Agrawal y N. Yadav, "Cyber security: Current threats, challenges, and prevention methods," en *2022 International Conference on Advances in Computing, Communication and Materials (ICACCM)*, Dehradun, India: IEEE, 10 de nov. de 2022, págs. 1-9, ISBN: 9781665474399. DOI: 10.1109/ICACCM56405.2022.10009154. visitado 24 de mayo de 2025. dirección: https://ieeexplore.ieee.org/document/10009154/.

[22] F. Z. Glory, A. Ul Aftab, O. Tremblay-Savard y N. Mohammed, "Strong Password Generation Based On User Inputs," en *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, Vancouver, BC, Canada: IEEE, oct. de 2019, págs. 0416-0423, ISBN: 9781728125305. DOI: 10.1109/IEMCON.2019.8936178. visitado 24 de mayo de 2025. dirección: https://ieeexplore.ieee.org/document/8936178/.

[23] R. Shay et al., "Encountering stronger password requirements: User attitudes and behaviors," en *Proceedings of the Sixth Symposium on Usable Privacy and Security*, Redmond Washington USA: ACM, 14 de jul. de 2010, págs. 1-20, ISBN: 9781450302647. DOI: 10.1145/1837110.1837113. visitado 4 de jun. de 2025. dirección: https://dl.acm.org/doi/10.1145/1837110.1837113.

[24] J. Hallett, N. Patnaik, B. Shreeve y A. Rashid, ""Do this! Do that!, and Nothing will Happen" Do Specifications Lead to Securely Stored Passwords?" En *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, Madrid, ES: IEEE, mayo de 2021, págs. 486-498, ISBN: 9781665402965. DOI: 10.1109/ICSE43902.2021.00053. visitado 30 de jun. de 2025. dirección: https://ieeexplore.ieee.org/document/9402024/.

[25] H. Hussain, "Password security: Best practices and management strategies," *SSRN Electronic Journal*, 2022, ISSN: 1556-5068. DOI: 10.2139/ssrn.4136333. visitado 29 de jun. de 2025. dirección: https://www.ssrn.com/abstract=4136333.

[26] A. Calder, *Implementing information security based on ISO 27001/ISO 27002*. Van Haren, 2020.

[27] W. Y. Saputra, S. Sugiarti, H. Junianto y D. Suhartono, "PASSWORD STRENGTH STUDY USING THE ZXCVBN ALGORITHM AND BRUTE-FORCE TIME ESTIMATION TO STRENGTHEN CYBERSECURITY," *Jurnal Pilar Nusa Mandiri*, vol. 21, n.º 1, págs. 52-59, 14 de mar. de 2025, ISSN: 2527-6514, 1978-1946. DOI: 10.33480/pilar.v21i1.6119. visitado 9 de jun. de 2025. dirección: https://ejournal.nusamandiri.ac.id/index.php/pilar/article/view/6119.

[28] L. Ertaul, M. Kaur y V. A. K. R. Gudise, "Implementation and performance analysis of pbkdf2, bcrypt, scrypt algorithms," en *Proceedings of the international conference on wireless networks (ICWN)*, The Steering Committee of The World Congress in Computer Science, Computer …, 2016, pág. 66.

[29] N. Alaa y F. Al-Shareefi, "A Comparative Study Between Two Cybersecurity Attacks: Brute Force and Dictionary Attacks," *Journal of Kufa for Mathematics and Computer*, vol. 11, n.º 2, págs. 133-139, 31 de ago. de 2024, ISSN: 2518-0010, 2076-1171. DOI: 10.31642/JoKMC/2018/110216. visitado 7 de jul. de 2025. dirección: https://journal.uokufa.edu.iq/index.php/jkmc/article/view/15772.

[30] Laatansa, R. Saputra y B. Noranita, "Analysis of GPGPU-Based Brute-Force and Dictionary Attack on SHA-1 Password Hash," en *2019 3rd International Conference on Informatics and Computational Sciences (ICICoS)*, Semarang, Indonesia: IEEE, oct. de 2019, págs. 1-4, ISBN: 9781728146102. DOI: 10.1109/ICICoS48119.2019.8982390. visitado 26 de mayo de 2025. dirección: https://ieeexplore.ieee.org/document/8982390/.

[31] M. A. S. Netto, R. N. Calheiros, E. R. Rodrigues, R. L. F. Cunha y R. Buyya, "HPC cloud for scientific and business applications: Taxonomy, vision, and research challenges," *ACM Computing Surveys*, vol. 51, n.º 1, págs. 1-29, 31 de ene. de 2019, ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3150224. visitado 28 de mayo de 2025. dirección: https://dl.acm.org/doi/10.1145/3150224.

[32] D. Khaldi, P. Jouvelot, C. Ancourt y F. Irigoin, "Task Parallelism and Data Distribution: An Overview of Explicit Parallel Programming Languages," en *Languages and Compi-*

lers for Parallel Computing, H. Kasahara y K. Kimura, eds., red. de D. Hutchison et al., vol. 7760, Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, págs. 174-189, ISBN: 9783642376573 9783642376580. DOI: 10.1007/978-3-642-37658-0_12. visitado 27 de mayo de 2025. dirección: http://link.springer.com/10.1007/978-3-642-37658-0_12.

[33] Centro de Investigación en Matemáticas A.C. et al., "Tendencias y Desafíos de la Computación de Alto Rendimiento en la Nube," *RISTI - Revista Ibérica de Sistemas e Tecnologias de Informação*, n.º 49, págs. 131-146, 31 de mayo de 2023, ISSN: 16469895. DOI: 10.17013/risti.49.131-146. visitado 28 de mayo de 2025. dirección: http://scielo.pt/scielo.php?script=sci_arttext&pid=S1646-98952023000100131&lng=pt&nrm=iso&tlng=es.

[34] Kaggle, *Strong Passwords in RockYou2024.txt*, [Online]. Disponible en: https://www.kaggle.com/datasets/bwandowando/strong-passwords-in-rockyou2024-txt. [Consultado: 18 de oct. 2025], 2024. dirección: https://www.kaggle.com/datasets/bwandowando/strong-passwords-in-rockyou2024-txt.

[35] C. CUDA, "What is CUDA?," 2022.

[36] B. J. Svensson, K. Claessen y M. Sheeran, "GPGPU Kernel Implementation using an Embedded Language: a Status Report," ene. de 2010.

[37] M. Jay, V. Ostapenco, L. Lefèvre, D. Trystram, A.-C. Orgerie y B. Fichel, "An experimental comparison of software-based power meters: focus on CPU and GPU," en *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, IEEE, 2023, págs. 106-118.

[38] P. S. Akulapally, "GPU Exploration using PyCuda," 2024.

[39] N. Rao, N. Liebers, A. S. Leger y S. J. Matthews, "Comparing the Performance of Numba and CUDA for Historical Analysis of Synchrophasor Data," en *2024 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT)*, IEEE, 2024, págs. 1-5.

[40] L. de Moore, “Ley de Amdahl,” Tesis doct., Universidad de Los Andes, Mérida 5101 Venezuela.

ANEXOS

4.5 Anexo A — Resultados de pruebas

https://utneduec-my.sharepoint.com/:f:/g/personal/tdpujotar_utn_edu_ec/IgBCr7aqcsTQS5ZPJZqjIDpNAf8gUOP2PoCYPpRvDR6rwjw?e=2beCKc