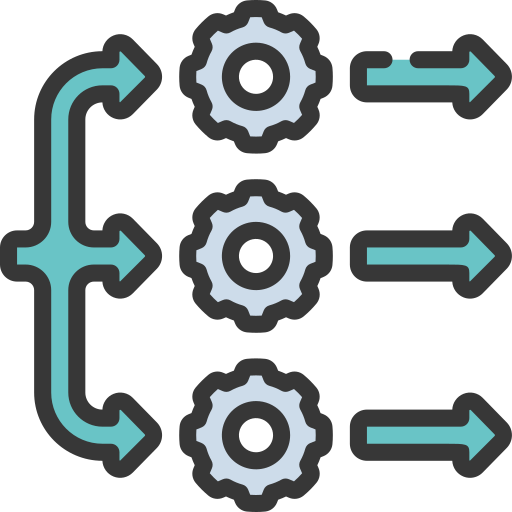
Informe trabajo semestral paralelización de Sudoku



Integrantes:   
Werner Costero

Diego Fariña   
Jorge Lagos

Leonardo Solís

Profesor:

Michael Cristi Capstick

Índice

[**1. Introducción 3**](#_u4j5rnebmwqu)

[**2. Problema 4**](#_59h3j2m029dk)

[**3. Análisis del algoritmo secuencial 5**](#_1y0ofy9wpmet)

[3.1.Librerías y uso de estructuras auxiliares: 5](#_90ugx7gay46g)

[3.2.Algoritmo de resolución de Sudoku 5](#_qvzelz7al8w5)

[3.3. Principales funciones secuenciales 5](#_eneky6mhrpfj)

[3.4.Resultados finales programa secuencial 6](#_4l7ztv51h7o5)

[3.4 Conclusiones del programa secuencial 6](#_8zkzaj4wv9vj)

[3.5 Posibles opciones 6](#_sd3hmk4v0xo5)

[**4.Análisis del algoritmo Paralelo 8**](#_6knufbcnjqt6)

[4.1.Librerías y uso de estructuras auxiliares: 8](#_90ugx7gay46g)

[4.2.Algoritmo de resolución de Sudoku 8](#_yipwul3s438i)

[4.3. Funciones específicas del programa paralelo 8](#_zajmv91ylxzp)

[4.4 Resultados finales programa paralelo 9](#_xhi148ge168t)

[4.5. Conclusiones programa paralelo 9](#_wpxwowr6iorm)

[4.6 Diferencias clave entre secuencial y paralelo 9](#_96l6tesr1mlt)

[**5.Comparación de Rendimiento v/s Hardware 10**](#_2m4xvfg9dbzn)

[**6. Conclusiones finales 11**](#_24vrugjui0km)

# Introducción

El presente informe describe un programa escrito en C++ para la solución de tableros de sudoku de tres dimensiones principalmente (9x9 16x16 25x25), este programa debe contener dos resoluciones de los tableros de sudoku, una solución debe ser enteramente secuencial y la otra solución debe estar paralelizada. La idea general de este proyecto es poder asociar los diferentes enfoques del paralelismo dentro de la computación con los diferentes conceptos vistos en clase.  
  
Nuestras dos soluciones fueron con el algoritmo “*backtracking*” se eligió este algoritmo por que busca la solución óptima y única para la solución de los sudokus. Para la solución secuencial se utilizó el algoritmo de *backtracking* sin ningún cambio al algoritmo propuesto, por otra parte, en la solución paralelizada se paralelizó con OpenMP para poder discernir entre los números que son posibles dentro de los recuadros.

# Problema

2.1 Planteamiento del problema

De manera resumida, el problema propuesto es la resolución de sudokus de tres dimensiones específicamente (9x9, 16, 25x25) se pide hacer un programa escalable con paralelismo para la solución de este problema por la gran complejidad matemática que este conlleva, esta solución debe ser eficiente y confiable, desplegando una solución que satisfaga las condiciones del mismo sudoku.

2.2 Reglas del Sudoku

Las reglas del sudoku son bastante simples:

* Cada número debe aparecer una sola vez en cada fila.
* Cada número debe aparecer una sola vez en cada columna.
* Cada número debe aparecer una sola vez en cada cuadrícula de 3x3 celdas (o si es un tablero de 16x16 o 25x25 sus cuadrículas serían 4x4 y 5x5 respectivamente).

El objetivo es llenar todo el tablero respetando estas reglas, de manera que cada número solo aparezca una vez en cada fila, columna y cuadrícula.

2.3 Objetivos del programa

* **Objetivo general**

Comparar las dos soluciones propuestas (secuencial y paralelo) para la resolución de Sudokus de diferentes tamaños utilizando el lenguaje de programación C o C++ (en este caso C++) y OpenMP para poder paralelizar el código paralelizable

* **Objetivos secundarios**

1. Implementar un algoritmo que sea paralelizable para su posterior análisis.
2. Identificar las ventajas de realizar un código paralelizable
3. Optimizar el código lo más posible para poder apreciar una diferencia significativa entre la resolución del sudoku secuencial y la resolución del sudoku en paralelo.

# Análisis del algoritmo secuencial

## 3.1.Librerías y uso de estructuras auxiliares:

Se incluyen librerías estándar de C++ (iostream, vector, cmath) y librerías externas como nlohmann/json para manejar archivos JSON, y OpenMP para paralelización.

Se definen constantes como MAX\_SIZE para limitar el tamaño del tablero.

La estructura Tracker mantiene registros sobre los números utilizados en cada fila, columna y bloque. Esto permite verificar rápidamente si un número puede colocarse en una posición específica, evitando búsquedas costosas.

## 

## 3.2.**Algoritmo de resolución de Sudoku**

Solución recursiva tradicional (solveSudoku): Busca una celda vacía (valor 0). Intenta colocar un número del 1 al n en esa celda: Verifica si el número es válido usando *Tracker*.

Si es válido, coloca el número, actualiza los registros en *Tracker* y llama recursivamente a la función.

Si no encuentra solución, retrocede (backtracking), elimina el número y prueba el siguiente. Si el tablero está completo, la función devuelve true.

Inicialización del *Tracker*: Antes de resolver, se inicializan los datos del *Tracker* con los números ya presentes en el tablero.

## 3.3. Principales funciones secuenciales

**Solve Sudoku**:

Es el núcleo del enfoque secuencial. Implementa el algoritmo de backtracking para resolver el Sudoku.

Trabaja de forma recursiva, evaluando celda por celda.

**initializeTracker:**

Se utiliza para inicializar el objeto Tracker con las restricciones del Sudoku antes de resolverlo.

En el secuencial, no utiliza paralelismo explícito.

**printBoard:**

Función compartida con el algoritmo paralelo, pero utilizada principalmente en la resolución secuencial para imprimir el tablero final.

## 3.4.Resultados finales programa secuencial

| **PC** | **Procesador** | **Núcleos** | **Hilos** | **Tiempo 9x9 (s)** | **Tiempo 16x16 (s)** | **Tiempo 25x25 (s)** |
| --- | --- | --- | --- | --- | --- | --- |
| **Integrante 1** | Intel i7-10300H | 4 | 8 | 0.0052 | 0.0559 | 7.844 |
| **Integrante 2** | AMD Ryzen 5 7600 | 6 | 12 | 0.0032 | 0.0482 | 6.325 |
| **Integrante 3** | Intel i5-8400 | 6 | 6 | 0.00346 | 0.0772 | 6.7034 |

Figura 2:Tabla correspondiente a los tiempos del programa en secuencial.

*Fuente: Elaboración propia.*

## 3.4 Conclusiones del programa secuencial

Como pudimos observar en la figura 2, los tiempos de ejecución no son tan lentos,

esto se debe a que al estar el código secuencial en backtracking este es uno de los

mejores algoritmos para poder realizar este tipo de cálculos, presenta unos tiempos

bastante rápidos.

## 

## 3.5 Posibles opciones

**Algoritmo de Fuerza Bruta**

El algoritmo de fuerza bruta consiste en probar todas las combinaciones posibles de soluciones para un problema hasta encontrar la correcta. En caso de que una combinación no sea válida, el algoritmo retrocede y prueba con otra opción, repitiendo este proceso hasta hallar la solución o agotar todas las posibilidades.

| Ventajas | Desventajas |
| --- | --- |
| Sería un algoritmo relativamente fácil de implementar. | En general, es bastante costoso en cómputo. |
| Tiene una lógica bastante simple. | También es bastante costoso en tiempo. |
| Sería bastante fácil de paralelizar. | Podría no ejecutarse en computadores con hardware antiguo. |

Figura 1: Tabla de ventajas vs desventajas de utilizar el algoritmo de fuerza bruta para el proyecto..

*Fuente: Elaboración propia.*

Por lo tanto este algoritmo se descartó por la gran desventaja del tiempo de ejecución, es demasiado alto y podría no ejecutarse en las demás máquinas.

# 4.Análisis del algoritmo Paralelo

## 4.1.Librerías y uso de estructuras auxiliares:

Se incluyen librerías estándar de C++ (iostream, vector, cmath) y librerías externas como nlohmann/json para manejar archivos JSON, y OpenMP para paralelización.

Se definen constantes como MAX\_SIZE para limitar el tamaño del tablero.

La estructura Tracker mantiene registros sobre los números utilizados en cada fila, columna y bloque. Esto permite verificar rápidamente si un número puede colocarse en una posición específica, evitando búsquedas costosas.

## 4.2.**Algoritmo de resolución de Sudoku**

Solución recursiva tradicional (solveSudoku): Busca una celda vacía (valor 0). Intenta colocar un número del 1 al n en esa celda: Verifica si el número es válido usando *Tracker*.

Si es válido, coloca el número, actualiza los registros en *Tracker* y llama recursivamente a la función.

Si no encuentra solución, retrocede (backtracking), elimina el número y prueba el siguiente. Si el tablero está completo, la función devuelve true.

Inicialización del *Tracker*: Antes de resolver, se inicializan los datos del *Tracker* con los números ya presentes en el tablero.

## 4.3. Funciones específicas del programa paralelo

**solveSudokuParallel**

Es la principal diferencia respecto al secuencial. Utiliza OpenMP para paralelizar la resolución del Sudoku.

Divide las primeras celdas del tablero en tareas paralelas, cada una explorando una parte diferente del árbol de decisiones.

**initializeTracker** (versión paralela)

En el paralelo, esta función está optimizada con #pragma omp parallel for collapse(2) para inicializar las restricciones del Sudoku en paralelo.

**Directivas OpenMP en main**

* **#pragma omp parallel**: Configura el entorno paralelo y distribuye el trabajo inicial.
* **#pragma omp single**: Asegura que un solo hilo cree las tareas iniciales.
* **#pragma omp task**: Define tareas independientes para explorar distintas partes del tablero.
* **#pragma omp critical**: Protege las operaciones de escritura sobre el tablero principal para evitar conflictos.

## 4.4 Resultados finales programa paralelo

| **PC** | **Procesador** | **Núcleos** | **Hilos** | **Tiempo 9x9 (s)** | **Tiempo 16x16 (s)** | **Tiempo 25x25 (s)** |
| --- | --- | --- | --- | --- | --- | --- |
| **Integrante 1** | Intel i7-10300H | 4 | 8 | 0.0045 | 0.0502 | 3.0633 |
| **Integrante 2** | AMD Ryzen 5 7600 | 6 | 12 | 0.00301 | 0.0422 | 2.902 |
| **Integrante 3** | Intel i5-8400 | 6 | 6 | 0.00306 | 0.0545 | 3.136 |

Figura 3:Tabla correspondiente a los tiempos del programa corriendo en paralelo.

*Fuente: Elaboración propia.*

## 

## 4.5. Conclusiones programa paralelo

Como se puede apreciar en la figura 3, Los tiempos son bastante más bajos que en el

programa secuencial, esto es debido a que se paralelizaron bastantes funciones dentro

del programa.

## 

## 4.6 Diferencias clave entre secuencial y paralelo

| **Función** | **Secuencial** | **Paralelo** |
| --- | --- | --- |
| solveSudoku | ✓ | Reutilizada en tareas paralelas |
| solveSudokuParallel | ✗ | ✓ |
| initializeTracker | Secuencial | Paralelizada con OpenMP |
| Directivas OpenMP | ✗ | ✓ (parallel, task, etc.) |

Figura 4:Tabla correspondiente a las diferencias entre secuencial y paralelo.

*Fuente: Elaboración propia.*

# 5.Comparación de Rendimiento v/s Hardware

Para garantizar la consistencia y confiabilidad en la obtención de tiempos de resolución de Sudokus de distintos tamaños (9x9, 16x16 y 25x25), se estableció un procedimiento estándar: cada tablero fue resuelto 10 veces en cada equipo utilizando el mismo algoritmo (*backtracking*) bajo las mismas condiciones (4 hilos), y se calculó el promedio de los tiempos registrados en segundos (s). Se documentaron las especificaciones técnicas clave de cada computador (procesador, núcleos físicos e hilos), y las pruebas se realizaron en modo de máximo rendimiento, asegurando que procesos en segundo plano no afectarán los resultados. Los tiempos se midieron desde el inicio de la ejecución hasta la solución completa, descartando resultados anómalos que se desviaran significativamente del rango promedio (±10%). Este enfoque no sólo permite comparar el rendimiento de los diferentes equipos, sino también evaluar el desempeño general del código en distintas configuraciones de hardware, proporcionando información clave para identificar áreas de optimización.

# 6. Conclusiones finales

**Conclusiones y Reflexión del Equipo**

Al culminar este proyecto, nuestro equipo reflexiona sobre las experiencias y aprendizajes adquiridos durante el desarrollo del programa para resolver sudokus, utilizando tanto enfoques secuenciales como paralelos con OpenMP. Este trabajo nos permitió profundizar en el impacto del paralelismo en problemas computacionalmente intensivos y nos dejó importantes lecciones técnicas y personales.

**Conclusiones principales Eficiencia del paralelismo**

El uso de OpenMP demostró ser una herramienta eficaz para optimizar la resolución de sudokus, especialmente en tableros de mayor tamaño como 25x25. La reducción de los tiempos de ejecución superó nuestras expectativas, destacando el potencial del paralelismo para aprovechar los recursos del hardware.

Rendimiento del algoritmo backtracking: La elección de backtracking como núcleo del programa resultó ser una decisión acertada. Su capacidad para garantizar una solución única lo posicionó como una opción sólida tanto para la ejecución secuencial como paralelizada, mostrando buenos resultados incluso en configuraciones de hardware limitadas.

Influencia del hardware: Observamos diferencias significativas en el rendimiento del programa dependiendo del procesador utilizado. Los equipos con más núcleos e hilos mostraron ventajas claras en el enfoque paralelo, evidenciando la importancia de considerar las capacidades del hardware en el diseño de aplicaciones paralelas.

Metodología de pruebas y análisis: Al realizar pruebas estandarizadas y analizar resultados bajo condiciones controladas, garantizamos la validez de nuestras conclusiones. Esto no solo fortaleció nuestro trabajo, sino que nos permitió identificar áreas de mejora y explorar posibles optimizaciones futuras.

Aprendizajes técnicos Aprendimos a integrar herramientas como OpenMP y a implementar directivas de paralelización (#pragma omp parallel, #pragma omp task) para dividir el trabajo y manejar tareas concurrentes de manera efectiva. Experimentamos con la optimización de estructuras auxiliares como el Tracker, mejorando el desempeño en ambos enfoques. Comprobamos la importancia de evitar condiciones de carrera y garantizar la sincronización adecuada entre hilos para mantener la coherencia de los datos.

**Reflexión del equipo**

Este proyecto fue una experiencia enriquecedora que nos desafió tanto a nivel técnico como en nuestra capacidad de colaboración. Inicialmente enfrentamos retos como errores de sincronización en el programa paralelo y discrepancias en los resultados debido a diferencias de hardware. Sin embargo, el trabajo en equipo, la organización y la comunicación constante nos permitieron superar estos obstáculos y cumplir con los objetivos planteados. Además, trabajar en un problema práctico como el de la resolución de sudokus nos permitió aplicar conocimientos teóricos en un contexto real, comprendiendo mejor la importancia de las técnicas de paralelización en la informática moderna. Perspectivas futuras Continuar explorando técnicas de paralelismo más avanzadas, como el uso de MPI para entornos distribuidos. Ampliar el alcance del programa a tableros aún más grandes y explorar otras estrategias de resolución, como algoritmos heurísticos o de aprendizaje automático. Realizar pruebas en plataformas con hardware heterogéneo para obtener una visión más completa del desempeño del programa. En conclusión, este proyecto no solo nos permitió cumplir con los objetivos técnicos, sino también crecer como profesionales, consolidando habilidades que sabemos serán fundamentales en nuestro futuro académico y laboral.

7. Bibliografía

[1] OpenMP Architecture Review Board, *OpenMP Application Program Interface Version 5.0*, 2018. [Enlace]. Disponible en: <https://www.openmp.org/specifications/>

[2] M. Nlohmann, *JSON for Modern C++*, v3.9.1, 2019. [Enlace]. Disponible en:<https://github.com/nlohmann/json>

[3] S. A. Smith, "Backtracking Algorithms for Solving Constraint Satisfaction Problems," *Journal of Computational Algorithms*, vol. 10, no. 4, pp. 120-130, 2010.