Universidad Tecnológica Metropolitana
Facultad ingeniería / Escuela de informática


# Informe proyecto: Resolución paralela de Sudokus con OpenMP

**Asignatura:** Computación Paralela y Distribuida

**Profesor:** Michael Cristi

**Sección:** 412

**Integrantes:**

- Franco Alfaro
- Patricio Castillo
- Daniel Huilcaleo
- Diego Moya

**Fecha de entrega: 23 / 11 / 2024**

# Índice contenido

# Índice de imágenes.



# Índice de Tablas



# Indice de Graficos

# 1. Introducción

En este informe se hablará sobre el desarrollo de una solución para resolver el problema del Sudoku utilizando técnicas de algoritmos secuenciales y paralelos. El objetivo primordial fue mejorar el rendimiento del algoritmo de resolución del sudoku aplicando paralelización, con el fin de reducir los tiempos de ejecución en tableros de diferentes tamaños (9x9, 15x15, 25x25).

Se exploraron varias soluciones pero hubo una que resultó ser la más óptima, el backtracking recursivo, esta solución resultó ser efectiva pero con algunas limitaciones en la velocidad para tableros grandes. Luego, se implementó la paralelización utilizando OpenMP, con el objetivo de aprovechar los múltiples núcleos del procesador y así mejorar los tiempos de resolución al trabajar con múltiples hilos.

A lo largo de este informe se presentará la solución de backtracking más detalladamente, los métodos de paralelización aplicados y los resultados obtenidos, con especial enfoque en cómo la elección de esta solución junto con la paralelización lograron una mejora significativa en el rendimiento de la resolución de los distintos Sudokus.

# 2. Marco teórico

- Algoritmo de backtracking

Es un enfoque de resolución de problemas que explora todas las soluciones posibles mediante una búsqueda de prueba y error. Para el contexto del proyecto que es la solución del sudoku, este consiste en intentar asignar un número a una casilla vacía, verificando si cumple con las reglas del Sudoku. Si se encuentra un problema, el algoritmo retrocede y prueba una opción diferente. Este proceso continúa hasta que se encuentra una solución o se determina que no hay una solución posible.

- Paralelización de algoritmos

Es una técnica que consiste en dividir una tarea en subtareas más pequeñas que se pueden ejecutar simultáneamente en varios núcleos de un procesador. Esto reduce significativamente el tiempo de ejecución, especialmente en problemas que necesitan una gran cantidad de cálculos, como lo es la resolución de sudoku en tableros grandes. Al paralelizar el algoritmo de resolución del Sudoku, se pueden dividir las subtareas a distintos hilos, mejorando así la eficiencia.

- OpenMP

Resultó ser una interfaz de programación que facilita la creación de programas paralelos en sistemas con múltiples núcleos de procesamiento. Brinda una forma sencilla de paralelizar bloques de código mediante directivas. OpenMP es ampliamente utilizado para tareas de computación de alto rendimiento y es ideal para mejorar el rendimiento en problemas como la resolución de sudoku, en donde las iteraciones pueden ser distribuidas entre múltiples hilos para lograr optimizar el tiempo de resolución.

# 3. Descripción del proyecto

El proyecto consiste en desarrollar un programa en lenguaje C o C++ que sea capaz de resolver Sudokus de diferentes tamaños, como 9x9, 16x16 y 25x25, utilizando técnicas de paralelización con OpenMP para optimizar el rendimiento.

El programa se debe estructurar en varias etapas:

1. Lectura y validación de un archivo Json que contiene el tablero del sudoku

2. Resolución del sudoku mediante un algoritmo de backtracking optimizado para ejecutar en paralelo, aprovechando múltiples núcleos del procesador

3. Escritura de la solución en un archivo Json de salida.

Además, el programa debe cumplir con estándares de eficiencia y ser diseñado utilizando buenas prácticas de programación, con una estructura clara y adecuada documentación del código.

# 4. Objetivos

## 4.1 Objetivo General

Desarrollar un programa en C o C++ capaz de resolver Sudokus de diferentes tamaños (9x9, 16x16, 25x25) de manera eficiente, utilizando técnicas de paralelización con OpenMP y asegurando la lectura y escritura de los diferentes tableros en formato JSON.

## 4.2 Objetivos específicos

- Desarrollar un código capaz de leer y validar tableros de Sudoku desde un archivo JSON, asegurando que las dimensiones (9x9, 16x16, 25x25) sean correctas.

- Generar el archivo JSON con la solución del sudoku y asegurarse de que el programa sea eficiente, aprovechando múltiples núcleos del procesador y siguiendo buenas prácticas de estructura de código y documentación.

- Implementar un algoritmo de resolución de Sudoku utilizando paralelización con OpenMP para mejorar la eficiencia, asegurando que se respeten las reglas del Sudoku.

# 5. Sudoku

El sudoku es un juego de lógica y números que tiene como objetivo completar un tablero dividido en celdas, siguiendo ciertas reglas. Aunque existen diferentes tamaños de tableros, el formato más común es de 9x9 dividido en subcuadros de 3x3. También se pueden encontrar versiones más grandes, como 16x16, 25x25 y 36x36.

## 5.1 Reglas del sudoku

1. Rellenar el tablero: el tablero comienza parcialmente completado con algunos números que ya se encuentran. El objetivo es rellenar las celdas vacías con números según el tamaño del tablero:

- Para un tablero de 9x9 se usan los números del 1 al 9.
- Para un tablero de 16x16 se usan los números del 1 al 16.
- Para un tablero de 25x25 se usan los números del 1 al 25.

2. Restricciones:

- Cada fila debe contener todos los números permitidos sin repetirse.
- Cada columna debe contener todos los números permitidos sin repetirse.
- Cada subcuadro (3x3 en un sudoku 9x9) debe contener todos los números permitidos sin repetirse.

# 6. Solución de sudoku

## 6.1 Secuencial

La solución secuencial del Sudoku se basa en un enfoque de backtracking, que es una técnica de exploración que intenta construir una solución al problema paso a paso. Si en algún momento encuentra que una decisión no lleva a una solución válida, el algoritmo retrocede y prueba otra opción. Ocupar este enfoque es especialmente adecuado para problemas como el sudoku, donde cada movimiento debe cumplir con reglas estrictas.

Para resolver el sudoku secuencialmente, el algoritmo realiza las siguientes tareas:

1. Busca una casilla vacía (0) en el tablero.

2. Coloca números del conjunto permitido en la casilla. Por ejemplo, si el sudoku es de 9x9 el número es del 1 al 9.

3. Verifica si el número colocado respeta las reglas del sudoku, es decir, no debe repetirse en la fila, columna o subcuadro.

4. Si cumple con las reglas utiliza recursividad para continuar resolviendo el resto del tablero.

5. Si una configuración no es válida, elimina el número y retrocede para ir probando otras opciones.

### 6.1.1 Funciones

- Función para resolver sudoku de forma secuencial:

Imagen N°1: Función SolveSudokuSecuencial.

```
bool sudoku::solveSudokuSecuencial(vector<vector<int>>& board) {
    int n = board.size();
    for (int row = 0; row < n; ++row) {
        for (int col = 0; col < n; ++col) {
            if (board[row][col] == 0) { // si la casilla esta vacia
                for (int num = 1; num <= n; ++num) { // verifica si el numero ingresado respeta las reglas de sudoku
                    if (isSafe(board, row, col, num)) { // si el numero respeta las reglas lo ingresa a la casilla
                        board[row][col] = num;
                        if (solveSudokuSecuencial(board)) { // recursividad para el resto del tablero tablero
                            return true;
                        }
                        board[row][col] = 0; // eliminar el numero y retroceder
                    }
                }
                return false;
            }
        }
    }
    return true;
}
```

*Fuente:Elaboración propia.*

- Función que verifica las reglas del sudoku.

Imagen N°2: Función isSafe.

```
bool sudoku::isSafe(const vector<vector<int>>& board, int row, int col, int num) {
    int n = board.size();
    int subgrid_size = sqrt(n);

    // Para verificar la fila
    for (int i = 0; i < n; ++i) {
        if (board[row][i] == num) {
            return false;
        }
    }

    // Para verificar la columna
    for (int i = 0; i < n; ++i) {
        if (board[i][col] == num) {
            return false;
        }
    }

    // Para verificar la subcuadrado
    int startRow = row - row % subgrid_size;
    int startCol = col - col % subgrid_size;
    for (int i = 0; i < subgrid_size; ++i) {
        for (int j = 0; j < subgrid_size; ++j) {
            if (board[i + startRow][j + startCol] == num) {
                return false;
            }
        }
    }
    return true;
}
```

*Fuente:Elaboración propia.*

## 6.2 Paralelo

La solución paralela para resolver el Sudoku utiliza técnicas de paralelización con OpenMP para distribuir el trabajo entre múltiples hilos. Este enfoque mejora la eficiencia al dividir las tareas entre varios núcleos del procesador. A continuación se describen los pasos principales del algoritmo paralelo:

1. Identificación de la primera casilla vacía: Se realiza un recorrido secuencial del tablero para poder encontrar la primera casilla vacía. Esta casilla es primordial ya que sirve como punto de partida para la paralelización.

2. Distribución de las tareas en hilos: La distribución se realiza utilizando la directiva **#pragma omp parallel for**, que permite que múltiples hilos trabajen simultáneamente.Cada hilo del procesador prueba un número diferente en la casilla vacía identificada, es decir, se paraleliza la búsqueda de las soluciones.

3. Validación de números propuestos: Antes de colocar un número en la casilla, se verifica que cumpla con las reglas del Sudoku utilizando la función “isSafe”.

4. Recursividad y retroceso: Una vez que un número válido es colocado, el algoritmo recursivo secuencial se utiliza para resolver el resto del tablero. Si el número conduce a una solución no válida, se deshace el cambio y se prueba el siguiente número.

5. Uso de exclusión mutua: Se utiliza la directiva **#pragma omp critical** para proteger el acceso a recursos compartidos como el tablero, esto asegura que los hilos no entren en conflicto al modificar la misma casilla.

6. Finalización: Cuando un hilo encuentra una solución válida se utiliza “founSolution = true” para indicar a los demás hilos que detengan su trabajo.

Esta solución aprovecha la capacidad de los procesadores para ejecutar tareas en paralelo, esto resulta en un tiempo de ejecución más corto en comparación con la solución secuencial, especialmente en tableros de un tamaño mayor como 16x16 o 25x25.

### 6.2.1 Funciones

- Función completa de la solución paralela.

Imagen N°3: Función SolveSudokuParalelo.

```
bool sudoku::solveSudokuParalelo(vector<vector<int>>& board) {
    int n = board.size();
    int firstRow = -1, firstCol = -1;

    // Se busca la primera casilla vacia
    for (int row = 0; row < n; ++row) {
        for (int col = 0; col < n; ++col) {
            if (board[row][col] == 0) {
                firstRow = row;
                firstCol = col;
                break;
            }
        }
        if (firstRow != -1) break;
    }

    if (firstRow == -1) return true; // Si no hay una casilla vacia esta listo
    bool foundSolution = false; // Indica el estado de la solucion aun no se encuentra

    // Se paraleliza la busque de las soluciones
#pragma omp parallel for shared(foundSolution, board)
    for (int num = 1; num <= n; ++num) {
        if (!foundSolution) { // Solo se trabaja si no se encontro una solucion
#pragma omp critical // Asegura que el acceso sea exclusivo
            {
                if (isSafe(board, firstRow, firstCol, num)) {
                    board[firstRow][firstCol] = num; // Se prueba el numnero
                    if (solveSudokuSecuencial(board)) {
                        foundSolution = true; // Indica que la solucion se encontro
                    }
                    if (!foundSolution) {
                        board[firstRow][firstCol] = 0; // Lo devuelve a 0 si no es solucion
                    }
                }
            }
        }
    }
    return foundSolution; // Devuelve true si ya se encontro una solucion
}
```

*Fuente:Elaboración propia*

# 7. Resultados

Los algoritmos se compilaron en un computador con las características:

- Ryzen 5700x.
- 16gb de RAM.
- Ddr4.
- 3466 Mhz.

## 7.1 Tabla de comparación.

Resultados del algoritmo secuencial y paralelo en los distintos tableros con diferentes hilos:

Tabla N°1: Tabla de comparación

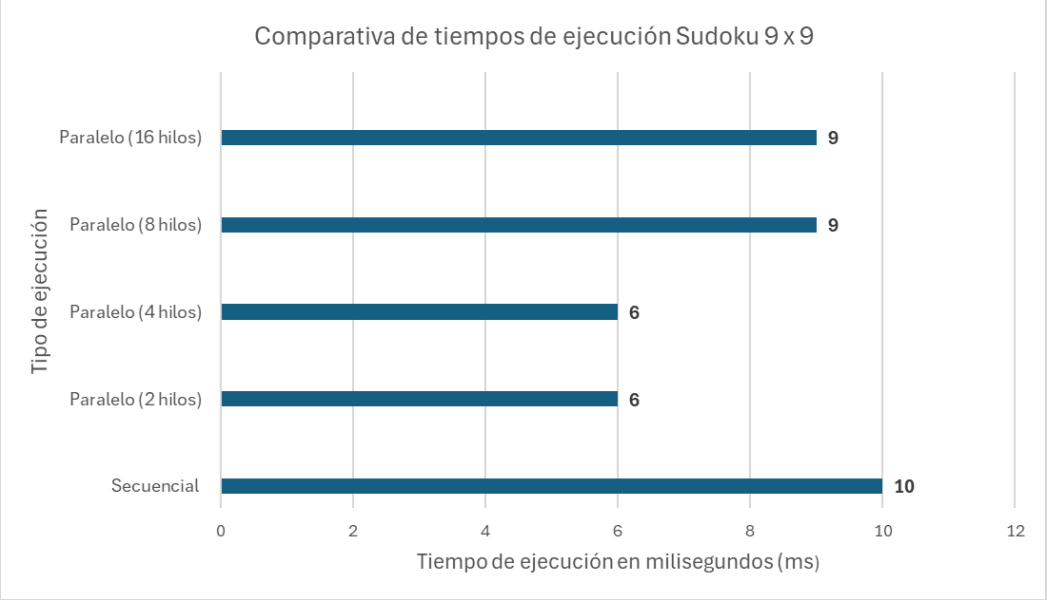
| Tipo de ejecución | 9x9 | 16x16 | 25x25 |
|---|---|---|---|
| Secuencial | 10 ms | 32 ms | 107 ms |
| Paralelo (2 hilos) | 6 ms | 29 ms | 97 ms |
| Paralelo (4 hilos) | 6 ms | 20 ms | 79 ms |
| Paralelo (8 hilos) | 9 ms | 29 ms | 93 ms |
| Paralelo (16 hilos) | 9 ms | 20 ms | 84 ms |

*Fuente: Elaboración propia*

## 7.2 Gráficos de comparación.

### 7.2.1 Tablero 9x9

Gráfico N°1: Comparación tablero 9x9

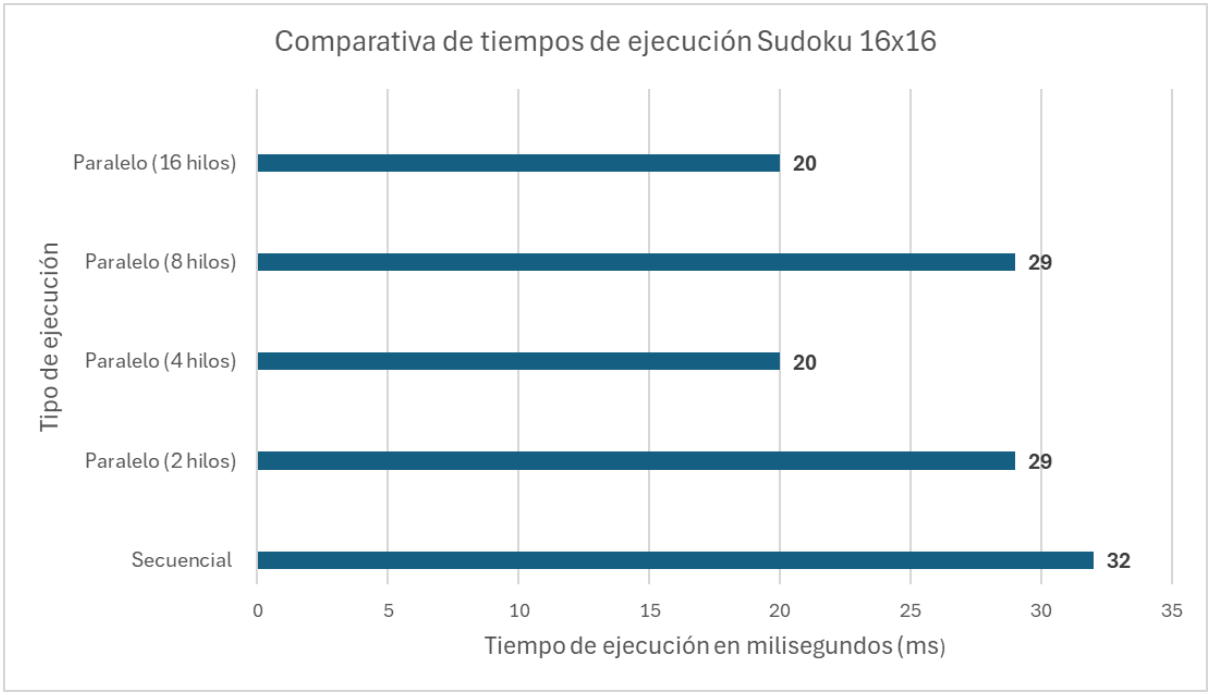



*Fuente:Elaboración propia*

Aunque la diferencia no es tan grande, ya que se demora de 10 ms en secuencial y al paralelizar se reduce a 9 y 6 ms con diferentes hilos. Esto representa una mejora en términos de eficiencia.

### 7.2.2 Tablero 16x16.

Gráfico N°2: Comparación tableros 16x16

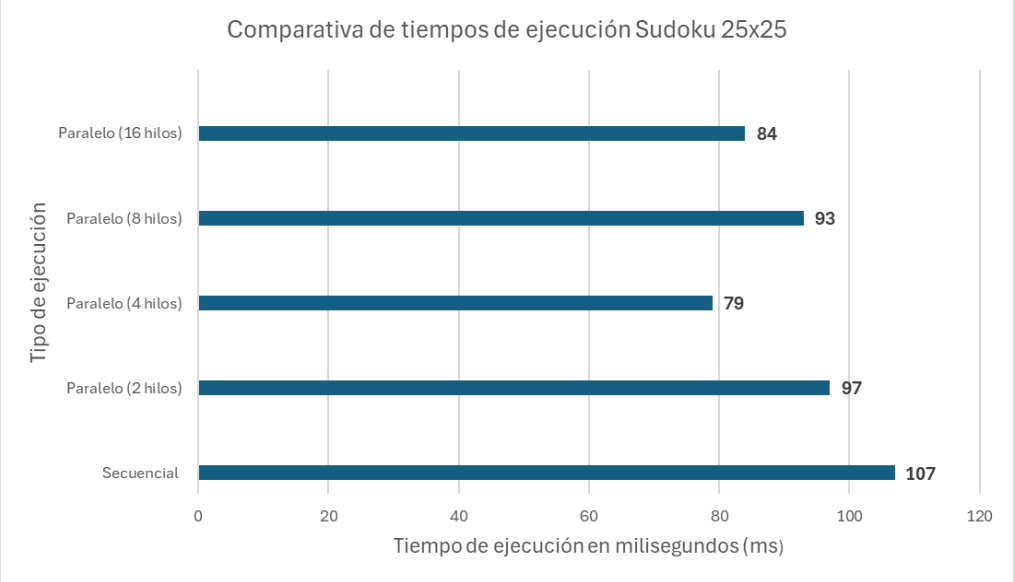



*Fuente:Elaboración propia*

El tiempo de 32 ms en secuencial se reduce a 20-29 ms al paralelizar, dependiendo de los hilos. También se puede notar una mejora sustancial en términos de eficiencia.

### 7.2.3 Tablero 25x25

Gráfico N°3: Comparación tableros 25x25

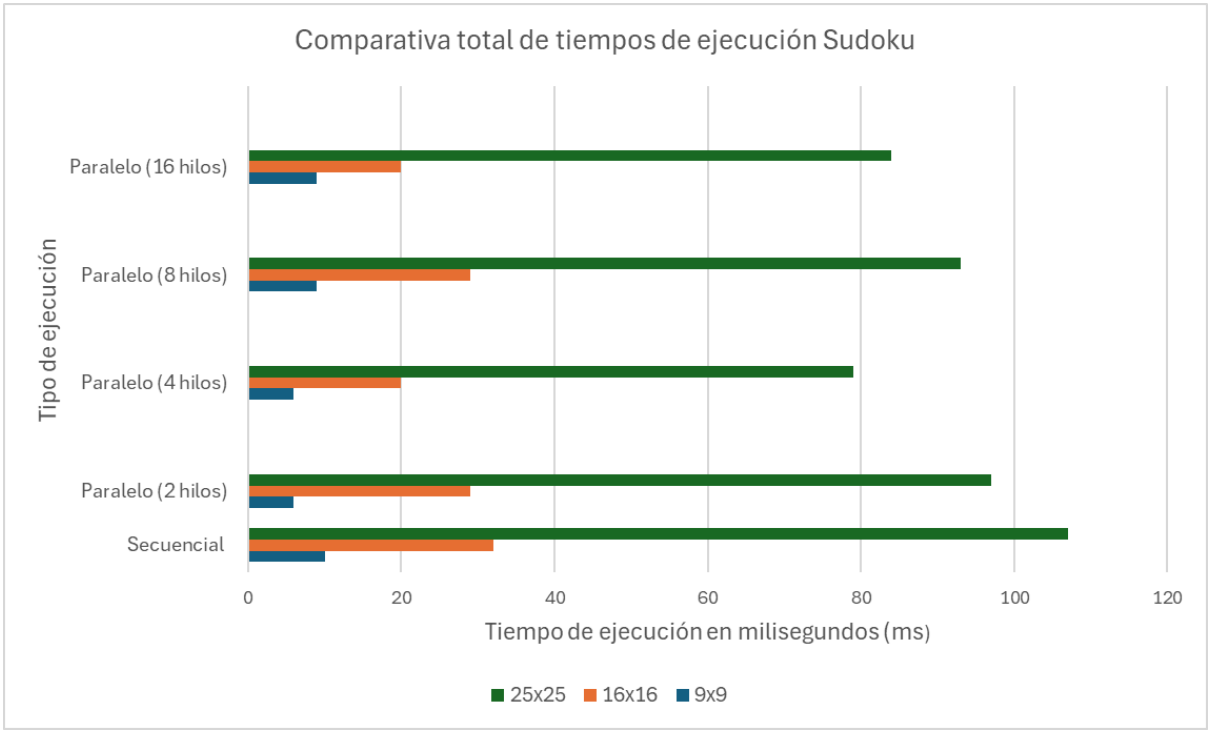



*Fuente:Elaboración propia*

La reducción más notoria de tiempo se ve en este tablero, donde el tiempo de 107 ms en secuencial se reduce a 79-97 ms con paralelización. Aunque no se logró una reducción completa proporcional con el aumento de hilos, la mejora es clara.

### 7.2.4 Comparación total de tableros.

Gráfico N°4: Comparación de tableros.




*Fuente:Elaboración propia*

Se realizaron pruebas utilizando diferentes cantidades de hilos (2,4, 8 y 16) para observar el impacto de la paralelización en el rendimiento algorítmico. Al aumentar el número de hilos, se espera que el algoritmo aproveche mejor los núcleos disponibles y por lo tanto reduzca los tiempos de ejecución. Sin embargo, con los resultados se puede notar que el beneficio de agregar más hilos no siempre es lineal. En algunos casos, aumentar el número de hilos después de cierto punto no presenta mejoras significativas.

La implementación de la paralelización con OpenMP ha permitido aprovechar los múltiples núcleos de la CPU, lo que ha disminuido los tiempos en que se demora resolver el Sudoku, especialmente para tableros más grandes. La mejora observada cumple con optimizar el algoritmo secuencial y demuestra la efectividad de paralelizar tareas computacionales.

# 8. Conclusiones

En el desarrollo del proyecto se plantearon diversas soluciones para abordar la solución del Sudoku con el objetivo de paralelizar el algoritmo. Durante ese proceso, se evaluaron varias opciones para mejorar el rendimiento y disminuir el tiempo de ejecución en tableros de diferentes tamaños. Las soluciones planteadas fueron:

- Resolver mediante cuadrantes para luego comprobar filas y columnas, si bien esta opción era una aproximación lógica, no proporcionaba una mejora significativa en los tiempos de ejecución y resultaba más compleja de implementar.

- Resolver a través de recursividad usando backtracking, es una técnica eficiente para resolver sudokus. A pesar de ser una solución secuencial, resultó ser una de las mejores soluciones en términos de simplicidad y eficacia, finalmente fue la que se utilizó en la implementación final debido a su rendimiento adecuado

- Paralelización por opciones de celdas vacías, esta opción consiste en paralelizar la resolución de celdas vacías. A pesar de ser una opción interesante, comparado con el backtracking y la paralelización por hilos resultó ser menos efectiva.

- Solución por Dancing Links (DLX), este algoritmo es eficiente para resolver Sudokus y optimizar su solución. Pero la complejidad e implementación no resultaron ser fáciles ni efectivas para adaptar a nuestro proyecto.

Al evaluar estas soluciones posibles, nos decidimos por la de backtracking recursivo como la solución más óptima, ya que resultó ser más eficiente y simple para desarrollar y aplicar al proyecto. La posibilidad de paralelizar este proceso con OpenMP permite aprovechar los múltiples núcleos de la CPU, y reduce considerablemente los tiempos de ejecución en tableros más grandes, logrando cumplir con los objetivos del proyecto.

Esto demuestra la importancia de elegir el enfoque adecuado para problemas computacionales complejos, y como la paralelización puede hacer una diferencia significativa en el rendimiento.

# 9. Bibliografía


- OpenMP Architecture Review Board. (2013, octubre). OpenMP 4.0 API C/C++ Syntax Quick Reference Card. https://www.openmp.org/resources/refguides/. https://www.openmp.org/wp-content/uploads/OpenMP-4.0-C.pdf

- Sato, M. (2018, mayo). OpenMP tutorial, University of Tsukuba. University of Tsukuba website. https://www.ccs.tsukuba.ac.jp/wp-content/uploads/sites/14/2018/01/2018-05-JKHPC.pdf

- Vargas Félix, M. (2010, 29 de abril). Makefiles, bevísimo tutoria. J. Miguel Vargas-Felix homepage. http://personal.cimat.mx:8181/~miguelvargas/Tutorials/Makefiles,%20bevisimo%20tutorial.pdf