Taller 1: Procesamiento Paralelo y el Problema del Viajero

William Aristizabal - Juan Joya - Ana Hernandez

1. Objetivo del Laboratorio:

El objetivo de este laboratorio es implementar y comparar la eficiencia de un algoritmo secuencial y uno paralelo (utilizando múltiples procesos) para resolver el Problema del Viajante (Travelling Salesperson Problem, TSP) mediante el método de fuerza bruta. El resultado de este laboratorio será un informe que demuestre el speedup (aceleración) logrado con el paralelismo.

2. Marco Teórico:

El Problema del Viajante (TSP) consiste en encontrar la ruta más corta que permite visitar un conjunto de ciudades exactamente una vez y regresar al punto inicial. Es considerado un problema NP-difícil, ya que el número de rutas posibles crece factorialmente con el número de ciudades.

Debido a esta complejidad, el método de fuerza bruta, que evalúa todas las permutaciones posibles, se vuelve computacionalmente costoso. Sin embargo, este método es ideal para comparar rendimiento entre enfoques secuenciales y paralelos, ya que su ejecución es repetitiva e independiente entre rutas, lo que permite distribuir el trabajo entre varios procesos.

El procesamiento paralelo divide las tareas en subprocesos o procesos simultáneos, reduciendo el tiempo total de cómputo cuando se cuenta con un CPU multinúcleo. El TSP

con fuerza bruta es un caso perfecto para estudiar el paralelismo gracias a su naturaleza altamente divisible.

3. Metodología:

En este proyecto se implementó y comparó dos enfoques de solución para el problema del viajero mediante el método de fuerza bruta, En el código se implementó una versión secuencial, ejecutada en un solo proceso, y una versión paralela, utilizando múltiples procesos, esto con el fin de comparar su eficiencia e identificar la diferencia entre las dos implementaciones.

El método de fuerza bruta evalúa todas las posibles permutaciones de rutas entre las ciudades y selecciona la de menor distancia total. Aunque este enfoque no es eficiente para grandes cantidades de ciudades, es ideal para evaluar el rendimiento entre ejecución secuencial y paralela bajo las mismas condiciones.

Datos:

Los datos se obtuvieron mediante coordenadas aleatorias para un conjunto de n ciudades, representadas como puntos en un plano 2D, en nuestro caso se probaron para 9, 10 y 11 ciudades, para evaluar el comportamiento de cada algoritmo. A partir de estas coordenadas se construyó una matriz de distancias euclidianas que contiene las distancias entre cada par de ciudades:

Esta matriz fue la base para calcular la distancia total de cada ruta posible.

Configuración de Hardware:

Las pruebas se realizaron en un equipo con las siguientes características:

- Procesador: Intel Core i5 (8 núcleos, 2.4 GHz)
- Memoria RAM: 16 GB
- Tipo de CPU: 64 bits
- Número de hilos disponibles: 8
- Sistema operativo: Windows 10

Estas características permiten ejecutar múltiples procesos en paralelo y observar cómo el rendimiento varía al incrementar el número de ciudades o procesos.

Configuración de software:

- Lenguaje: Python 3.10.6
- El entorno de ejecución fue VS Code.

- Librerías utilizadas:
    - numpy → manejo de matrices y cálculos numéricos.
    - itertools → generación de permutaciones de rutas.
    - matplotlib → graficar la mejor ruta.
    - concurrent.futures → gestión del paralelismo con ProcessPoolExecutor.
    - time → medición de tiempos de ejecución.

Explicacion de Codigo:

1. Generación de ciudades y matriz de distancias:

   Se crean n ciudades con coordenadas aleatorias y se calcula la matriz de distancias euclidianas entre todas las parejas de ciudades.

2. Función secuencial:

   Recorre todas las permutaciones posibles de rutas, calcula su distancia total y selecciona la ruta con la menor distancia.

   Se mide el tiempo total de ejecución con time.perf_counter().

3. Función paralela:

   Utiliza ProcessPoolExecutor para distribuir subconjuntos de rutas entre varios procesos.

   Cada proceso calcula la mejor ruta de su lote y luego los resultados se combinan para encontrar la mejor global.

Ambos métodos imprimen:

- La mejor ruta encontrada
- La distancia total mínima
- El tiempo de ejecución
- Finalmente se grafica la mejor ruta con matplotlib.

4. Resultados:

Para evaluar el rendimiento de los algoritmos, se realizaron pruebas con 9, 10 y 11 ciudades, midiendo el tiempo de ejecución de las versiones secuencial y paralela. A continuación, se describen los hallazgos obtenidos a partir de las figuras correspondientes.

9 Ciudades:

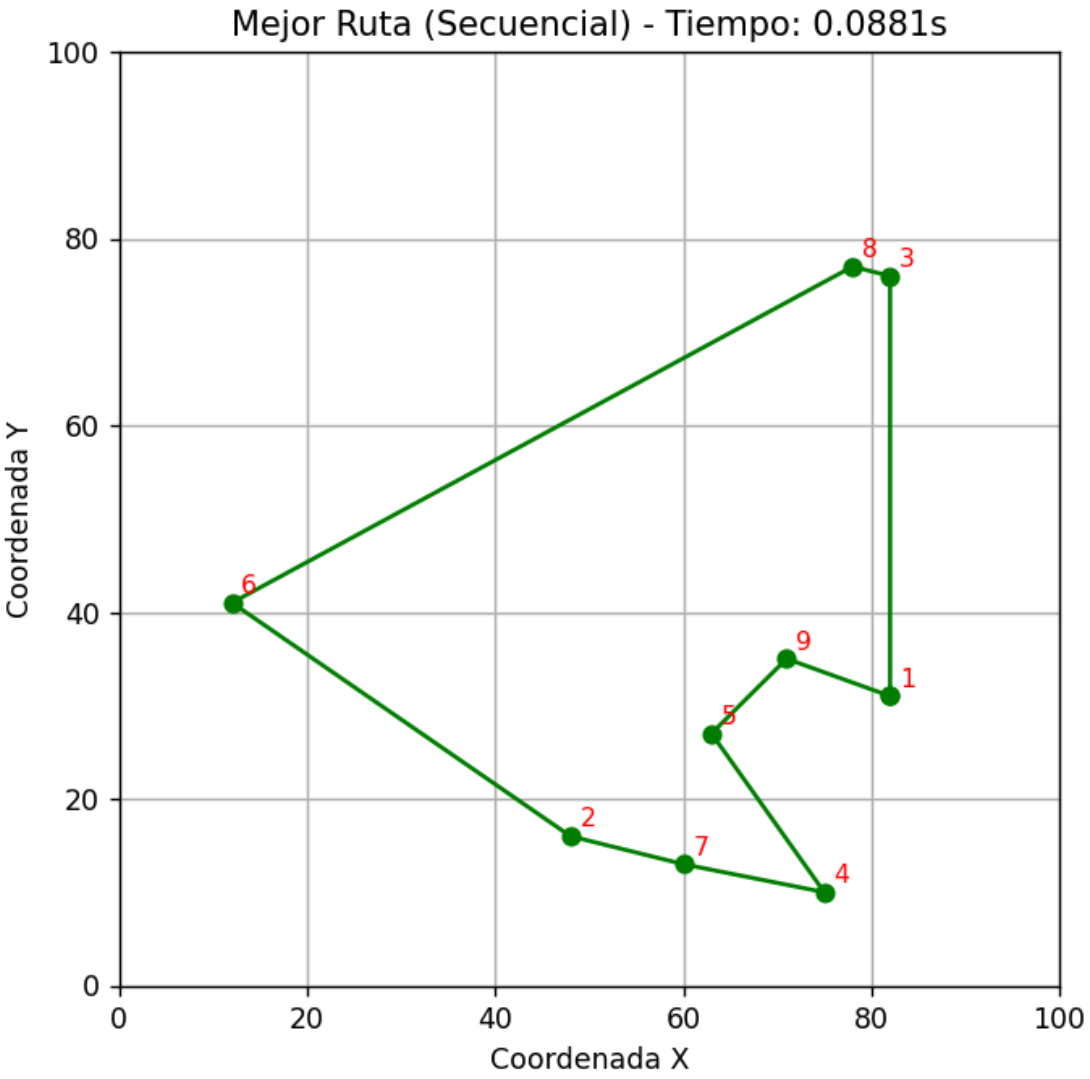

Figura 1.1

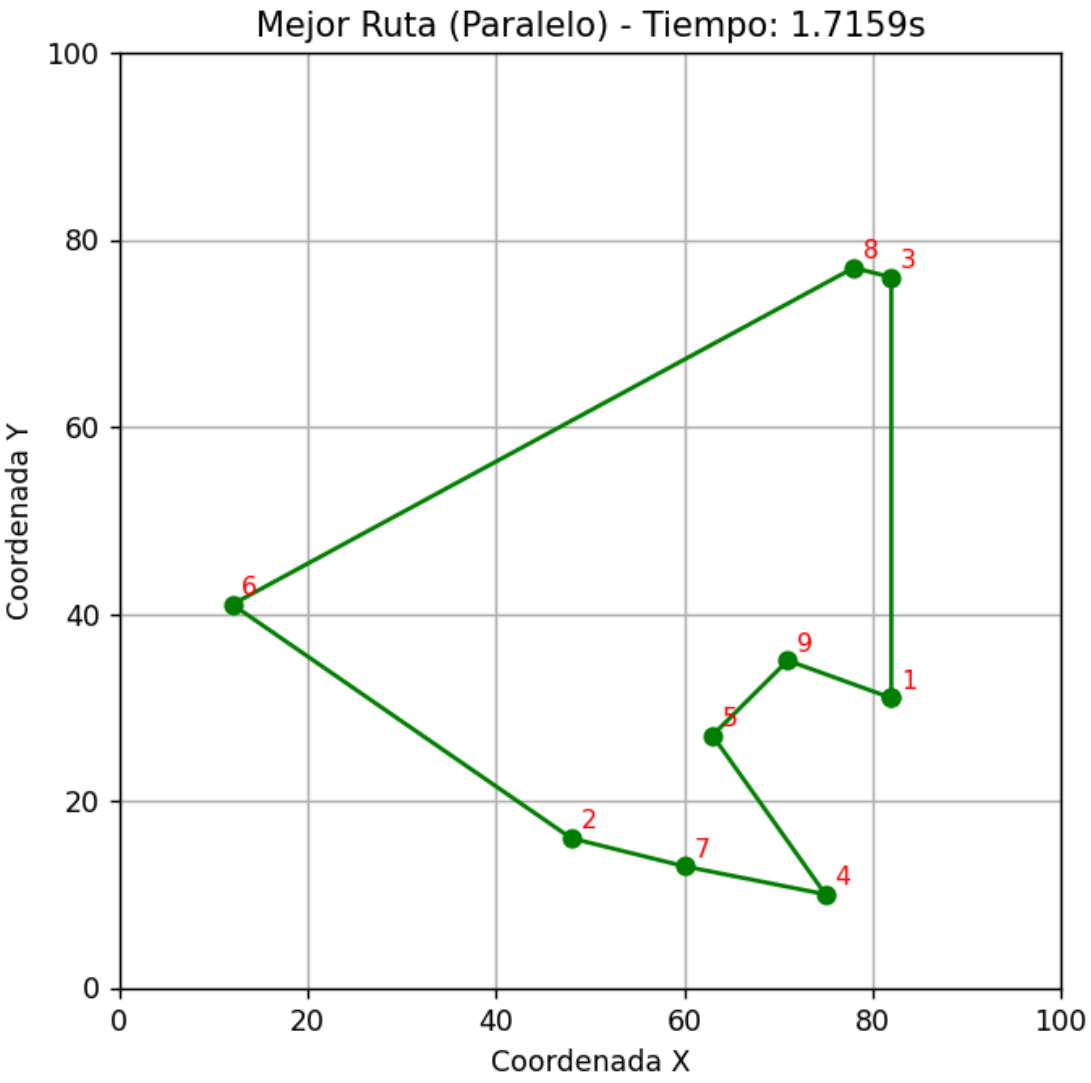

Figura 1.2.

En las Figuras 1.1 y 1.2 se observa que ambos algoritmos encuentran la misma ruta óptima. Sin embargo, el algoritmo secuencial presenta un mejor desempeño, con una diferencia de más de 1 segundo frente a la versión paralela.

Esto indica que para problemas pequeños, donde la carga computacional no es tan elevada, el algoritmo secuencial puede ser más eficiente debido a que el algoritmo paralelo incurre en una sobrecarga asociada a la creación y coordinación de procesos.

10 Ciudades:

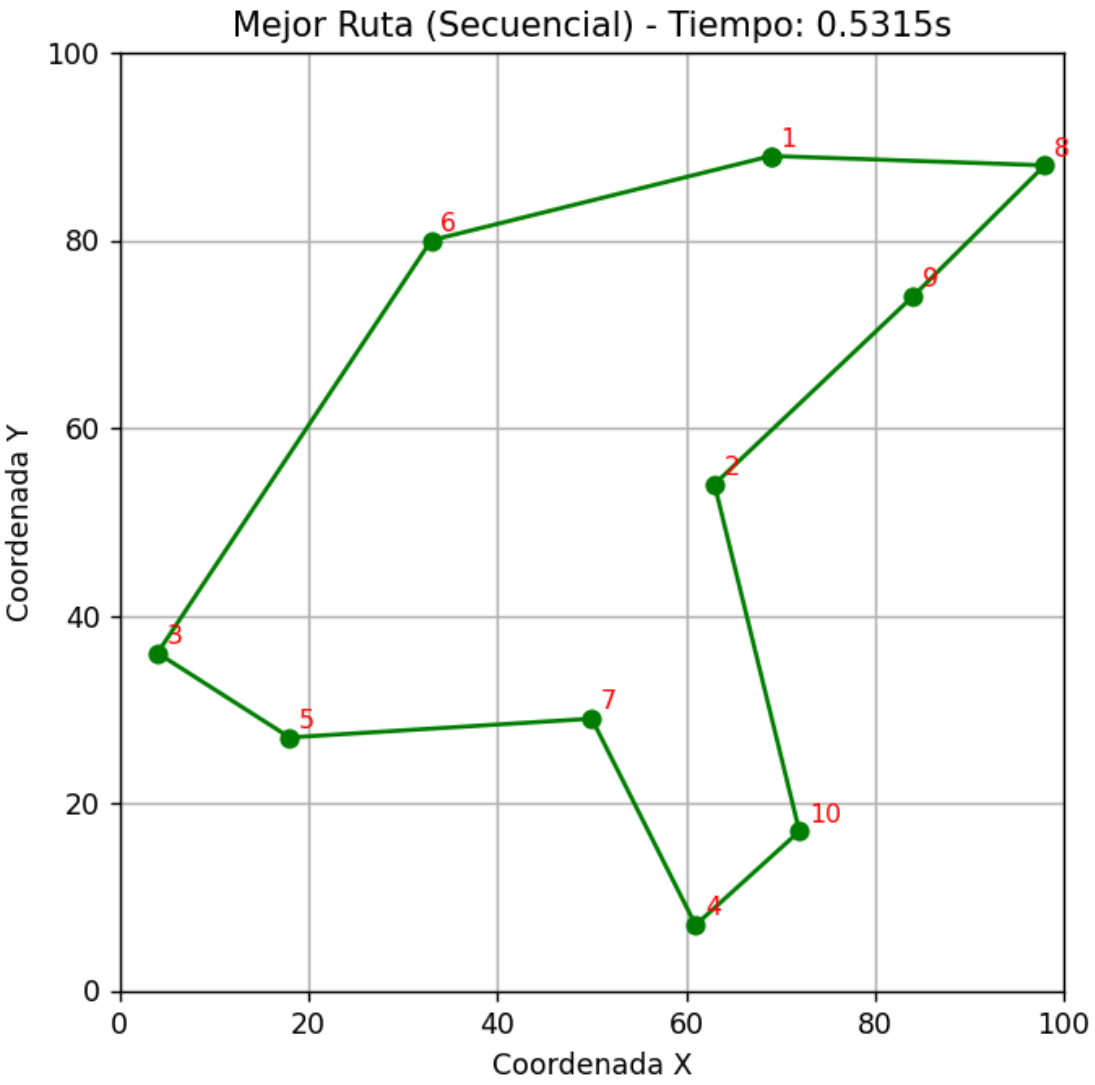

Figura 2.1.

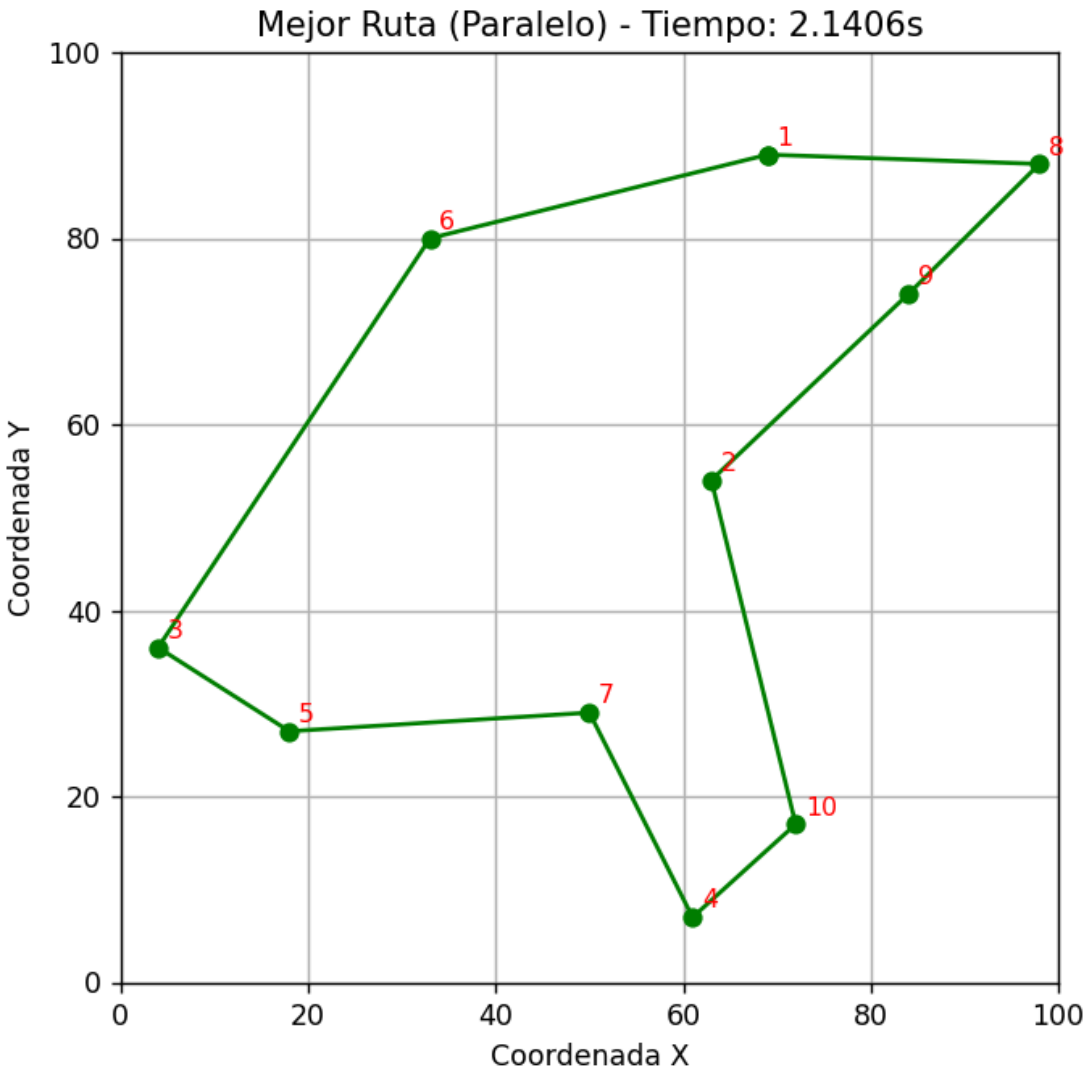

Figura 2.2.

Los resultados de las Figuras 2.1 y 2.2 presentan un comportamiento similar al de la prueba con 9 ciudades. Nuevamente, el algoritmo secuencial supera al paralelo en tiempo de ejecución.

Esto refuerza la idea de que, cuando el número de ciudades aún no genera una carga computacional suficientemente grande, el paralelismo no logra compensar la sobrecarga de comunicación entre procesos.

11 Ciudades:

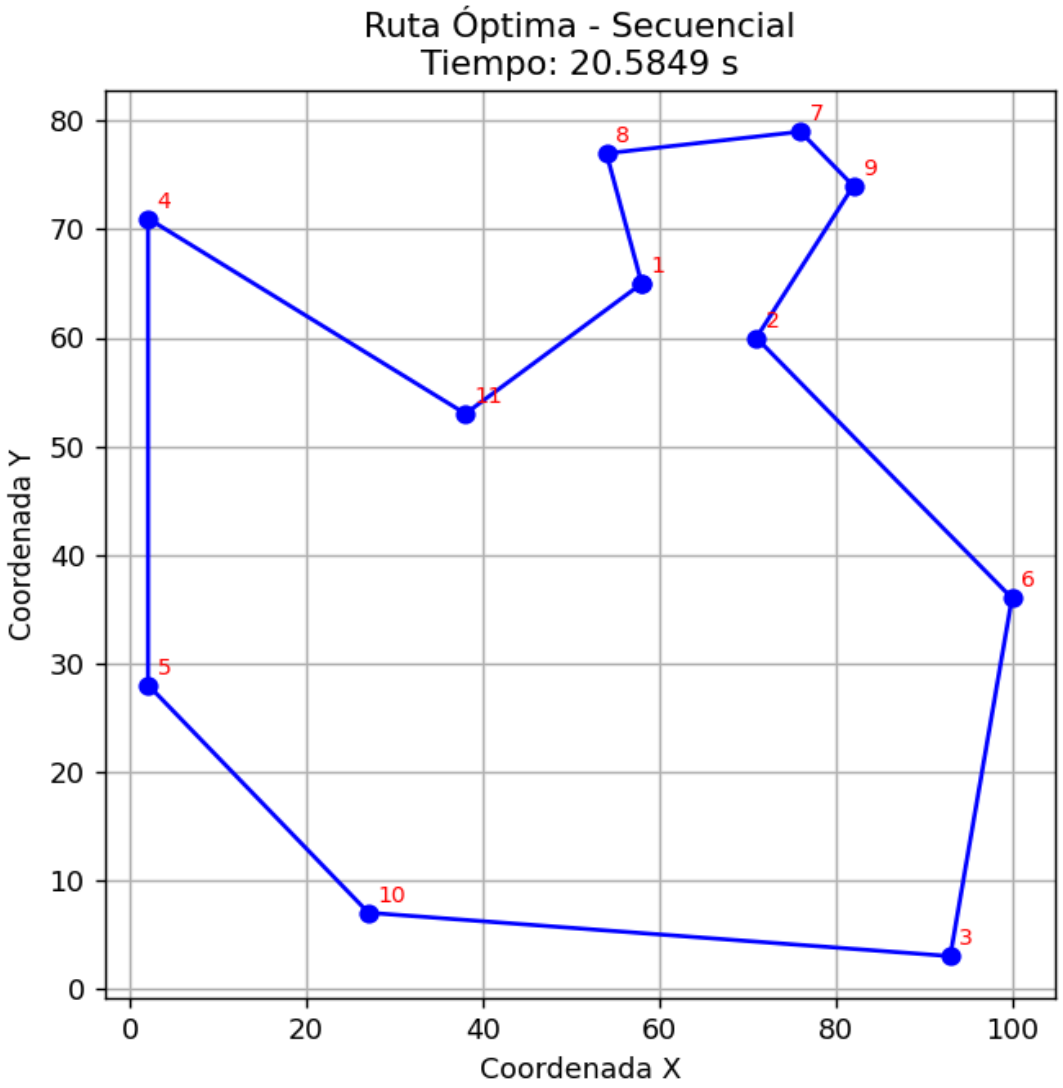

Figura 3.1.

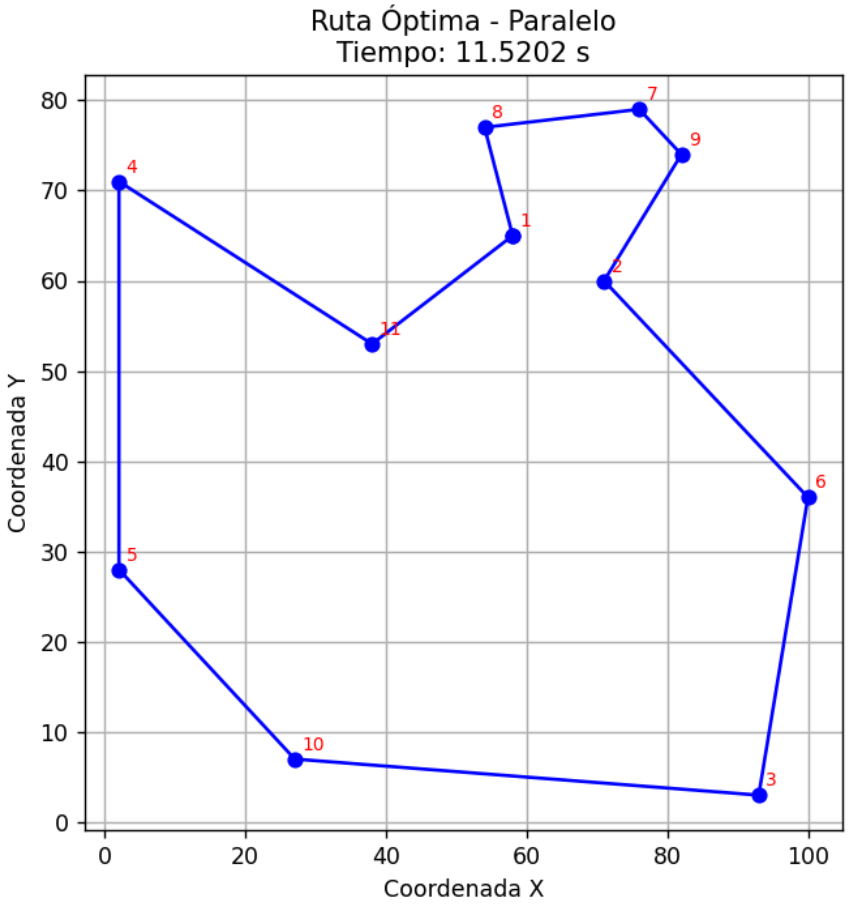

Figura 3.2.

En las Figuras 3.1 y 3.2 se observa un cambio significativo respecto a los casos anteriores. Para 11 ciudades, el algoritmo paralelo obtiene un tiempo de ejecución considerablemente menor que la versión secuencial.

Este comportamiento indica que, al aumentar el tamaño del problema, el procesamiento paralelo se vuelve más eficiente, ya que la carga de trabajo puede dividirse adecuadamente entre los núcleos del procesador, reduciendo el tiempo total de ejecución.

En otras palabras, cuando el número de permutaciones es suficientemente grande, el paralelismo sí logra compensar la sobrecarga inicial y se convierte en la mejor opción.

5. **Análisis de rendimiento:**

Para comparar las implementaciones secuencial y paralela del algoritmo de fuerza bruta para el TSP, se evaluaron los tiempos de ejecución con 9, 10 y 11 ciudades. Los resultados obtenidos fueron los siguientes:

Número de ciudades	Secuencial	Paralelo
9	0.08 s	1.71 s
10	0.50 s	2.10 s
11	20.50 s	11.50 s

A partir de estos datos, se pueden extraer los siguientes análisis:

Caso de 9 ciudades:

El algoritmo secuencial tarda 0.08 segundos, mientras que el paralelo tarda 1.71 segundos, siendo más de 20 veces más lento.

Esto se debe a que para un número reducido de ciudades, la cantidad de permutaciones es pequeña y el trabajo computacional no es suficiente para compensar la sobrecarga del paralelismo, la cual incluye:

- Creación de procesos
- Transferencia de datos

- Recolección y combinación de resultados

Caso de 10 ciudades:

El comportamiento es similar. El método secuencial requiere 0.50 segundos, mientras que el paralelo toma 2.10 segundos.

Aunque la diferencia es menor que en el caso anterior, el algoritmo paralelo sigue siendo aproximadamente 4 veces más lento.

Esto confirma que, cuando el tamaño del problema aún no es lo suficientemente grande, el costo de gestionar múltiples procesos continúa superando los beneficios del paralelismo.

Caso de 11 ciudades:

Aquí se observa un cambio importante. El algoritmo secuencial tarda 20.50 segundos, mientras que el paralelo tarda 11.50 segundos, siendo 1.8 veces más rápido.

Este resultado demuestra que, a partir de cierto umbral, el paralelismo sí se vuelve beneficioso. El número de permutaciones crece factorialmente, por lo que la carga de trabajo aumenta de manera explosiva.

En este punto, dividir el cálculo entre varios núcleos reduce significativamente el tiempo total, haciendo que el paralelismo supere al enfoque secuencial.

6. Conclusiones:

- El paralelismo no siempre garantiza mejores tiempos de ejecución.

  En problemas pequeños, como el TSP con 9 y 10 ciudades, la sobrecarga asociada a la creación y coordinación de procesos supera el beneficio del cómputo paralelo. Esto hace que el enfoque secuencial sea significativamente más eficiente en estos casos.

- El paralelismo se vuelve útil a medida que aumenta la complejidad del problema.

  Cuando el número de ciudades aumenta a 11, el crecimiento factorial del espacio de búsqueda produce una carga computacional suficiente para que el paralelismo sea ventajoso. En esta prueba, el algoritmo paralelo casi duplicó la eficiencia respecto al secuencial.

- El límite a partir del cual el paralelismo es beneficioso depende del costo del problema y del hardware.

  En sistemas con más núcleos o menor latencia en la creación de procesos, el punto donde el paralelismo supera al secuencial puede aparecer antes. Esto demuestra la fuerte relación entre la arquitectura del hardware y el rendimiento de algoritmos paralelos.

- El TSP es un caso de estudio ideal para evaluar paralelismo.

  Debido a su complejidad factorial, incluso incrementos pequeños en el número de ciudades generan diferencias marcadas en el tiempo de ejecución. Esta característica permite observar de manera clara cuándo y cómo el paralelismo mejora el rendimiento.

- El laboratorio permitió comprender la importancia de medir, comparar y justificar el uso del paralelismo.

  Más allá de la teoría, los resultados experimentales confirmaron que las decisiones de diseño en cómputo paralelo deben sustentarse en pruebas concretas, ya que la eficiencia depende tanto del problema como del entorno de ejecución.