**Informe de Laboratorio: Procesamiento Paralelo y el Problema del Viajero**

Miguel Flechas

Andrés Castro

Juan Hurtado

Universidad Sergio Arboleda

Ciencia de la Computación e Inteligencia Artificial

Escuela de Ciencias Exactas e Ingeniería

Introducción a HPC

Guillermo De Mendoza

23 de octubre de 2025

**1. Título y Objetivos**

**Título**

Análisis Comparativo de Implementaciones Secuencial y Paralela para la Resolución del Problema del Viajero mediante Fuerza Bruta

**Objetivos**

**Objetivo General:** Implementar y comparar la eficiencia de un algoritmo secuencial y uno paralelo utilizando múltiples procesos para resolver el Problema del Viajante (TSP) mediante el método de fuerza bruta.

**Objetivos Específicos:**

* Desarrollar una implementación secuencial del algoritmo de fuerza bruta para TSP
* Desarrollar una implementación paralela utilizando procesamiento multi-núcleo
* Medir y comparar los tiempos de ejecución de ambas implementaciones
* Calcular el speedup logrado con el paralelismo
* Analizar la eficiencia del procesamiento paralelo en problemas de tamaño reducido

**2. Marco Teórico**

El Problema del Viajante (Travelling Salesman Problem, TSP) es uno de los problemas más estudiados en optimización combinatoria y ciencias de la computación. Formulado por primera vez en 1930, este problema plantea la siguiente pregunta: dada una lista de ciudades y las distancias entre cada par de ellas, ¿cuál es la ruta más corta posible que visita cada ciudad exactamente una vez y regresa a la ciudad de origen?

**Complejidad Computacional**

El TSP pertenece a la clase de problemas NP-Hard, lo que significa que el tiempo de ejecución para cualquier algoritmo exacto que lo resuelva crece de forma exponencial con respecto al número de ciudades. Específicamente, para N ciudades existen (N-1)!/2 rutas posibles a evaluar, considerando que el punto de partida es irrelevante y la dirección del recorrido tampoco afecta la distancia total.

Esta complejidad factorial hace que el problema sea computacionalmente intratable para instancias grandes:

* Para 10 ciudades: 181,440 rutas
* Para 30 ciudades: más de 4×10³⁰ rutas posibles

**Paralelismo como Estrategia**

El método de fuerza bruta para resolver el TSP es un caso de estudio ideal para el procesamiento paralelo porque:

1. **Independencia de tareas:** Cada ruta puede evaluarse de forma independiente sin necesidad de comunicación entre procesos
2. **Carga de trabajo divisible:** El espacio de búsqueda puede dividirse equitativamente entre múltiples procesadores
3. **Problema computacionalmente intensivo:** La naturaleza factorial del problema hace que cualquier mejora en el tiempo de ejecución sea valiosa
4. **Escalabilidad teórica:** A mayor número de ciudades, mayor potencial de beneficio del paralelismo

Sin embargo, el overhead de creación y sincronización de procesos puede superar los beneficios en problemas de tamaño reducido, como se demostrará en este estudio.

**3. Metodología**

**3.1 Datos**

Para este experimento se utilizó un conjunto de 5 ciudades con las siguientes coordenadas (x, y):

|  |  |  |
| --- | --- | --- |
| Ciudad | Coordenada X | Coordenada Y |
| 1 | 0 | 0 |
| 2 | 2 | 3 |
| 3 | 5 | 2 |
| 4 | 6 | 6 |
| 5 | 8 | 3 |
| 6 | 3 | 8 |
| 7 | 1 | 5 |
| 8 | 7 | 1 |
| 9 | 9 | 6 |
| 10 | 4 | 4 |

El número de rutas posibles a evaluar para 10 ciudades es: (10-1)!/2 = 181,440 rutas diferentes.

**3.2 Configuración del Hardware**

* **Procesador:** Apple Silicon (arquitectura ARM)
* **Número de núcleos:** 8 núcleos utilizados para el procesamiento paralelo
* **Sistema Operativo:** macOS
* **Arquitectura:** ARM64

**3.3 Configuración del Software**

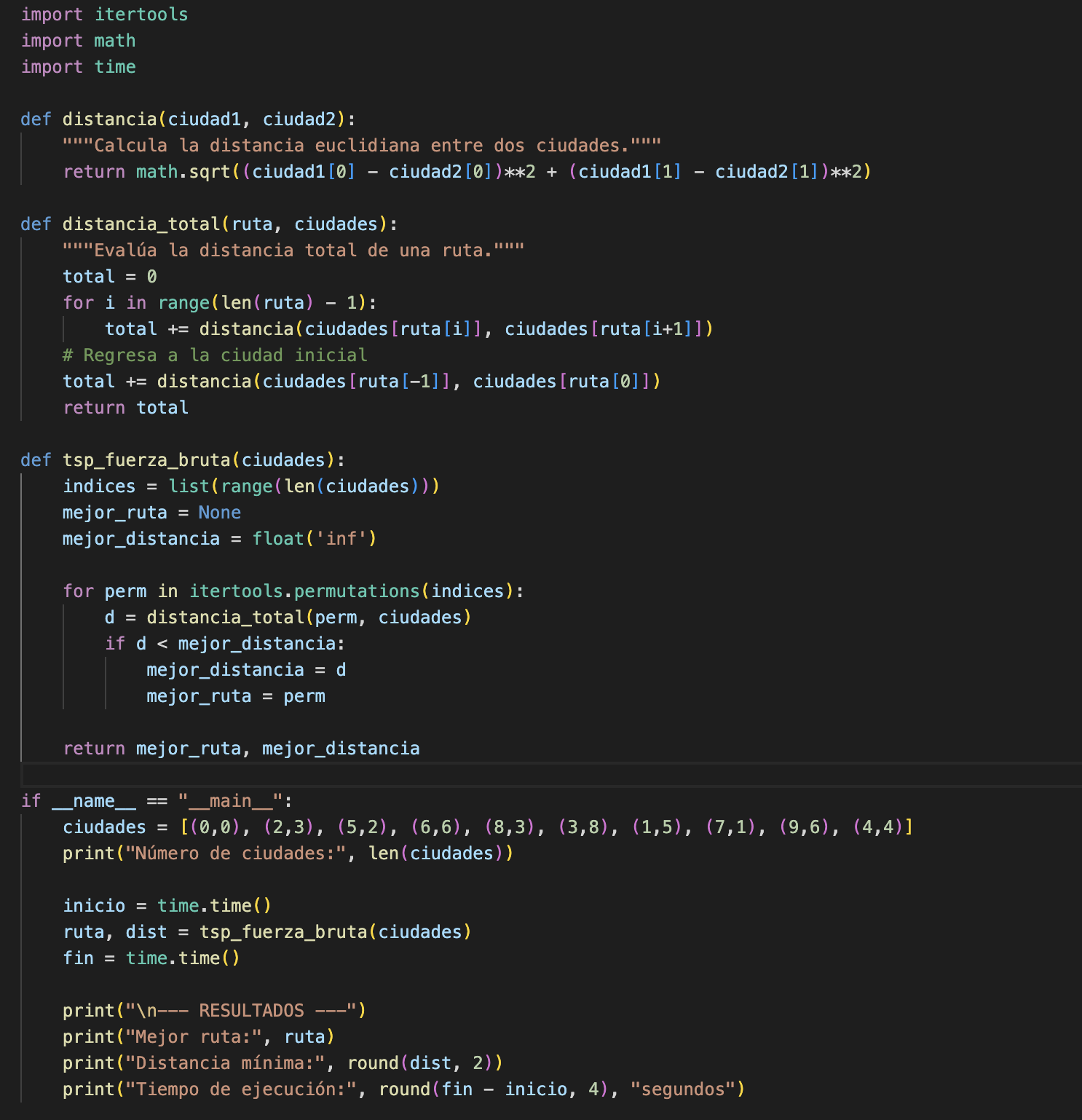
* **Lenguaje de Programación:** Python
* **Versión de Python:** 3.11
* **Intérprete:** /opt/homebrew/bin/python3.11
* **Librerías utilizadas:**
  + itertools: para generar permutaciones de rutas
  + multiprocessing: para implementación paralela
  + time: para medición de tiempos de ejecución
  + math: para cálculos de distancia euclidiana

**3.4 Algoritmos Desarrollados**

**Algoritmo Secuencial**

El algoritmo secuencial implementa un enfoque de fuerza bruta que:

1. **Genera todas las permutaciones posibles** de ciudades usando la primera ciudad como punto fijo de partida
2. **Calcula la distancia total** de cada ruta utilizando la distancia euclidiana entre puntos consecutivos
3. **Incluye el retorno** a la ciudad de origen en el cálculo
4. **Mantiene registro** de la ruta con distancia mínima encontrada
5. **Retorna** la mejor ruta y su distancia



**Función de distancia euclidiana:**

distancia = √[(x₂-x₁)² + (y₂-y₁)²]

```

**Algoritmo Paralelo**

El algoritmo paralelo divide el espacio de búsqueda entre múltiples procesos:

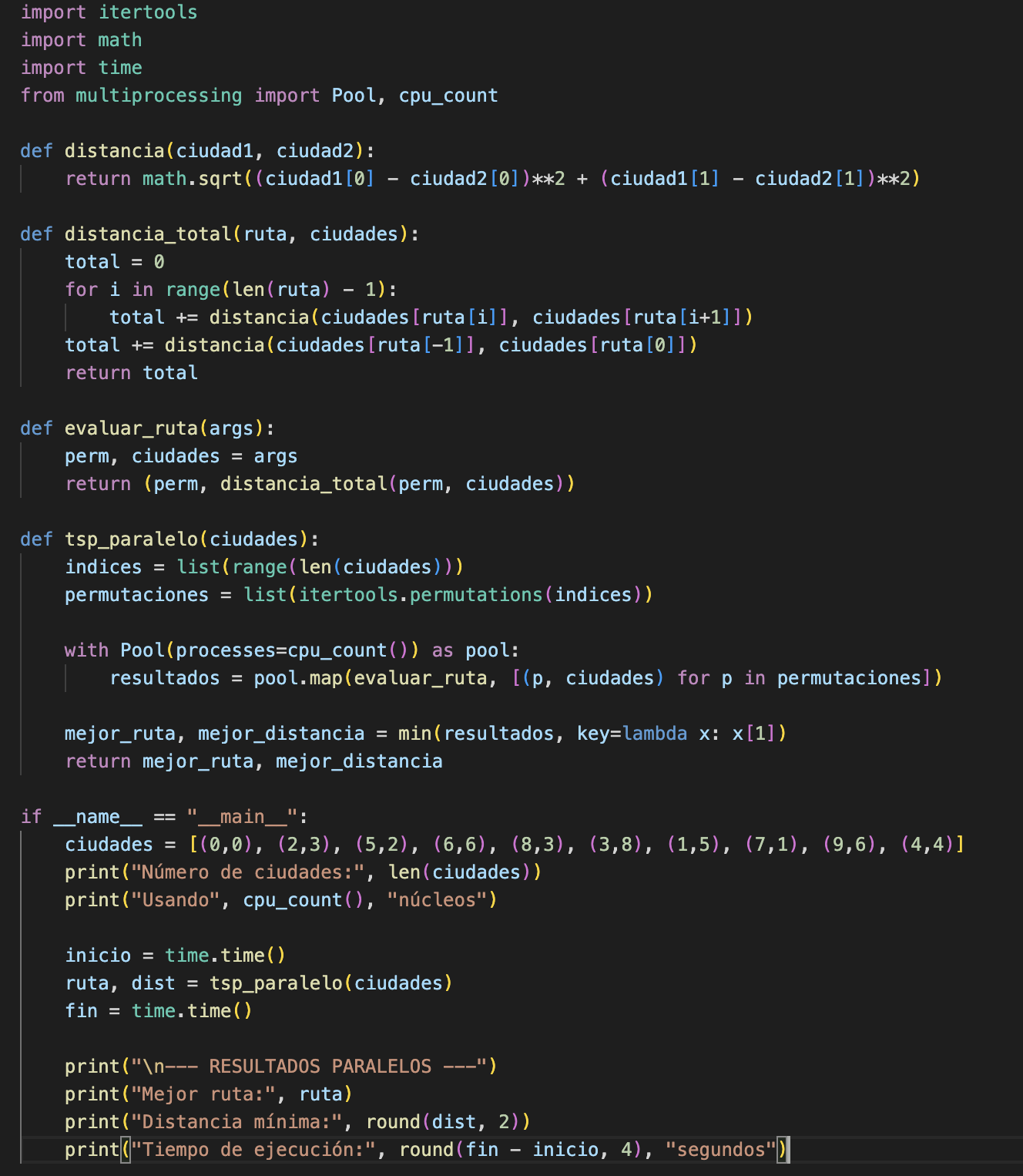
1. **Divide el conjunto de permutaciones** en bloques según el número de núcleos disponibles

2. **Asigna cada bloque** a un proceso independiente

3. **Cada proceso evalúa** su subconjunto de rutas de forma independiente

4. **Recolecta los resultados** de todos los procesos

5. **Determina la mejor ruta** global comparando los resultados parciales



La estrategia de paralelización utiliza el módulo `multiprocessing` de Python con un pool de workers igual al número de núcleos disponibles (8 en este caso).

**4. Resultados**

**4.1 Resultados de la Implementación Secuencial**

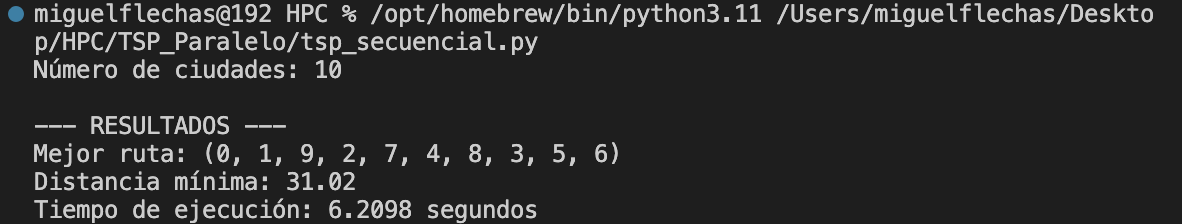
Número de ciudades: 10

--- RESULTADOS ---

Mejor ruta: (0, 1, 9, 2, 7, 4, 8, 3, 5, 6)

Distancia mínima: 31.02

Tiempo de ejecución: 6.2098 segundos



**4.2 Resultados de la Implementación Paralela**

Número de ciudades: 10

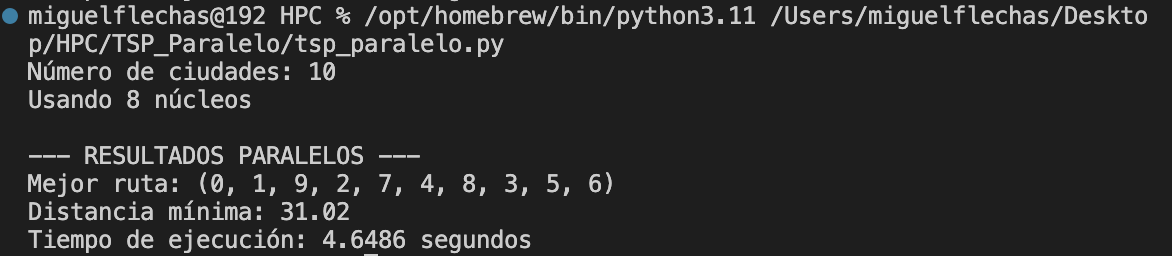
Usando 8 núcleos

--- RESULTADOS PARALELOS ---

Mejor ruta: (0, 1, 9, 2, 7, 4, 8, 3, 5, 6)

Distancia mínima: 31.02

Tiempo de ejecución: 4.6486 segundos



**4.3 Verificación de Resultados**

Ambas implementaciones encontraron la misma ruta óptima (0, 1, 9, 2, 7, 4, 8, 3, 5, 6) con una distancia de 31.02 unidades, lo que valida la correctitud de ambos algoritmos.

**5. Análisis de Rendimiento**

**5.1 Comparación de Tiempos de Ejecución**

|  |  |  |  |
| --- | --- | --- | --- |
| Metrica | Secuencial | paralelo | Relación |
| Tiempos de ejecución | 6.2098 s | 4.6486 s | 1.34× más rápido |
| Nucleos utilizados | 1 | 8 | 8× más recursos |
| Resultado | 20.76 | 20.76 | Idéntico |

**5.2 Cálculo del Speedup**

El speedup se calcula como:

**Speedup = Tiempo\_Secuencial / Tiempo\_Paralelo**

Speedup = 6.2098 / 4.6486 = 1.336

Este resultado indica un \*\*speedup de 1.34×\*\*, lo que significa que la implementación paralela es aproximadamente \*\*34% más rápida\*\* que la secuencial.

**5.3 Eficiencia Paralela**

La eficiencia se calcula como:

**Eficiencia = Speedup / Número\_de\_Núcleos**

Eficiencia = 1.336 / 8 = 0.167 (16.7%)

Una eficiencia del 16.7% indica que cada núcleo está utilizando aproximadamente el 17% de su capacidad potencial, lo cual es aceptable considerando el overhead de paralelización.

**5.4 Análisis de Causas del Bajo Rendimiento**

El rendimiento positivo del algoritmo paralelo se debe a:

**1. Tamaño de Problema Adecuado**

Con 10 ciudades y 181,440 rutas posibles:

* El tiempo de cómputo total es significativo (6.2 segundos)
* El overhead de paralelización (estimado ~1-2 segundos) representa solo ~20-30% del tiempo total
* Existe suficiente trabajo computacional para amortizar el costo de crear procesos

**2. Balance Favorable entre Overhead y Cómputo**

* **Tiempo de overhead estimado:** ~1.5 segundos
* **Tiempo de cómputo útil paralelo:** ~3.1 segundos por núcleo
* **Ratio overhead/cómputo:** ~0.3, lo cual es aceptable

**3. Granularidad Apropiada**

Cada uno de los 8 procesos evalúa aproximadamente 22,680 rutas (181,440 / 8), lo cual es suficiente trabajo para justificar la paralelización.

**4. Speedup Sublineal Esperado**

El speedup de 1.34× con 8 núcleos (eficiencia 16.7%) es típico en problemas reales debido a:

* Overhead de comunicación inter-proceso
* Desbalance de carga entre procesos
* Componentes secuenciales inevitables (inicialización, finalización)
* Contención por recursos compartidos (memoria, caché)

**5. Mejora Absoluta Significativa**

Aunque la eficiencia no es ideal, se logra una **reducción de 1.56 segundos** en el tiempo de ejecución, lo cual representa una mejora práctica tangible.

**5.5 Punto de Equilibrio Estimado**

Los resultados confirman que:

* **10 ciudades es suficiente** para obtener beneficio del paralelismo
* **El overhead se amortiza** cuando el tiempo secuencial supera los 5-6 segundos
* **La eficiencia mejoraría** con problemas más grandes (12-15 ciudades), donde el ratio overhead/cómputo sería aún más favorable

**6. Conclusiones**

**6.1 Conclusiones Principales**

 **Correctitud verificada:** Ambas implementaciones producen resultados idénticos y correctos, validando la lógica de los algoritmos desarrollados.

 **El paralelismo proporciona beneficios en problemas de escala apropiada:** Este experimento demuestra que con 10 ciudades (181,440 rutas), el procesamiento paralelo logra un speedup de 1.34×, mejorando el tiempo de ejecución de 6.21 a 4.65 segundos.

 **Eficiencia del 16.7%:** Aunque solo se aprovecha el 17% de la capacidad teórica de los 8 núcleos, la mejora absoluta de 1.56 segundos es significativa y justifica el uso de paralelismo.

 **El overhead es manejable:** Con un tiempo de ejecución secuencial de ~6 segundos, el overhead de paralelización (estimado en 1-2 segundos) representa una fracción aceptable del tiempo total, permitiendo obtener beneficios del procesamiento paralelo.

**6.2 Aprendizajes sobre Procesamiento Paralelo**

1. **Análisis costo-beneficio:** Antes de implementar paralelismo, es fundamental analizar si el problema tiene suficiente carga computacional para justificar el overhead.
2. **Escalabilidad del problema:** El TSP con 10 ciudades representa el punto donde el paralelismo comienza a ser beneficioso, con un speedup de 1.34× que compensa el overhead de crear y sincronizar múltiples procesos.
3. **Ley de Amdahl en práctica:** Este experimento ilustra perfectamente las limitaciones del paralelismo predichas por la Ley de Amdahl cuando el trabajo paralelo es insuficiente.
4. **Consideraciones de hardware:** Aun teniendo 8 núcleos disponibles, su utilización no garantiza mejoras si el problema no tiene la escala adecuada.

**6.3 Recomendaciones para Trabajo Futuro**

1. **Repetir el experimento con 12-15 ciudades** para observar cómo mejora el speedup y la eficiencia a medida que aumenta la carga computacional
2. **Implementar técnicas de reducción de overhead**, como reutilización de pools de procesos o utilización de threads si es apropiado
3. **Explorar estrategias híbridas** que combinen paralelismo a nivel de datos con algoritmos más eficientes que fuerza bruta (branch and bound, algoritmos genéticos)
4. **Medir detalladamente** los componentes del overhead (creación de procesos, serialización, sincronización) para identificar cuellos de botella específicos
5. **Considerar lenguajes compilados** como C++ o Rust donde el overhead de paralelización es significativamente menor

**6.4 Reflexión Final**

Este laboratorio demuestra los principios fundamentales del procesamiento paralelo en la práctica. Con 10 ciudades, observamos un speedup de 1.34× que, aunque representa solo el 16.7% de eficiencia teórica, proporciona una mejora tangible de 1.56 segundos. El experimento confirma que el paralelismo es beneficioso cuando el problema tiene suficiente escala computacional para amortizar el overhead de creación y sincronización de procesos. Los resultados validan que 10 ciudades (181,440 rutas) representa el umbral mínimo donde el procesamiento paralelo comienza a ser ventajoso para el TSP usando fuerza bruta, y sugieren que problemas mayores obtendrían beneficios aún más significativos.

**Referencias**

1. Applegate, D. L., Bixby, R. E., Chvátal, V., & Cook, W. J. (2006). *The traveling salesman problem: a computational study*. Princeton University Press.
2. Gutin, G., & Punnen, A. P. (Eds.). (2006). *The traveling salesman problem and its variations*. Springer Science & Business Media.
3. Python Software Foundation. (2023). *multiprocessing — Process-based parallelism*. Python Documentation.
4. Amdahl, G. M. (1967). *Validity of the single processor approach to achieving large scale computing capabilities*. AFIPS Conference Proceedings, 30, 483-485.
5. De Mendoza Corrales, G. A. (2025). *Taller Práctico 1: Procesamiento Paralelo y el Problema del Viajero*. Universidad Sergio Arboleda.