Problema del vendedor viajero: Evaluación en CPU y GPU.

Camila Cárdenas Fuentes

Facultad de Ingeniería, UACh,
General Lagos 2086 Valdivia, Chile

1. Introducción

El problema del vendedor viajero o TSP (Travel Salesmen Problem) consiste en encontrar la ruta más corta que debe llevar a cabo un vendedor que debe recorrer un determinado y preestablecido conjunto de ciudades. Comenzando en una ciudad de origen, a la cual debe volver al finalizar su recorrido, con la restricción de que por cada ciudad sólo pase una vez. Siento este uno de los problemas de optimización más estudiados.
El TSP se puede representar con un grafo G = (N, A), donde N es la cantidad de ciudades y A es el conjunto de arcos que conectan a las ciudades entre si, cada arco tiene un peso d_{ij} el cual representan la distancia entre la ciudades i y la ciudad j [1].

Existen diferentes tipos de algoritmos para resolver el TSP, como los algoritmos genéticos (AGs) que son algoritmos de optimización basados en la teoría de la selección natural, la evolución y la genética [2] y los algoritmos de colonias de hormigas (ACO) que están inspirados en el comportamiento natural de las hormigas cuando buscan comida.

Para el análisis a realizar se utilizo el algoritmo de colonias de hormigas elaborado por Rachit Mehrotra [4], que posee lenguajes C++, CUDA y Ruby.

El algoritmo de colonias de hormigas es una técnica probabilística de búsqueda encontrar la solución de problemas de optimización combinatoria.
El cual consiste en que las hormigas recorren aleatoriamente los bordes de un gráfico, buscando el vértice objetivo. Cuando una hormiga, encuentra el objetivo regresa a la colonia, dejando marcadores de feromonas para que otras hormigas los sientas. Las feromonas aumentan la probabilidad de una siguiente hormiga siga el camino, pero con el tiempo las feromonas se evaporan. Por lo cual a medida que más hormigas siguen el camino y dejan caer más feromonas, se fomentan los caminos más cortos y el algoritmo converge hacia una solución casi óptima [3].

2. Problema y oportunidad

Se presenta la problemática encontrar algoritmos con un buen rendimiento, en relación al tiempo de ejecución y el speedup, para resolver el problema del vendedor viajero.
Gracias a los algoritmos existentes, como el algoritmo de colonias de hormigas se puede comparar su implementación en CPU y GPU.

3. Pregunta de Investigación

Para el presente proyecto se genero la siguiente pregunta de investigación:

¿El algoritmo del colonias de hormigas, para resolver el problema del vendedor viajero, tiene un mejor rendimiento en GPU en comparación a su implementación en CPU?.

4. Metodología

Para la evaluación del algoritmo colonias de hormigas en CPU y GPU, se analizaran dos medidas de rendimiento que son :

1. El tiempo de ejecución, medido en segundos (s).

2. El speedup que es un relación entre los tiempos de ejecución secuencial y paralela de un algoritmo (CPU/GPU). El objetivo de la paralelización es disminuir el tiempo de ejecución con el fin de maximizar el speedup frente al algoritmo secuencial.

El algoritmo seleccionado tiene una implementación para CPU de manera secuencial y para GPU secuencial y paralela, en donde se paralelizaron dos parte del algoritmo, las cuales son: i) El recorrido de las hormigas y ii) La actualización de feromonas, utilizando CUDA (Compute Unified Device Architecture).

Antes de ejecutar el algoritmo, tanto en CPU como en GPU, se procedió a crear una función de tiempo, la cual tiene como objetivo entregar el tiempo de ejecución, para posteriormente realizar las evaluaciones correspondientes y a la vez, se realizaron ciertas modificaciones al algoritmo en GPU que presentaba funciones de CUDA obsoletas.

Para las diferentes pruebas realizadas, tanto en CPU como GPU, se modificaron algunos parámetros de entrada del algoritmo los cuales se detallan en el cuadro 1, para luego poder comparar los resultados.

Parámetros	**Valor**	**Descripción**
CITIES	25 - 50 - 100	Cantidad de ciudades
ANTS	100 - 300 - 500 700 - 900 - 1000	Cantidad de hormigas

Cuadro 1. Parámetros

El máximo de hormigas a considerar fue de 1000 por una de las condiciones del algoritmo en CUDA , ya que cada hormiga ocupaba un núcleo al momento de la paralelización.

Los parámetros de distancia máxima y horas máximas se mantuvieron en 100 y 50 respectivamente, el coeficiente que controla la contribución de cada feromona (Alpha) fue 1 y la tasa de evaporación de las feromonas (Rho) fue de 0,5. Estos valores se encontraban definidos en el algoritmo.

Al momento de ejecutar los algoritmo, se utilizaron archivos de texto que fueron generados por un map generador, codificado en Ruby, que contenía información aleatoria para cada mapa con N cantidad de ciudades.

5. Resultados

Los siguientes gráficos muestran los tiempos de ejecución para 25, 50 y 100 ciudades, con un rango de hormigas entre 100 y 1000, tanto para la implementación en CPU y GPU.

En el gráfico de la figura 1 se observa que el menor tiempo de ejecución corresponde a las pruebas realizas en la CPU y al llegar a las 900 hormigas los tiempos aproximadamente se igualan y disminuye el tiempo en GPU.

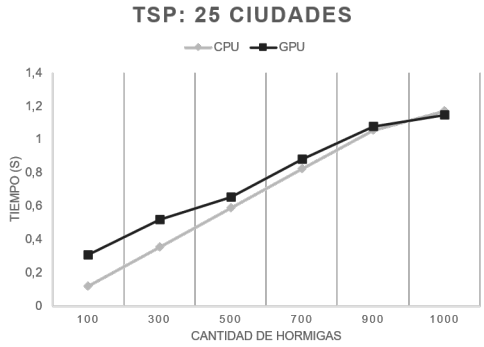

Figura 1. Tiempo de ejecución Vs Cantidad de hormigas

El en gráfico correspondiente a la figura 2, se puede observar que al aumentar la cantidad de hormigas la implementación en GPU tiene un menor tiempo de ejecución, donde se destaca una diferencia significativa, entre CPU y GPU con 300 hormigas.

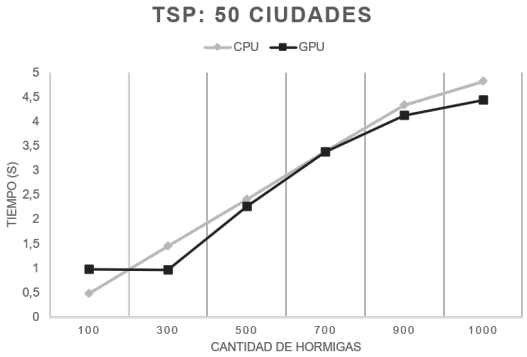

Figura 2. Tiempo de ejecución Vs Cantidad de hormigas

En La figura 3 correspondiente a las 100 ciudades, el tiempo en la GPU empieza a disminuir aproximadamente con una cantidad de 500 hormigas, en comparación a la CPU que el tiempo de ejecución aumenta.

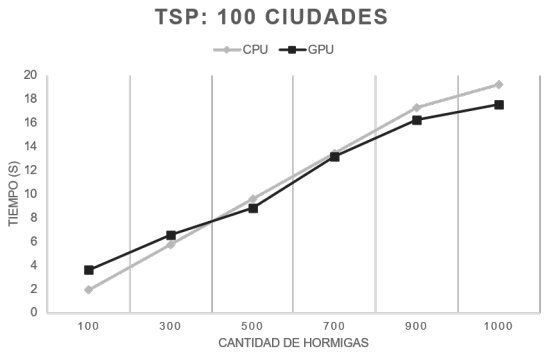

Figura 3. Tiempo de ejecución Vs Cantidad de hormigas

La figura 4 mediante un gráfico de barras, se comparan los diferentes speedup. Donde el mejor speedup se observa para 50 ciudades con 300 hormigas.

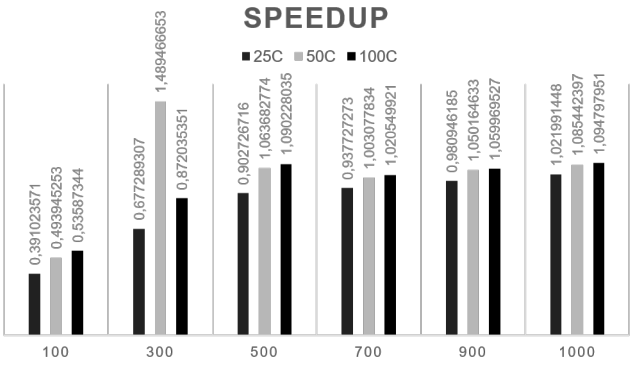

Figura 4. Speedup

6. Conclusiones

Al analizar la diferencia de tiempo de ejecución entre la CPU y GPU, la diferencia es mínima, ya sea en segundos o mili-segundos, pero a pesar de esto, los resultados se obtienen de forma eficaz y rápidamente.

Se debe considerar que al momento de tener una cantidad pequeña de ciudades, por ejemplo el caso de 25 ciudades, es preferible utilizar la implementación en CPU, ya que la comunicación entre CPU y GPU se terminan transformando en un cuello de botella, lo cual hace que la GPU tome tiempos de ejecución mayor.

Como el algoritmo en GPU no esta totalmente paralizado, lo que conlleva a que el traspaso de información entre GPU y CPU sume tiempo adicional, seria recomendable paralizar lo mas posible el algoritmo con el fin de disminuir dicho traspaso de información y así reducir el tiempo, para alcanzar rendimientos mayores.

En las empresas, ya sea de venta de productos o solo de distribución, deben considerar las mejores rutas para sus recorridos de entrega, por lo cual el TSP es uno de los problemas que esta presente día a día en ellas y las formas de resolverlo, que algunas empresas tienen, no son las mas adecuadas o eficaces en relación a los tiempos de obtención de los recorridos adecuados y los costos asociados. Por lo cual, una adecuada implementación de algoritmos para resolver este problema puede aportar significativamente a la empresa, ya sea reduciendo los costos por viajes o recorridos mal planeados, y los tiempos de respuestas para obtener dichos recorridos.

Referencias

1. M. Dorigo, M. Birattari, and T. Stützle. Ant colony optimization. *Computational Intelligence Magazine, IEEE*, 1:28–39, 12 2006.
2. F. Luger George. Artificial intelligence structures and strategy for complex problem solving. *Pearson Addison Weasley*, 2009.
3. R. Mehrotra. Implementing the travel salesman problem using cuda in gpu.
4. R. Mehrotra. Travel salesman problem using genetic algorithm ant colony optimization. URL https://github.com/rachitmehrotra1/TSP-GPU.