Solving Large Batches of Traveling Salesman Problems with Parallel Computing

1st Yahaira Gomez
Universidad Católica San Pablo
yahaira.gomez@ucsp.edu.pe

2nd Christopher Yquira
Universidad Católica San Pablo
christopher.yquira@ucsp.edu.pe

***Resumen*—En este trabajo, se pretende replicar el trabajo propuesto en clase, mostrar los resultados obtenidos y algunos cambios que hicimos al proyecto original. Primero describimos y comparamos dos implementaciones de solucionadores en serie y paralelos para grandes lotes de problemas de viajante de comercio utilizando la heurística de Lin-Kernighan (LKH). La implementación de solucionador paralelo llegó a ser útil solo en casos donde se tenía que resolver simultáneamente muchas instancias de TSP de tamaño mediano a grande. Sin embargo, en caso de pocos datos las implementaciones seriales son más eficientes. Los resultados obtenidos indicaron que la computación paralela que usa hiperprocesamiento para resolver TSP de 150 y 200 ciudades puede aumentar la utilización general de los recursos informáticos hasta en un 25 % en comparación con la computación de un solo subproceso y los tiempos que demora cada implementación con diferentes tamaños de ciudades.**

***Index Terms*—Computación paralela, Heurística de Lin-Kernighan (LKH, problema del agente viajero (TSP), problema de lotes**

I. Introducción

En los últimos años, gracias a la llegada de los procesadores multinúcleo, las computadoras han ganado más potencia al proporcionar más ciclos de reloj por CPU. Sin embargo, la mayoría de las implementaciones de software siguen siendo programas de un solo procesador ineficientes. Escribir un programa paralelo eficiente y escalable es una tarea compleja, pero existen librerías que brindan el poder de la computación paralela con cambios simples en la implementación. Resolver grandes lotes de problemas del agente viajero (TSP) es un problema que puede ser resuelto haciendo uso de paralelismo debido a la independencia en sus operaciones (cada instancia de TSP se puede resolver llamando a un TSP Solver en paralelo). En este trabajo se replicó la propuesta hecha por S.G. Ozdena, A.E. Smitha y K.R. Gueb, al cual nos referiremos como ”trabajo original”, haciendo unos ligeros cambios que se explicarán a continuación. El trabajo original propone usar dos heurísticas: Lin-Kernighan (LKH) y Concorde. Sin embargo, nosotros optamos por usar LKH debido a su accesibilidad de información e implementación. Originalmente también se propone implementar el proyecto en C, pero decidimos usar C++ debido a que tiene herramientas que mejoran el performance de las tareas paralelas.

El presente trabajo genera una base de datos con direcciones aleatorias de n ciudades con direcciones desde un punto ’x’ a un punto ’y’, esta implementación se hizo con Python y nosotros elegimos el número de dichas ciudades. El desafío es encontrar la ruta más óptima entre estos dos puntos haciendo uso de un modelo secuencial y paralelo.

Se implementaron 2 modelos para resolver dicho problemas haciendo uso de LKH. En el caso del modelo serial se envía un conjunto de puntos ’x’ e ’y’ a una función de contenedor uno por uno, en serie. Mientras que el modelo paralelo usa hyper-threading, ejecuta las mismas acciones que el modelo secuencial solo que de manera simultánea.

El resto del arrtículo está dividido de la siguiente manera: La sección 3 muestra parte de la implementación propuesta, en la sección 4 mostramos los resultados obtenidos. Por último, en la sección 5 damos algunas conclusiones sobre el proyecto propuesto.

II. Trabajo realizado

El ejemplo que nos presentan los autores, utiliza una red de distribución a gran escala, esta red incluye múltiples empresas de fabricación. Los proveedores se encargan de entregar las piezas directamente a las fábricas o a través de almacenes, pero la entrega directa del proveedor a un almacén o una fábrica sería ineficiente. Por tanto, un camión será el que visita a varios proveedores y recoge todas las piezas.

Es por ello que se debe calcular el costo total de distribución. Esto quiere decir que debemos crear varias rutas de distribución, con diferentes condiciones para encontrar la mejor red de distribución a gran escala.

En otras palabras, debemos calcular las distancias de ruta más cortas entre ubicaciones de almacenamiento y entre ubicaciones de almacenamiento a una ubicación de recogida y depósito, es decir, una ubicación donde comienza y termina un recorrido TSP. Luego, debemos crea un TSP basado en los pedidos de productos, es decir, cada pedido es un recorrido de selección o un TSP. Este pedido será enviado al solucionador LKH y leerá la distancia TSP resultante de este solucionador. Para encontrar los mejores diseños, debemos Considerar numerosos pedidos de productos que representan la actividad de preparación de pedidos del almacén. Identificamos el diseño que da el costo de viaje promedio mínimo.

La metodología para resolver grandes lotes de TSP incluye tres pasos principales. Primero, se establecen las condiciones específicas del problema (por ejemplo, una estructura de diseño de almacén o una estructura de red de distribución). Por lo general, esto implica crear ubicaciones y calcular distancias entre ubicaciones. Es por ello que necesitamos evaluar el costo

total o promedio de los TSP para la configuración dada del primer paso. Por lo tanto, para el segundo paso, se crean y evalúan varios TSP con un solucionador de TSP. Finalmente, se calcula el costo total o promedio de los TSP.

II-1. Implementación serial: En este caso, manteniendo el mismo ejemplo de los autores, debemos enviar un conjunto de pedidos a una función de contenedor uno por uno en serie. Esto quiere decir que, para cada pedido, buscamos los productos con sus ubicaciones en el almacén y generamos una matriz de distancia que contiene solo los artículos de este pedido en particular. Esto debido a que las distancias de ruta más cortas ya se calcularon en la etapa anterior, solo generamos una submatriz de la matriz de distancia principal que contiene las distancias de ruta más cortas de todos los pares entre cada ubicación de almacenamiento, la ubicación de recolección y depósito en el almacén. Con base en esta matriz de distancia, generamos un archivo en el formato de archivo TSP, para así poder llamar a la función Concorde o LKH para poder resolver el archivo correspondiente, leerlo y mantener la distancia después de la ejecución. Finalmente, borramos el archivo TSP generado, junto con cualquier archivo generado de Concorde/LKH y continuamos con el siguiente pedido. La siguiente figura muestra el proceso de la ejecución en serie a dicho problema, esta ejecución finaliza cuando se evalúan todos los pedidos.

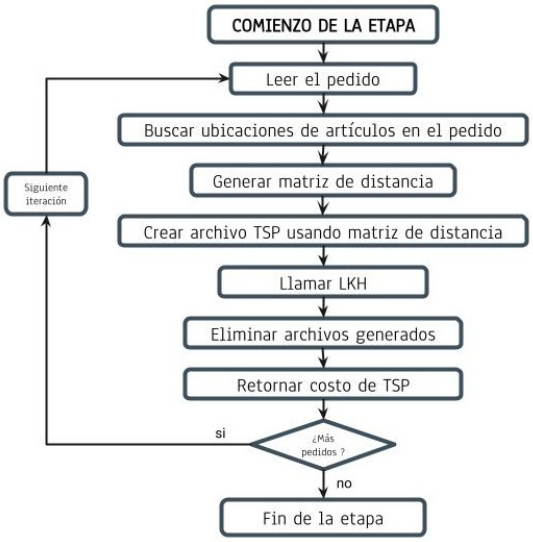


Figura 1. Implementación serial de LKH

II-2. Implementación paralela: Es importante darnos cuenta de la mejor forma para poder aplicar la paralelización, no siempre todo puede ser paralelizado. Si observamos con detalle, las operaciones dentro del bucle son independiente de cualquier otra operación externa, aquí es donde podemos utilizar For paralelo y enviar un conjunto de órdenes a la función contenedora en paralelo. El resto de operaciones son iguales a una ejecución en serie, pero se realizan en paralelo hasta completar todas las órdenes. En siguiente figura, los bloques con líneas de un color más suave representan las operaciones que se realizan simultáneamente en paralelo, mientras los otros bloques se realizan en secuencial.

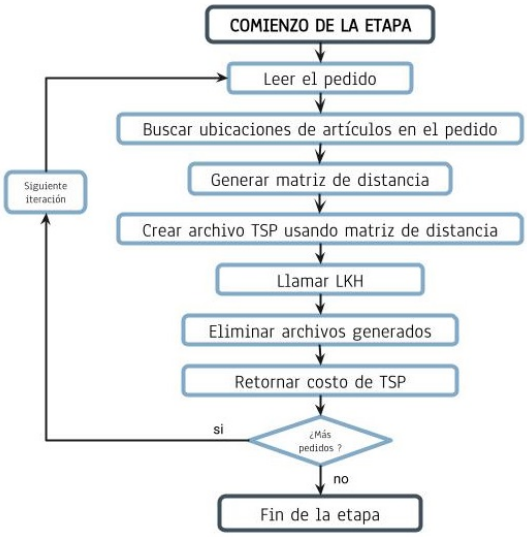


Figura 2. Implementación paralela de LKH

III. Resultados

Se implementaron ambos modelos y se comparó su rendimiento en ambos casos, el código puede ser encontrado en el siguiente link al repositorio. Primero se tiene un archivo python que nos ayuda a generar las direcciones de las ciudades, que recibe la cantidad de ciudades a crear y el número de batches o lotes. Para la evaluación de estos modelos se decidió usar 100 ciudades con 8, 16, 32, 64, 128 lotes, tal y como se ve en la Figura 3. Dentro de cada lote tenemos 100 ciudades, cada una con su dirección 'x' e 'y', para un mejor entendimiento vea la Figura 4

Una vez que tenemos los datos de cada ciudad, hacemos uso de la implementación realizada en el punto 3 y los resul-

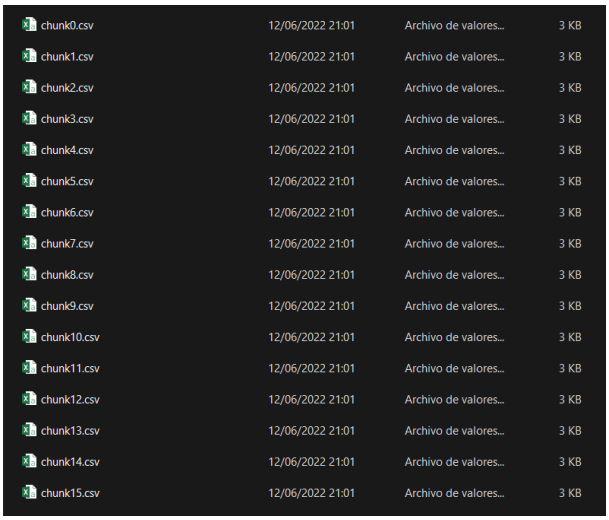
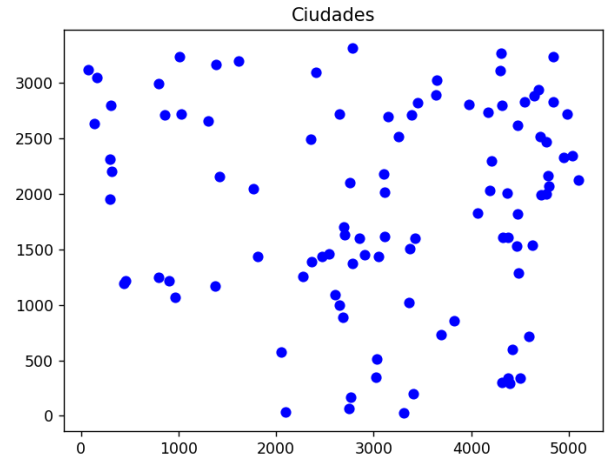


Figura 3. 16 lotes con posición de ciudades



tados muestran que la implementación paralela tiene mejores resultados a partir de los 16 batches. Hay que tener en cuenta que para la implementación paralela se usó los núcleos de la PC en la que se realizó dicha implementación (8 núcleos). Los resultados se aprecian en la Figura 5.

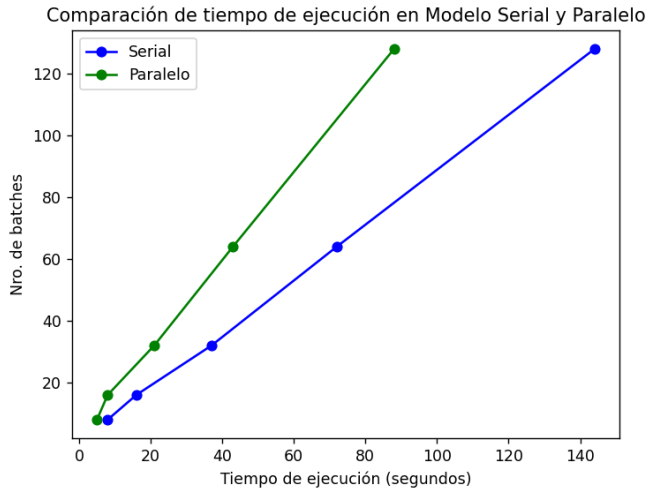


Figura 4. Ciudades

IV. Conclusiones

Gracias a los resultados obtenidos, podemos ver que, tanto en la teoría como en la práctica, las técnicas de cómputo paralelo pueden reducir significativamente el tiempo de cómputo general y aumentar la utilización de la CPU para resolver grandes problemas, como lotes de TSP. Esto se hace mediante métodos de clase paralelos simples y efectivos utilizando el lenguaje de C++. Resolver grandes lotes de TSP es el paso más intensivo desde el punto de vista computacional en muchas aplicaciones que involucran enrutamiento. Nuestros resultados muestran que el uso de métodos de clases paralelas de C++ es una forma simple, eficaz y escalable de paralelizar la resolución de grandes lotes de TSP. El programador puede escribir funciones de envoltorio para LKH, e implementar "bucles for paralelos"para aprovechar los procesadores multinúcleo.

Nuestros resultados también muestran que, tanto para datos reales como para datos generados, un algoritmo de programación como LPT funciona mejor que un método ingenuo como EDR, aunque el método utilizado para estimar los tiempos de procesamiento de los TSP no es muy preciso (tamaño de TSP, en este caso). La heurística de Lin-Kernighan (LKH) es teóricamente mucho más rápida que Concorde en promedio en ejecuciones únicas, paralelas, distribuidas por 2 computadoras y distribuidas por 3 computadoras para resolver 10 000 TSP de 200 ciudades, respectivamente.

En general, un ordenador que utiliza la programación paralela puede aprovechar mejor sus recursos para procesar y resolver grandes problemas como el problema presentado. La mayoría de los ordenadores modernos tienen un hardware que incluye múltiples núcleos, hilos o procesadores que les permiten ejecutar muchos procesos a la vez y maximizar su potencial informático.

Adicionalmente, un trabajo a futuro muy interesante a tener en cuenta es la implementación de paralela y distribuida, ya que, en este caso, tendríamos una arquitectura master-esclavo para realizar una computación distribuida, donde nos basaríamos principalmente en la misma estructura de paralelo, realizando un cambio al momento de cada pedido, llamaríamos un nuevo proceso al que denominaríamos esclavo. En general tendríamos un gran numero de esclavos, los cuales realizaran las tareas por separado y al finalizar su tarea, le enviarían la respuesta al master y de esa forma resolveríamos el problema de una forma mucho mas eficiente.

Referencias

[1] S.G.Ozdena, A.E.Smitha y K.R.Gue, *Solving large batches of traveling salesman problems with parallel and distributed computing*, Septiembre, 2017.