
# Programación Paralela

## Práctica 3. Implementación distribuida de un algoritmo de equilibrado dinámico de la carga usando MPI

Juan Francisco Díaz Moreno

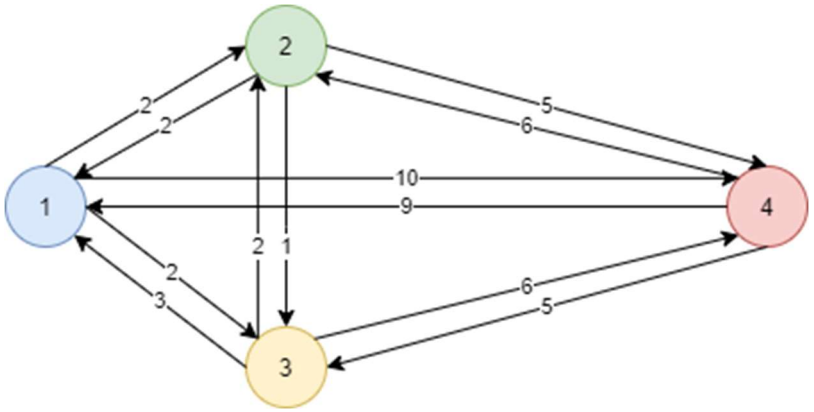
9 de junio de 2020

# 1. Índice

## 2. Introducción

En esta tercera práctica, se trabaja una versión del Problema del Viajante de Comercio (*Traveling Salesman Problem* – TSP). Este se basa en la obtención de una solución optimizada que permita a un comerciante realizar un recorrido por una serie de ciudades con el menor coste posible, partiendo de un grafo dirigido compuesto de un conjunto de vértices (ciudades) y arcos etiquetados (distancias entre ciudades).

Un ejemplo de grafo que siga esta estructura sería el siguiente, que representa los caminos entre 4 ciudades:




El problema va a ser resuelto utilizando un algoritmo de acotación y ramificación (*Branch-and-Bound*), que dinámicamente construye un árbol de búsqueda cuya raíz es el problema inicial y sus nodos hoja son caminos entre todas las necesidades.

Para resumir un poco cómo trabaja el algoritmo, diremos que iremos explorando distintas combinaciones entre las ciudades obteniendo resultados. Para ahorrar trabajo inútil, se añade el concepto de cota superior, que se actualizará en todo momento con el coste de la solución más óptima que se vaya encontrando.

Esta cota permitirá que el árbol vaya podándose, es decir, los nodos intermedios cuyo coste supere la cota dejarán de ser expandidos, de tal forma que nos ahorramos el cómputo de soluciones peores a la que ya tenemos, pues solo buscamos la mejor de todas.

En el material de la práctica se proporciona una versión secuencial del problema. En la práctica se pide desarrollar una versión que utilice MPI para distribuir dinámicamente la carga de trabajo entre distintos procesos.

Esta versión trabajará con los procesos situados en una estructura de anillo, en el que cada proceso inicialmente se comunicará con el siguiente y con el anterior, pudiendo en algunos casos comunicarse con otro cualquiera.

Para ello, podríamos indicar 3 aspectos importantes de la versión distribuida: el equilibrado de cota, la terminación elegante y la difusión de la cota superior.

## 3. Equilibrado de carga

En este programa todos los procesos tienen el mismo rol. No existe ni un maestro ni varios esclavos. Todos los procesos se ejecutan de la misma manera y pueden realizar todas las tareas.

Uno de los procesos, en este caso el 0, es el encargado de leer la matriz inicial y de mandarla al resto de procesos, ya que todos la van a necesitar al completo.

El nodo inicial estará almacenado en este primer proceso. Este empezará a trabajar con él, expandiéndolo y avanzando en la estructura de nodos.

Cuando un nodo no tenga trabajo, pedirá trabajo al siguiente. Si este puede mandarle (teniendo al menos 2 nodos), lo hará. Si no, reenviará la petición al siguiente proceso del anillo. Cuando la petición llegue a un proceso que pueda enviar nodos, este proceso se los enviará directamente al proceso que realizó la petición.

## 4. Terminación elegante

Para que el programa termine, hay que establecer una manera para que los procesos se den cuenta de que no hay más nodos que explorar, ya que en otro caso se quedarían pidiendo trabajo a otros nodos sin fin.

Este algoritmo se pondrá en marcha cuando un proceso reciba su propia petición de trabajo, entendiendo que ningún otro proceso puede darle nodos y que, por lo tanto, no queda trabajo que hacer. Si esto ocurre, el nodo pasa a un estado pasivo y comienza el proceso de detección de fin.

Una primera aproximación se basaría en que cuando un proceso llegue a ese estado, mandaría al proceso anterior un token. Ese token servirá para comprobar si se ha terminado el trabajo y por defecto empezará teniéndolo el proceso 0.

Cuando un proceso reciba el token y esté en estado pasivo, reenviará el token al proceso siguiente. Si sigue en estado activo, esperará a estar pasivo para reenviarlo.

Cuando el token vuelva al proceso 0, se asumiría que ya no quedan nodos por explorar, por lo que el programa podría terminar. Pero esto no siempre es así. Puede suceder que un proceso en estado pasivo haya reenviado el token, pero posteriormente reciba trabajo de otro proceso, por lo que el programa no podría terminar a pesar de que el token vuelva al proceso 0.

Para evitarlo, se hace uso del algoritmo de terminación de Dijkstra. Para solucionar el problema anterior, tanto los procesos como el token tendrán dos colores: blanco o negro. Inicialmente, tanto los procesos como el token serán de color blanco.

El token se irá enviando entre los procesos como se ha descrito anteriormente, con la peculiaridad de que si pasa por un proceso de color negro quedará del mismo color. En ese momento, el proceso en cuestión volverá a ser de color blanco.

Cuando un proceso envíe trabajo a otro de mayor identificador, pasará a color negro. Esto implica que siempre que llegue el token de color negro al proceso 0, este considera que, aunque haya regresado, no se puede terminar aún la ejecución, por lo que si sigue en estado pasivo volvería a enviar el token al proceso anterior, pero de color blanco.

La ejecución podrá terminar cuando el token regrese al proceso 0 de color blanco, pues eso indicaría que todos los procesos serían de color blanco y que no habría más nodos que evaluar.

Cuando se llegue a esta situación, el proceso 0 mandará un mensaje de finalización que irá rotando entre los procesos “matándolos”.

Este mensaje se aprovechará para decidir la mejor solución entre las que ha encontrado cada proceso. Cuando un proceso reciba dicho mensaje, comprobará si es mejor la solución que ha recibido en el mensaje de finalización o la suya, y enviará al siguiente proceso el mensaje con la mejor solución de las dos. Posteriormente, acabará su ejecución.

Cuando el mensaje de finalización vuelva al proceso 0, lo hará con la mejor solución de todas, por lo que podrá mostrarla y finalizar él también la ejecución.

## 5. Difusión de la Cota Superior

Este apartado, a pesar de ser opcional, pues el programa funciona perfectamente sin él, permitirá obtener mejoras sustanciales en la eficiencia del programa, como veremos posteriormente.

Cada proceso tendrá una cota local para realizar las podas, pero es posible que otro proceso tenga una cota mejor, por lo que se estarían explorando nodos innecesarios.

Por ello, los procesos también podrán compartir sus cotas para encontrar la más baja en cada momento y podar el mayor número de nodos de forma global.

Como condición inicial, cada proceso podrá poner en circulación una cota al mismo tiempo. Aunque el proceso encuentre una solución mejor, no podrá comunicarla hasta que reciba su cota anterior. Esto se hace para disminuir la cantidad de mensajes que se envían.

Cuando un proceso reciba una cota, comprobará si es mejor la suya o la que ha recibido, estableciendo como cota local la mejor de las dos y enviándola al siguiente proceso.

## 6. Tablas de Tiempo e Iteraciones

Vamos a representar en tablas los tiempos de ejecución y las ganancias en velocidad del algoritmo paralelo frente al algoritmo secuencial. Para ello, ejecutaremos el algoritmo paralelo con 2 y 3 procesos, y con y sin difusión de cota.

Para la obtención de los tiempos de ejecución más adecuados, se ha ejecutado cada versión 10 veces, proporcionándose en tablas la media de ellas.

En esta primera tabla se muestran los datos de la versión secuencial junto a las ejecuciones de la versión paralela con y sin difusión de cota con **2 procesos**:

| N | $T_{SEQ}$ | $T_{PAR2\_SIN\_DIF}$ | $S_{PAR2\_SIN\_DIF}$ | $T_{PAR2\_CON\_DIF}$ | $S_{PAR2\_CON\_DIF}$ |
|---|---|---|---|---|---|
| 10 | 0.001972425 | 0.001071865 | 1.84 | 0.0008397 | 2.35 |
| 20 | 0.04985735 | 0.03643641 | 1.37 | 0.02357574 | 2.11 |
| 30 | 0.1899331 | 0.1624856 | 1.17 | 0.10261513 | 1.85 |
| 35 | 2.397633 | 1.53107 | 1.57 | 1.0055669 | 2.38 |
| 40 | 6.364473 | 2.624824 | 2.42 | 3.601493 | 1.77 |

A continuación, se muestran los datos de la versión secuencial junto a las ejecuciones de la versión paralela con y sin difusión de cota con **3 procesos**:

| N | $T_{SEQ}$ | $T_{PAR3\_SIN\_DIF}$ | $S_{PAR3\_SIN\_DIF}$ | $T_{PAR3\_CON\_DIF}$ | $S_{PAR3\_CON\_DIF}$ |
|---|---|---|---|---|---|
| 10 | 0.001972425 | 0.000862886 | 2.29 | 0.000632784 | 3.12 |
| 20 | 0.04985735 | 0.03022357 | 1.65 | 0.01799092 | 2.77 |
| 30 | 0.1899331 | 0.1611323 | 1.18 | 0.08328933 | 2.28 |
| 35 | 2.397633 | 1.257131 | 1.91 | 0.654818 | 3.66 |
| 40 | 6.364473 | 2.575762 | 2.47 | 1.62794 | 3.91 |

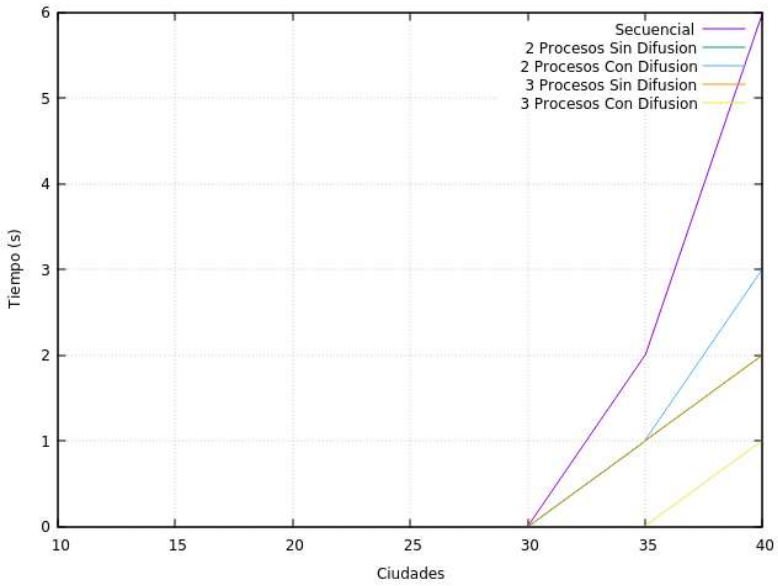
Finalmente, se incluyen unas tablas en las que se puede comprobar el número de iteraciones que realiza cada proceso en cada caso, primero sin difusión y luego con difusión:

| N | SEQ | $PAR_{2P}$ | | $PAR_{3P}$ | | |
|---|---|---|---|---|---|---|
| | P0 | P0 | P1 | P0 | P1 | P2 |
| 10 | 207 | 138 | 116 | 121 | 104 | 109 |
| 20 | 3755 | 2374 | 2322 | 2027 | 1795 | 1973 |
| 30 | 6957 | 5580 | 5544 | 5413 | 5472 | 5355 |
| 40 | 71107 | 38119 | 38309 | 32166 | 32148 | 32283 |
| 50 | 158556 | 56618 | 58428 | 56894 | 56382 | 57399 |

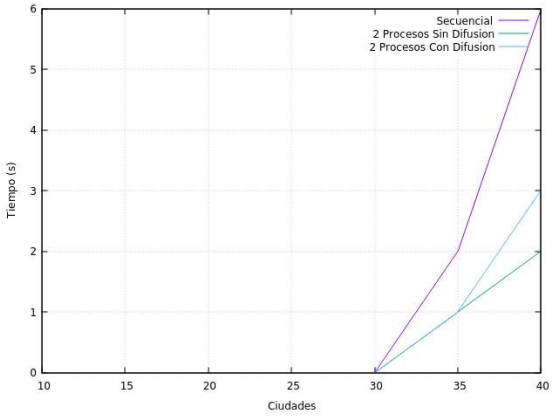
| N | SEQ | $PAR_{2P}$ | | $PAR_{3P}$ | | |
|---|---|---|---|---|---|---|
| | P0 | P0 | P1 | P0 | P1 | P2 |
| 10 | 207 | 114 | 100 | 84 | 67 | 73 |
| 20 | 3755 | 1599 | 1586 | 1062 | 1207 | 1033 |
| 30 | 6957 | 3585 | 3651 | 3127 | 3286 | 3105 |
| 40 | 71107 | 26099 | 26005 | 15741 | 16108 | 15587 |
| 50 | 158556 | 80824 | 80417 | 34703 | 34624 | 34304 |

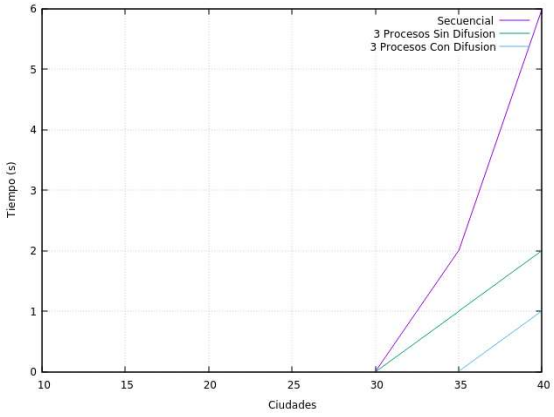
# 7. Gráficas

A partir de los datos obtenidos de las distintas ejecuciones del programa, tanto secuencial como paralelo con y sin difusión con 2 y 3 procesos, se han elaborado las siguientes gráficas para representar de forma visual la comparativa:



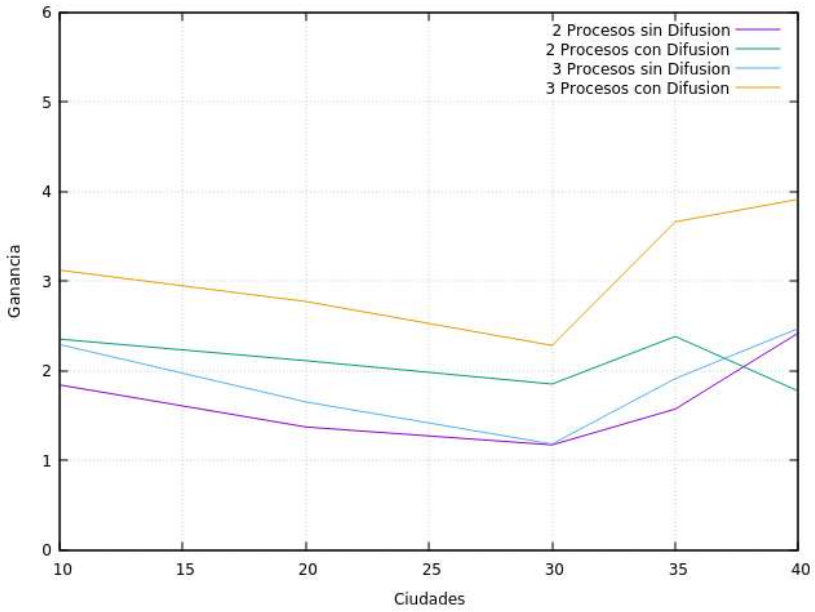

Mostramos también estas gráficas separando las ejecuciones con 2 y 3 procesos:




Se puede ver con claridad que las versiones con difusión de cota son mejores que las que no la utilizan, y que, a más procesadores, más paralelismo y por lo tanto, menor tiempo de ejecución.

Finalmente vamos a presentar una comparación de las ganancias en velocidad de las distintas ejecuciones:




Puede verse como (salvando la excepción de la difusión con 2 procesadores, de la que hablaremos en las conclusiones), la difusión de las cotas es más eficaz incluso que el añadir más procesadores en caso de 2 a 3.

# 8. Conclusiones finales

Mediante la realización de esta práctica hemos podido comprobar de nuevo la utilidad de los algoritmos paralelos frente a los secuenciales, utilizando de nuevo MPI.

Hemos comprobado como el hecho de dividir el trabajo dinámicamente entre los procesos mejora bastante la eficiencia de los programas. Pero si además utilizamos la lógica para implementar todos los beneficios del programa secuencial en el paralelo (como es la difusión de cota), es cuando obtenemos un verdadero beneficio.

Finalmente, hay un dato sorprendente al que no le he podido encontrar solución: el caso de los tiempos con 40 ciudades y 2 procesos. Es el único caso en el que no es sólo que la difusión no mejore los tiempos, si no que inexplicablemente los empeora, provocando que cada proceso realice más iteraciones.