Pontificia Universidad Católica Madre y Maestra

Campus: Santo Tomás de Aquino

Departamento de Ingeniería

Programación II

Mini-proyecto final

Javier Falcón (2016-5265)

Manuel Molina (2016-5468)

Santo Domingo, 05/12/2018

Introducción

Durante el transcurso de esta materia nos hemos enfocado en el estudio de la concurrencia a nivel de software y cómo esto afecta el *performance* de los algoritmos en su ejecución. Hemos trabajado y analizado algoritmos implementados bajo un esquema de memoria compartida y algoritmos fundamentados en el paso de mensajes. Para este experimento, se nos pide estudiar y observar el comportamiento de algoritmos de ordenamiento en paralelo bajo un esquema de paso de mensajes, dichos algoritmos son: *Odd-Even Transposition Sort* y *Bitonic-Sort.* Cabe destacar que los puntos de referencia para el análisis del comportamiento están relacionados con los conceptos analíticos de la Ley de Ahmdal y su cálculo de *speedup*.

Un aspecto interesante de este experimento es poder comparar las actuaciones de dos algoritmos paralelos cuyas implementaciones son bastante distintas, pretendiendo buscar una ilustración de las diferencias entre lo que se plantea teóricamente y lo que sucede en la práctica. Prevemos que la comunicación excesiva puede ser un problema al momento de evaluar el speedup y que esto podría costar una diferencia entre el ordenamiento paralelo y un ordenamiento secuencial, tal como lo es QuickSort.

Solución de la problemática

Para la realización del algoritmo de *Odd-Even Transposition Sort* fue necesario consultar documentación para lograr detallar cómo opera este algoritmo. Sabemos que algoritmo de ordenamiento está basado en la técnica de Bubble Sort en la cual se comparan dos números y se intercambian si el primero es más grande que el segundo. Al ser un algoritmo paralelo que utiliza paso de mensajes, le denominamos a esta rutina *compare-exchange,* debido a que cada proceso debe comparar su valor interno con el de su vecino y, de ser necesario, intercambiar valores. El *Odd-Even Transposition* utiliza dos fases para el ordenamiento: la fase par establece que sólo los procesos pares pueden intercambiar números con su vecino de la derecha; la fase impar se encarga de los intercambios de los procesos impares con su vecino derecho. Así sucesivamente hasta conseguir el arreglo ordenado.

En el caso del Bitonic Sort, no pudo llegar a ser implementado; sin embargo, entendemos que funciona de una manera diferente. Este algoritmo trabaja con secuencias bitónicas y para entenderlo consultamos las propiedades de este tipo de frecuencias. El algoritmo toma pares de números adyacentes para que hagan compare and exchange. El resultado de esto será una secuencia bitónica más larga, en la cual la secuencia creciente y la decreciente serán dos veces más grande que en la etapa anterior. Con esto, llegará un punto en donde solo quedará una secuencia decreciente, indicando que el algoritmo ha terminado de ordenar. Entre las dificultades que hubo para implementar este algoritmo tenemos la dificultad conceptual para entender el manejo de las reducciones del intercambio con cada fase de ordenamiento.

Resultados

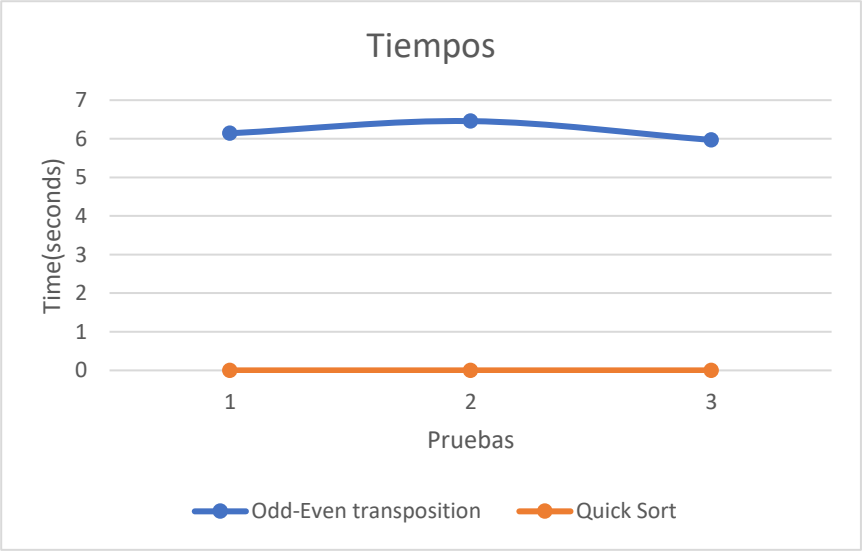
Primero se mostrarán los resultados de los cálculos analíticos que tomaremos como referencia más adelante en la presentación de los resultados experimentales.

Speed up teórico
3,321

Utilizando la ecuación de estimación de *speedup* para el *Odd-Even Transposition* conseguimos que el *speedup* analítico es de un total de 3,321. Sin embargo, los resultados experimentales tendrán un comportamiento diferente.

A continuación se presentarán los resultados de la medición de tiempo para los algoritmos de *Odd-Even Transposition* y *Quicksort.* Debido a que el experimento fue ejecutado utilizando la misma cantidad de procesos en cada intento, los aspectos a analizar anteriormente mencionados se vuelven inconsistentes.

Odd-Even transposition	Quick Sort
6.14185	0.000018542
6.45802	0.000018821
5.97032	0.000018767



Como se puede observar, el algoritmo secuencial de Quicksort presentó un tiempo de ejecución abismalmente menor que el algoritmo de ordenamiento paralelo.

En el caso del Quicksort obtuvimos un tiempo promedio de 0.00001871, mientras que para el *Odd-Even Transposition* se obtiene un tiempo promedio de 6.190063333.

Siguiendo con la discusión de los resultados, consideramos que el experimento no puede modelarse con lo establecido en la Ley de Amhdal debido a que la cantidad de procesadores fue fija en todas las ejecuciones. Esta es la razón por la que el speedup experimental dio un valor muy cercano a cero.

- **Diferencias con Quicksort:** los resultados de tiempo de ejecución contradicen la visión original que teníamos antes de ejecutar el experimento. Podemos atribuir esta amplia diferencia a que no se está utilizando el número óptimo de procesos para el algoritmo, provocando que haya comunicación excesiva que ocasiona un tiempo de ejecución más lento para el algoritmo paralelo.
- **Posibles causas de las diferencias:** Como dijimos anteriormente, esta amplia diferencia con los tiempos de ejecución puede darse gracias a que la granularidad es tan fina, que la comunicación de los procesos se vuelve excesiva, generando un *overhead*. Es posible también que se está comparando un algoritmo paralelo basado en BubbleSort que tiene peor tiempo de ejecución que la implementación secuencial. Sabemos que BubbleSort secuencial es uno de los algoritmos de ordenamiento más ineficiente que hay. Entonces, con esto en mente, podemos decir que se compara uno de los mejores algoritmos de ordenamiento secuencial (Quicksort) con uno de los peores paralelos (Odd-Even).