Análisis de rendimiento de algoritmos paralelos

Joaquín Andrés López Molina josandlopmol@gmail.com
Daniel Mauricio Rodríguez Alpizar danielmau231995@hotmail.com

Estudiantes de Ingeniería en Computación
Tecnológico de Costa Rica, Sede San Carlos, Escuela de Computación

RESUMEN

El desarrollo de aplicaciones de software de alta calidad, en una era de competitividad, es cada día más importante. Uno de los aspectos más relevantes a considerar son los tiempos de respuesta, tomando esto en cuenta, este artículo pretende mostrar mediante ejemplos, el comportamiento que presentan los algoritmos paralelos y la eficiencia obtenida al implementar algoritmos sobre sistemas multicore. Cabe aclarar que el mismo no es una investigación sobre paralelismo, por lo tanto no describe conceptos relacionados con el tema, sino más bien es un análisis de rendimiento para destacar la diferencia en tiempos de ejecución respecto al enfoque secuencial, mostrando al lector su indiscutible potencial.

Palabras clave: *algoritmos paralelos, sistemas multicore, programación concurrente, paralelismo, programación multinúcleo, enfoque paralelo.*

INTRODUCIÓN

Debido a la gran demanda a nivel computacional por la miniaturización de los procesadores, mejorando la capacidad de procesamiento, es que actualmente se cuenta con las tecnologías de multiprocesamiento, ejemplos de ellas son los nuevos procesadores para computadoras que compañías como Intel y AMD han desarrollado [1][2], inclusive hasta los teléfonos móviles cuentan con procesadores multinúcleo, es por tal razón que resulta sumamente importante obtener el mayor provecho de las prestaciones brindadas por la tecnología actual. Esto se puede lograr utilizando la programación paralela o concurrente, la cual puede cambiar la estructura de un problema, pero puede llegar a marcar grandes diferencias a nivel computacional en comparación con el enfoque secuencial, por lo tanto el objetivo principal de este escrito es demostrar las diferencias en los tiempos de respuesta entre ambas implementaciones. Así todas aquellas personas que están familiarizadas con el desarrollo del software, podrán notar las eventuales mejoras en el rendimiento de la aplicación, y de esta manera incentivar la programación paralela, puesto que en la actualidad, a pesar de contar con dichas tecnologías, en el momento de programar se sigue utilizando una implementación secuencial.

MATERIALES Y MÉTODOS

Para llevar a cabo este análisis, se trabajará con dos diferentes algoritmos desarrollados en C# , uno de ellos es un método de encriptación de datos, básicamente este método toma el valor numérico de la tabla ASCII de cada carácter del texto y lo combina con el valor proveniente de una clave, obteniendo un nuevo valor numérico, el cual sustituirá al original, transformando el carácter inicial en otro completamente diferente, los textos utilizados serán los siguientes libros convertidos en archivos de texto: El Origen de las especies de Charles Darwin, este se considera de tamaño grande, cuenta con 1.148.064 caracteres, Hidden

Treasures de Harry A. Lewis de tamaño mediano con 674.033 caracteres y por último se tiene el libro Metamorphosis de Franz Kafka considerado de tamaño pequeño, este posee 118.368 caracteres aproximadamente [3]. El otro programa que se utilizará es uno de los métodos de ordenamiento de arreglos más utilizados, el MergeSort, también conocido ordenamiento por mezcla. Ambos algoritmos están implementados de forma concurrente y secuencial, los mismos serán ejecutados sobre diferentes computadoras, la primera es una Toshiba con un procesador Core i3 (2.40 GHz) con 4 núcleos y 4GB de memoria RAM, luego sobre una computadora de escritorio con un procesador Core i7 (3.40 GHz) con 8 núcleos y 8GB de RAM. Para comprender mejor el comportamiento del enfoque concurrente, se tomará como ejemplo el algoritmo de encriptación, su implementación secuencial toma el texto completo y lo analiza línea por línea, hasta encriptarlo completamente, todo esto sobre un solo núcleo del procesador, como se muestra en la siguiente imagen.

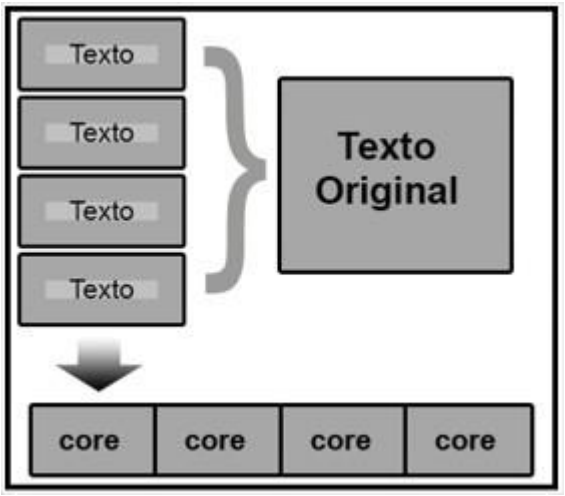

Figura 1. Método de encriptación. [Enfoque concurrente]

Por otro lado, el enfoque paralelo divide el texto original, como se cuenta con una computadora de 4 núcleos, el programa fue diseñado tomando en cuenta este dato, por lo tanto el texto fue dividido en 4 segmentos, posteriormente para aprovechar los 8 núcleos del procesador Core i7 se realizó una modificación al código, dividiendo el texto en 8 segmentos, de esta manera la aplicación valorará el número de núcleos con que cuenta el ordenador, permitiendo al usuario ejecutar el programa tanto para 4 núcleos como para 8. En la implementación concurrente se ejecuta simultáneamente cada segmento sobre cada núcleo del procesador, como se puede apreciar en la siguiente figura.

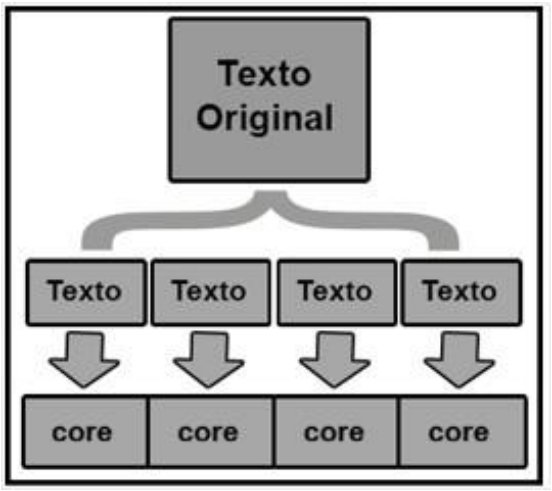

Figura 2. Método de encriptación. [Enfoque secuencial]

Esto se puede lograr utilizando algunas instrucciones del lenguaje C#, tales como el uso del ***Parallel.Invoke(), Parallel.For()*** **y** ***Parallel.ForEach()*** [4][5], de igual manera JAVA también cuenta con librerías que permiten la ejecución de tareas concurrentemente, sin embargo son un poco más difíciles de percibir. Por otra parte, el método de ordenamiento con el MergeSort, tiene una implementación similar [6][7], normalmente el algoritmo de ordenamiento divide el arreglo original en pequeños arreglos, hasta el menor punto posible, luego cuando todo está dividido, se empiezan a ordenar los pequeños arreglos gradualmente, hasta volver a conformar el arreglo, pero esta vez ordenado, tal como se muestra en la

figura 3. Partiendo de este hecho, la solución que se le dio a este algoritmo para poder ejecutarlo sobre un sistema multicore, fue dividir el arreglo principal en cuatro arreglos, aplicándole el MergeSort simultáneamente a las cuatro listas, posteriormente se unen los cuatro arreglos una vez que estén ordenados, siendo esta la etapa final del ordenamiento.

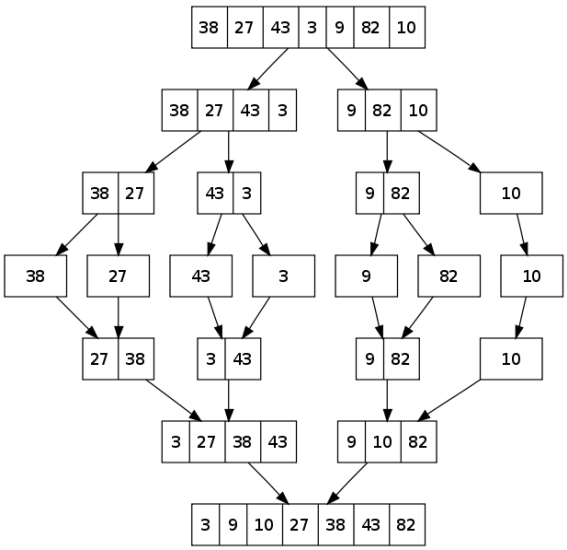

Figura 3. Ordenamiento MergeSort. Tomado de http://geeksquiz.com/merge-sort/

RESULTADOS

El paralelismo divide el problema en subproblemas y los resuelve al mismo tiempo, esto puede marcar grandes diferencias en los tiempos de respuesta, en comparación con el método secuencial, en la *tabla 1* se muestra la duración que tomó la ejecución del método de ordenamiento sobre las dos computadoras, donde se pueden apreciar las grandes ventajas de la programación paralela, en el mejor de los casos se logró obtener una reducción del **53%** aproximadamente en la Core i3, por otro lado con la Core i7, se obtuvo una reducción de tiempo del **71%.** En las *figuras 4 y 5*, mediante los gráficos, se puede apreciar de forma más representativa como varían los tiempos de respuesta del método MergeSort dependiendo del enfoque utilizado. *La tabla 2* al igual que la anterior, posee los datos correspondientes a la duración del método de encriptación, *las figura 6 y 7* corresponden a los gráficos que muestran la ejecución del método de encriptación sobre ambas arquitecturas.

Arquitectura	Tamaño del Array	Secuencial	Paralelo	Reducción
Core i 3 (4 Núcleos)	1.000.000	00:00:00.5180296	00:00:00.2700154	**47,88%**
	10.000.000	00:00:05.9023376	00:00:02.7751588	**52,99%**
	100.000.000	00:01:03.4186273	00:00:30.0527189	**52,62%**
Core i 7 (8 Núcleos)	1.000.000	00:00:00.2964005	00:00:00.1092001	**63,16%**
	10.000.000	00:00:03.4008060	00:00:01.0764018	**68,35%**
	100.000.000	00:00:37.8612665	00:00:10.7796189	**71,53%**

Tabla 1. Tiempos de ejecución del algoritmo de ordenamiento MergeSort (HH:MM:SS).

Arquitectura	Tamaño del Texto	Secuencial	Paralelo	Reducción
Core i 3 (4 Núcleos)	Texto Pequeño	00:00:07.6504376	00:00:01.0740617	**86,02%**
	Texto Mediano	00:04:18.1804534	00:00:43.1340757	**83,30%**
	Texto Grande	00:13:27.5510183	00:02:48:5716417	**79,20%**
Core i 7 (8 Núcleos)	Texto Pequeño	00:00:04.4460078	00:00:01.0296018	**77,03%**
	Texto Mediano	00:02:23.8946528	00:00:14.3052251	**90,07%**
	Texto Grande	00:07:40.6144092	00:00:44.9436789	**90,25%**

Tabla 2. Tiempos de ejecución del algoritmo de encriptación de datos (HH:MM:SS).

Con la encriptación de datos se puede observar resultados aún más sobresalientes, pasar de 13 minutos aproximadamente, a poco más de 2 minutos es un gran logro, considerando los 13 minutos como el 100% del tiempo de respuesta, vemos como la implementación paralela, reduce el tiempo de ejecución a un **20%** aproximadamente, ejecutando el mismo algoritmo sobre la Core i7, los tiempos se reducen aún más, gracias al poder de computo de este procesador, y a la modificación del programa (dividir el texto en 8 segmentos) para sacar el provecho de los 8 núcleos, en el mejor de los casos hubo una reducción del **90,25%** con respecto al enfoque secuencial.

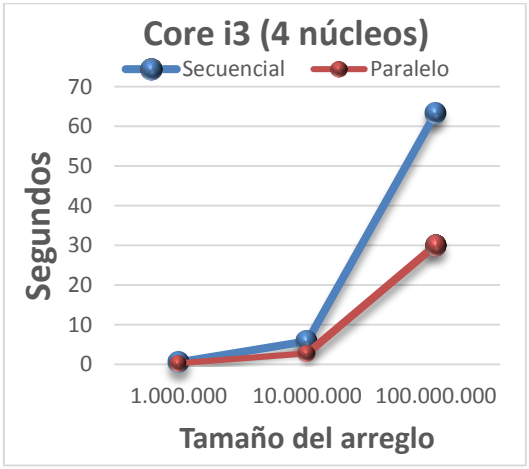

Figura 4. Comportamiento del MergeSort Computador Core i3

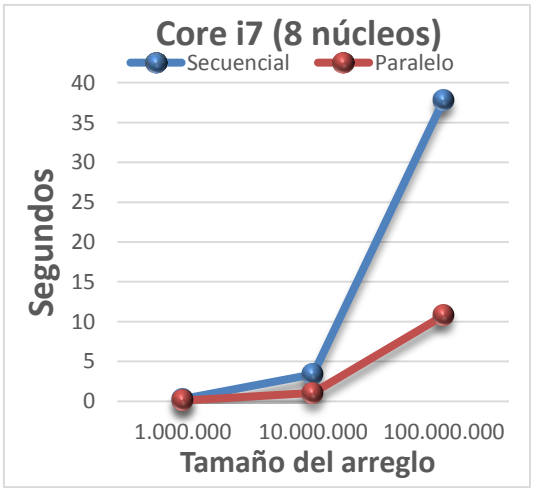

Figura 5. Comportamiento del MergeSort Computador Core i7

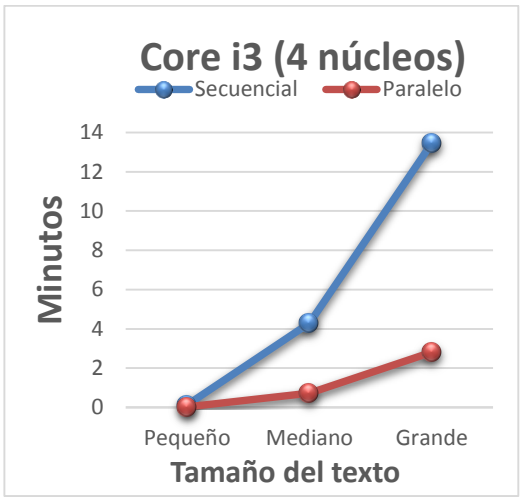

Figura 6. Comportamiento del método de encriptación de datos. Computador Core i3

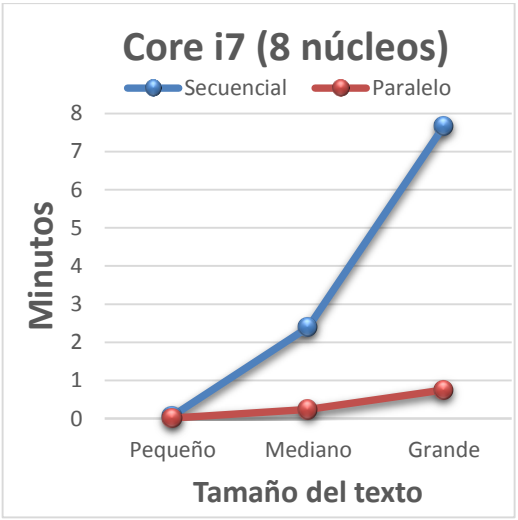

Figura 7. Comportamiento del método de encriptación de datos. Computador Core i7

CONCLUSIONES

Con base en los resultados mostrados anteriormente, se puede apreciar el gran potencial que tiene la programación concurrente en comparación con el enfoque secuencial, para este articulo solo se presentaron dos ejemplos en los cuales los resultados fueron muy significativos, no obstante este enfoque puede aplicarse a una gran cantidad de problemas que hoy día solo se han implementado de forma secuencial,

sin embargo también hay que aclarar que no en todos los casos los resultados son sobresalientes, existen situaciones en donde los tiempos se mantienen similares con ambos enfoques, esto es porque no todos los algoritmos son susceptibles a la programación multinúcleo. Cada año la innovación en las computadoras va creciendo considerablemente, mejorando sus capacidades de procesamiento, además se cuenta con lenguajes de programación, que nos permiten hacer uso de esas prestaciones ejemplos de ellos son Java y C#, entonces ¿Por qué no aprovechar los recursos? Puede que haya que adaptarse a un entorno algo desconocido, sin embargo las ventajas y el rendimiento proporcionado es lo suficientemente bueno como para aceptar este reto.

RECONOCIMIENTOS

Un agradecimiento a Jorge Alfaro Velazco, profesor del Tecnológico de Costa Rica, Sede San Carlos, que gracias a sus críticas constructivas y al apoyo brindado, la redacción de este manuscrito ha sido terminada de la mejor manera posible, de igual forma agradecerle a todos aquellos profesores que de alguna u otra manera colaboraron y brindaron aportes, puntos de vista y recomendaciones que contribuyeron con la elaboración de este artículo.

BIBLIOGRAFÍA

[1] AMD. (s.f.). *AMD*. Recuperado el 14 de Marzo de 2014, de http://www.amd.com/es-xl/products/processors/desktop/fx

[2] Dell. (s.f.). *Dell*. Recuperado el 14 de Marzo de 2014, de http://www.dell.com/learn/es/es/esbsdt1/campaigns/intel-core

[3] Gutenberg. (s.f.). *Gutenberg*. Recuperado el 08 de Marzo de 2014, de http://www.gutenberg.org/ebooks/2009?msg=welcome_stranger

[4] Microsoft Developer Network. (s.f.). *Microsoft Developer Network*. Recuperado el 29 de Marzo de 2014, de http://msdn.microsoft.com/en-us/library/dd460720(v=vs.110).aspx

[5] Microsoft. (s.f.). *Microsoft Developer Network*. Recuperado el 21 de Octubre de 2013, de http://msdn.microsoft.com/en-us/library/dd460705(v=vs.110).aspx

[6] *Stackoverflow*. (16 de Octubre de 2012). Recuperado el 06 de Abril de 2014, de http://stackoverflow.com/questions/12905880/why-is-my-parallel-quicksort-method-slower-than-the-sequential-mthod

[7] Tomy. (02 de Abril de 2011). *PuntoPeek*. Recuperado el 06 de Abril de 2014, de http://www.puntopeek.com/codigos-c/el-metodo-de-ordenacion-merge-sort/