Aplicación de técnicas de programación paralela para regularizar los ensayos de los sondajes

Ing. Milenis Fernández-Díaz [1*], Ing. José Gabriel Espinosa-Ramirez [2], Ing. Irina Ivis Santiesteban-Pérez [3]

[1] Centro de Geoinformática y Señales Digitales (GEYSED). Universidad de las Ciencias Informáticas. Carretera a San Antonio de los Baños km 2 ½, Reparto Torrens, La Lisa, La Habana, Cuba. CP: 19370. mfdiaz@uci.cu

[2] Centro de Ideoinformática (CIDI). Universidad de las Ciencias Informáticas. Carretera a San Antonio de los Baños km 2 ½, Reparto Torrens, La Lisa, La Habana, Cuba. CP: 19370. jgespinosa@uci.cu

[3] Centro de Tecnologías para la Formación (FORTES). Universidad de las Ciencias Informáticas. Carretera a San Antonio de los Baños km 2 ½, Reparto Torrens, La Lisa, La Habana, Cuba. CP: 19370. iisantiesteban@uci.cu

[*] Autor para correspondencia: mfdiaz@uci.cu

Resumen

La estimación de los recursos minerales es un factor clave a la hora de decidir la ejecución de un proyecto minero. Los cálculos que se realizan deben asegurar un alto grado de certeza, pues las inversiones requeridas son muy costosas. Para estimar los recursos minerales se realizan sondajes de los que se extraen ensayos con el objetivo de analizar las concentraciones de los minerales. Para aplicar métodos geoestadísticos de estimación se requiere que la información obtenida a partir del estudio de estos ensayos se regularice en intervalos de longitudes previamente definidas, a este proceso se le conoce como compositación o regularización de los ensayos. La compositación se caracteriza por el procesamiento de grandes volúmenes de datos, lo que demora la obtención de los resultados. Con el objetivo de disminuir los tiempos asociados, se proponen dos algoritmos paralelos de regularización por igual longitud basados en el uso de técnicas de memoria compartida proporcionadas por OpenMP y QThreadPool.

Palabras clave: compósitos; ensayos; OpenMP; QThreadPool; regularización de ensayos

Introducción

Los recursos minerales son la base de la sociedad moderna, y su demanda ha aumentado aceleradamente con el desarrollo tecnológico. Los recursos minerales son finitos y, en su mayoría, no renovables, haciendo que la estimación de sus depósitos sea un elemento primordial, tanto para evaluar las reservas como para optimizar el rendimiento económico para la explotación de un yacimiento determinado. La estimación de los recursos minerales es un factor clave a la hora de decidir la ejecución de un proyecto minero, con el fin de asegurar el éxito. A partir de esta tarea se puede evaluar la factibilidad de los proyectos mineros, así como garantizar planificaciones adecuadas para estos. Los cálculos que se realizan deben asegurar un alto grado de certeza, pues las inversiones requeridas son muy costosas.

La estimación de reservas es la operación que determina el valor comercial o industrial de un yacimiento. El cálculo de los recursos y reservas minerales se puede realizar utilizando diversas técnicas que son agrupadas en métodos clásicos o métodos modernos. En los métodos clásicos se “utilizan fundamentalmente valores medios o medias ponderadas para la estimación de bloques defi-

nidos convenientemente" (CUADOR-GIL, 2002). En los métodos modernos predominan los métodos geoestadísticos, teniendo como premisa fundamental la realización de estimaciones a partir de las características de variabilidad y de correlación espacial de los datos originales.

La Geoestadística es resultado de la aplicación de la Teoría de Funciones Aleatorias al reconocimiento y estimación de fenómenos naturales, fundamentalmente los relacionados con la actividad geólogo minera (CUADOR-GIL, 2002). Para la aplicación de métodos geoestadísticos es imprescindible la automatización de estos. Siguiendo esta línea en la Universidad de las Ciencias Informáticas, específicamente en el Proyecto Sistema Minero Cubano se desarrolla una aplicación informática que incluye entre sus funcionalidades la estimación de recursos minerales.

La estimación de los recursos se realiza a partir de la información generada en las campañas de exploración, en las cuales se realiza el muestreo y documentación de cada pozo de perforación atendiendo a una serie de parámetros. Como resultado de la exploración de los yacimientos minerales se conforma una base de datos geológica que contiene los datos referentes a los pozos perforados, así como de los ensayos extraídos de los pozos, las concentraciones de minerales y las características litológicas del suelo. Entiéndase pozos, como excavaciones que se realizan en un terreno utilizando métodos mecánicos, con el objetivo de obtener muestras del subsuelo a profundidades variables.

Las muestras o ensayos constituyen porciones del subsuelo extraídas de un yacimiento mineral con el propósito de estudiar detalladamente la concentración de minerales. Los ensayos poseen tamaños irregulares que atentan contra la utilización de métodos geoestadísticos (Ver Figura 1). Es importante aclarar que la Geoestadística solo tiene en cuenta el valor de los datos y no las dimensiones de estos; lo cual conlleva a la necesidad regularizar los datos. Si bien el análisis geoestadístico exige muestras de igual longitud, la regularización de los tamaños de los ensayos permite reducir la cantidad de datos y por tanto el tiempo de procesamiento de estos también se reduce. Además, se proporcionan datos homogéneos que son más fáciles de interpretar (ESTÉVEZ-CRUZ, 2009).

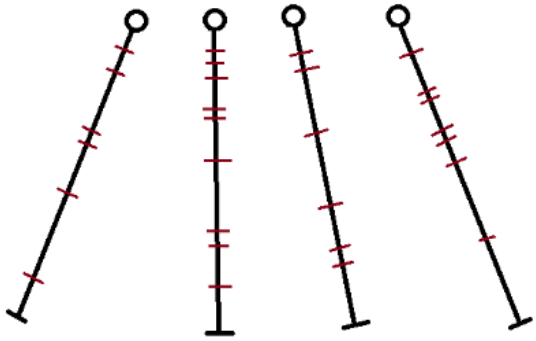

Figura 1. Pozos de perforación y ensayos obtenidos a partir de estos.

El proceso de regularización de los ensayos suele ser lento debido a que generalmente se trabaja con grandes volúmenes de información. La rapidez y la precisión son cualidades deseables en este tipo de operaciones. Las técnicas de programación en paralelo se han convertido en el único método eficaz para la solución rápida de problemas computacionalmente grandes y con muchos datos (GRAMA, y otros, 2003). Con el presente trabajo se pretende optimizar el tiempo de la regularización de los ensayos de los pozos de perforación, a partir de técnicas de programación en paralelo aprovechando los recursos computacionales.

Materiales y métodos

Regularización de los ensayos de los pozos de perforación

La regularización de los ensayos es el proceso de combinar muestras individuales de longitudes cortas en longitudes mayores. Se realiza el agrupamiento de los datos provenientes del muestreo realizado a los pozos de perforación, definiéndose un tamaño estándar para los intervalos de los ensayos. En la bibliografía referente a los principales sistemas mineros, a este proceso se le conoce como compositar (GEMCOM SOFTWARE INTERNACIONAL INC., 1998).

Los métodos de regularización son variados, se puede compositar a partir de datos existentes o a partir de las intersecciones de los pozos con superficies o sólidos. La longitud de los compósitos se puede determinar usando diversas técnicas, tales como longitudes fijas, intervalos de otra tabla, intersecciones con planos, valores de corte, entre otras (GEMCOM SOFTWARE INTERNACIONAL INC., 1998). En el presente trabajo se proponen algoritmos para regularizar los datos a partir de una longitud fija. Se asume que los intervalos de los ensayos se encuentran validados, de modo que no existan intervalos repetidos, solapados, no consecutivos o negativos, que puedan afectar el cálculo de los compósitos.

El método de compósitos por igual longitud consiste en definir una longitud constante para regularizar los ensayos. Este proceso se puede realizar de dos formas: comenzado por la boca del pozo (Figura 2), o en sentido contrario comenzando por el fondo del pozo (Figura 3). Evidentemente la longitud que se defina para los compósitos no debe exceder la longitud del pozo de perforación. Es posible que la longitud total del pozo no sea un múltiplo exacto de la longitud de los compósitos, originándose compósitos de menor tamaño al principio o al final del pozo, según la dirección en la que se haya realizado el proceso de regularización. Estos fragmentos pueden ser tenidos en cuenta o pueden ser desechados (GEMCOM SOFTWARE INTERNACIONAL INC., 1998).

Una vez regularizados los tamaños de los ensayos se deben calcular los valores de concentración de minerales correspondientes a cada uno de los intervalos obtenidos. Estos valores se determinan mediante el promedio entre la concentración de cada ensayo que compone los nuevos intervalos, quedando la fórmula de la siguiente manera:

$$compósito = \frac{\sum V_i * L_i * W_i}{\sum L_i * W_i}$$

Donde: $\mathbf{V_i}$ Valor de concentración del mineral, $\mathbf{L_i}$ Longitud del intervalo fuente en el compósito, $\mathbf{W_i}$ Valor de peso (por defecto 1).

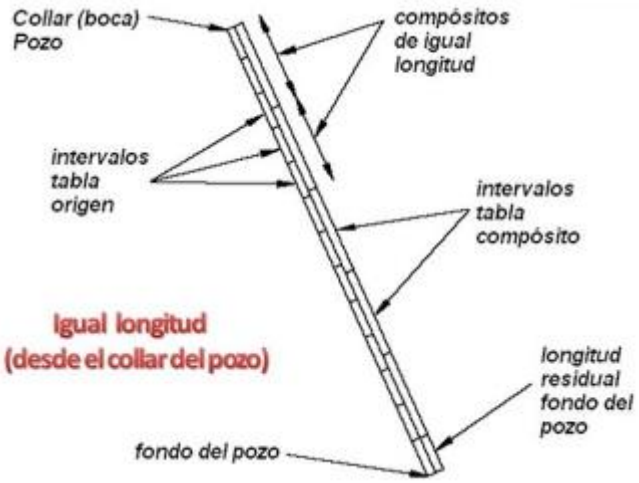

Figura 2. Regularización de ensayos por el método por igual longitud comenzando por la boca de pozo (GEMCOM SOFTWARE INTERNACIONAL INC., 1998)

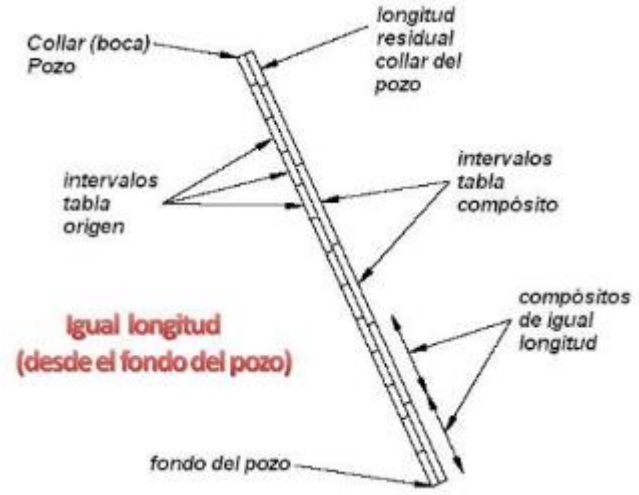

Figura 3. Regularización de ensayos por el método por igual longitud comenzando por el fondo del pozo (GEMCOM SOFTWARE INTERNACIONAL INC., 1998).

Técnicas de programación en paralelo utilizadas

La tabla que contiene los datos sobre la composición mineral de las muestras está constituida usualmente por un número alto de tuplas. Esto implica que la realización linear de la regularización sea lenta, no aprovechando en su totalidad las capacidades de cómputo instaladas. En busca de un mejor rendimiento es que se plantea utilizar técnicas que permitan paralelizar el proceso de regularización obteniendo menores tiempos de respuesta. En este documento se analizarán dos propuestas lineales de un algoritmo para la elaboración de los compósitos y sendos pares de variantes paralelizadas utilizando OpenMP (OPENMP, 2014) y QThreadPool (QT PROJECT, 2013) respectivamente.

OpenMP

OpenMP es una API (*Application Program Interface*) destinada a máquinas de memoria compartida, para programar explícitamente en aplicaciones paralelas multihilos. OpenMP se especifica para C/C++ y Fortran. Esta API está compuesta por un conjunto de directivas de compilador que el usuario utiliza para especificar qué regiones de código van a ser paralelizadas, y una biblioteca de

rutinas. El usuario también cuenta con un limitado número de variables de entorno que permiten especificar, entre otros aspectos, las condiciones que definirán la ejecución de la aplicación paralela (DORTA-LORENZO, 2008).
La implementación de OpenMP utiliza multiplicidad de hilos para obtener el paralelismo, mediante un modelo *fork-join*. Este modelo plantea la división del hilo maestro en hilos esclavos que se ejecutan concurrentemente, distribuyéndose las tareas sobre estos hilos, dependiendo de los parámetros indicados, la carga y otros factores. Estos hilos acceden a la misma memoria, aunque es posible gestionar estos accesos generando espacios de memoria privada (DORTA-LORENZO, 2008).

QThreadPool

En la programación paralela el número de hilos concurrentes está determinado por la cantidad de núcleos disponibles. Esto conlleva a dividir el problema de diferentes maneras de acuerdo con las características del hardware donde se vaya a ejecutar el programa en cuestión. Esta tarea no es sencilla cuando se desconocen las características de los ordenadores en los cuales serán ejecutados los programas.
El agrupamiento de hilos ofrece como principal ventaja la adaptación del paralelismo a las características del hardware subyacente, por ende se logra un mejor aprovechamiento de los sistemas multinúcleos. La clase QThreadPool (QT PROJECT, 2013) del marco de trabajo Qt proporciona una implementación de este patrón. El agrupamiento de hilos se basa en (TORRES, 2013):

1. Crear un grupo de hilos, una cola con las tareas que deben ser ejecutadas en dichos hilos y otra cola donde almacenar los resultados de las tareas ejecutadas.
2. Cada vez que un hilo del agrupamiento queda libre, toma la siguiente tarea de la cola y la ejecuta hasta que es completada, insertando los resultados en la cola correspondiente.
3. Cuando la cola de tareas está vacía, los hilos pueden morir o dormir hasta que hayan más tareas.

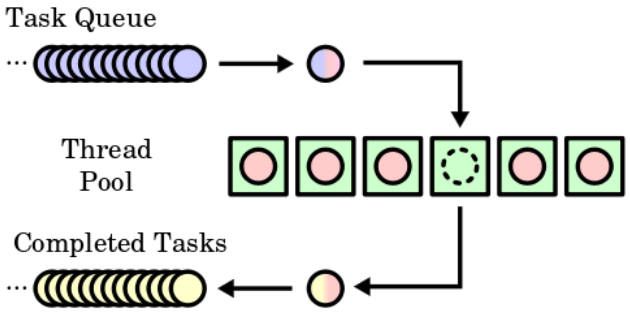

Figura 4. Patrón agrupamiento de hilos (TORRES, 2013).

La clase QThreadPool se ocupa de administrar y reciclar colecciones de hilos individuales contribuyendo a reducir los costos de creación de hilos. La programación multihilo a través de esta clase es fácil de entender debido a la abstracción que proporciona en cuanto a la gestión de hilos, alejando las preocupaciones en este sentido pues el programador solo se preocupará por definir las tareas y enviarlas a la instancia de QThreadPool, la cual decidirá dinámicamente la cantidad adecuada de hilos, invocará la ejecución de las tareas de forma paralela y notificará al usuario cuando el procesamiento haya finalizado. Una vez concluida

la ejecución de las tareas, estas son eliminadas automáticamente a menos que se especifique lo contrario (QT PROJECT, 2013).
Las aplicaciones Qt tienen una instancia global de QThreadPool que puede ser accedida fácilmente. Para utilizar un hilo del QThreadPool simplemente se define una clase que herede de *QRunnable* y se reimplementa la función *run*, luego se crea una instancia de la clase y se le pasa al QThreadPool a través de la función *start* (QT PROJECT, 2013). Es imposible pasar argumentos al método *run* por lo que se recomienda pasar los parámetros necesarios en el constructor de los objetos.

Implementación y ejecución

Para la implementación de los algoritmos, lineal y en paralelo, se utilizó el lenguaje de programación orientado a objetos C++ y como marco de trabajo Qt (QT PROJECT, 2014). Como entorno de desarrollo el Qt Creator (Qt Project, 2014), y como sistema de gestión de base de datos el PostgreSQL 9.1 (THE POSTGRESQL GLOBAL DEVELOPMENT GROUP, 2014). Para el proceso de compilación se empleó el compilador de Visual C++ que integra el API de OpenMP. Las versiones del algoritmo se ejecutaron en una computadora personal con 8 GB de RAM procesador Intel Core i7-3517U (INTEL, 2014) que cuenta con 2 núcleos y 4 subprocesos, con el sistema operativo Microsoft Windows 7 Home Edition.

Resultados y discusión

Algoritmo secuencial utilizado para la regularización de ensayos

Dado el tamaño que debe tener cada compósito y una tabla estructurada en columnas para los campos identificador del ensayo, identificador del pozo, inicio de la muestra, fin de la muestra, y *n* columnas, cada una de las cuales se corresponde con la ley presente para un mineral dado en la muestra. Se procede de la siguiente manera:

Para cada pozo realizar los siguientes pasos:

- Determinar la longitud del pozo.
- Determinar el conjunto de intervalos de los compósitos del pozo.
- Para cada intervalo de compósito del pozo:
    - Encontrar aquellos ensayos que caigan completamente o parcialmente en el intervalo correspondiente al compósito.
    - Calcular el valor que le corresponde al compósito en cada uno de los minerales analizados.

Al finalizar la ejecución del algoritmo se obtienen una lista de compósitos regularizados con la ley correspondiente a cada mineral analizado.

Algoritmos de Regularización

Para paralelizar el algoritmo se utilizaron técnicas de programación paralela en memoria compartida haciendo uso de hilos mediante la biblioteca OpenMP (OPENMP, 2014) y la clase QThreadPool (QT PROJECT, 2014). Paralelizar el algoritmo de regularización implica que en determinado momento se produzcan condiciones de competencia. Las condiciones de competencia son aquellas situaciones en las que dos o más procesos leen o escriben datos compartidos y el resultado depende de quién se ejecuta precisamente cuándo (TANEMBAUM, y otros, 1997).
Para evitar las condiciones de competencia es necesario controlar el acceso a la región crítica del código, aquella en la que se accede a la memoria compartida

(Tanembaum, y otros, 1997), mediante los semáforos implementados en la clase *QSemaphore* (QT PROJECT, 2014) del marco de trabajo de Qt, o la directiva "*critical*" de OpenMP. La clase QSemaphore basa en su funcionamiento en los semáforos, tipo de variables propuesta por Dijkstra en 1965 que realiza dos operaciones atómicas *UP* y *Down* sobre un contador para despertar o bloquear a los procesos que necesitan de un recurso sobre el cual se originan situaciones de competencia (TANEMBAUM, y otros, 1997).

Para la implementación del algoritmo se realizaron dos versiones secuenciales que sirven como base para realizar el estudio. Sobre cada una de estas se elaboraron dos versiones utilizando OpenMP y QThreadPool. Las implementaciones son similares distinguiéndose en la forma de realizar el cálculo del valor, el uso de variables referenciadas y de las secciones críticas. La variante 1 tiene tantos accesos a la sección crítica como la cantidad de ensayos, mientras que en la variante 2 este valor se reduce solamente a la cantidad de pozos.

Evaluación de los algoritmos paralelos

Para la evaluación de los algoritmos es importante contar con un conjunto de métricas de tasación de prestaciones, que consideren el tamaño de la entrada al algoritmo *n* y el número de procesadores *p*. En este caso los algoritmos fueron evaluados atendiendo a la Ganancia de Velocidad (*Speed Up*) que se ha obtenido con la ejecución en paralelo respecto al algoritmo secuencial. La ganancia de velocidad para *p* procesadores se determina mediante la fórmula: $S_P = \frac{T_S}{T_P}$ donde T_S es el tiempo de ejecución de un programa secuencial y T_P es el tiempo de ejecución de la versión paralela de dicho programa en *p* procesadores.

La Eficiencia (*E*) es otra métrica que también se analizó para evaluar el rendimiento de los algoritmos. La eficiencia significa el grado de aprovechamiento de los procesadores para la resolución del problema. El valor máximo que puede alcanzar es 1, que significa un 100% de aprovechamiento. La eficiencia se determina mediante la fórmula: $E = \frac{S_P}{p}$

A continuación se exponen los resultados obtenidos a partir de le ejecución de dos variantes secuenciales del algoritmo de regularización de ensayos y de sendos pares implementaciones, cada una de estas basada en OpenMP y QThreadPool respectivamente. Para observar el comportamiento de las implementaciones se utilizaron dos juegos de datos y dos asignaciones distintas de hilos, obteniéndose cuatro mediciones para cada versión. El conjunto de las versiones paralelas derivadas de cada implementación secuencial se identifica con el mismo número que la versión original.

Tabla 1. Comparación de los tiempos de ejecución, ganancia de velocidad y eficiencia de las implementaciones del algoritmo, para un juego de datos de 94 pozos y 4812 ensayos con 4 hilos.

Juego de datos de 94 pozos, 4812 ensayos						
Cantidad Hilos	4 Hilos					
Versión	Secuencial 1	OpenMP 1	QThreadPool 1	Secuencial 2	OpenMP 2	QThreadPool 2
T (ms)	846	524	377	322	370	290
S_P		1,61450382	2,24403183		0,87027027	1,11034483
E		0,40362595	0,56100796		0,21756757	0,27758621

Tabla 2. Comparación de los tiempos de ejecución, ganancia de velocidad y eficiencia de las implementaciones del algoritmo, para un juego de datos de 94 pozos y 4812 ensayos con 2 hilos.

Juego de datos de 94 pozos, 4812 ensayos						
Cantidad Hilos	2 Hilos					
Versión	Secuencial 1	OpenMP 1	QThreadPool 1	Secuencial 2	OpenMP 2	QThreadPool 2
T (ms)	846	452	557	322	363	374
S_P		1,87168142	1,51885099		0,88705234	0,86096257
E		0,93584071	0,75942549		0,44352617	0,43048128

Tabla 3. Comparación de los tiempos de ejecución, ganancia de velocidad y eficiencia de las implementaciones del algoritmo, para un juego de datos de 150 pozos y 15000 ensayos con 4 hilos.

Juego de datos de 150 pozos, 15 000 ensayos						
Cantidad Hilos	4 Hilos					
Versión	Secuencial 1	OpenMP 1	QThreadPool 1	Secuencial 2	OpenMP 2	QThreadPool 2
T (ms)	924	489	459	346	311	389
S_P		1,88957055	2,0130719		1,11254019	0,88946015
E		0,47239264	0,50326797		0,27813505	0,22236504

Tabla 4. Comparación de los tiempos de ejecución, ganancia de velocidad y eficiencia de las implementaciones del algoritmo, para un juego de datos de 150 pozos y 15000 ensayos con 2 hilos.

Juego de datos de 150 pozos, 15 000 ensayos						
Cantidad Hilos	2 Hilos					
Versión	Secuencial 1	OpenMP 1	QThreadPool 1	Secuencial 2	OpenMP 2	QThreadPool 2
T (ms)	924	519	500	346	326	447
S_P		1,78034682	1,848		1,06134969	0,77404922
E		0,89017341	0,924		0,53067485	0,38702461

Conclusiones

- La aplicación de técnicas de programación paralela en memoria compartida ofrece mayor rapidez en la regularización de los ensayos, al reducir el tiempo de cómputo de las operaciones requeridas.
- La gestión de los hilos y los mecanismos de acceso a las secciones críticas del código añaden un costo adicional a los algoritmos.
- QThreadPool manifiesta un mejor rendimiento en la medida en que se incrementa el número máximo de hilos disponibles.

- El incremento de los accesos a las secciones críticas desfavorece los rendimientos obtenidos utilizando la técnica OpenMP respecto al QThreadPool.
- Las estructuras de datos para la gestión o combinación de los resultados tienen un alto impacto en la cantidad de accesos a las secciones críticas y por tanto repercuten en la eficiencia de los algoritmos implementados.

Referencias

CUADOR-GIL, J. Q. “Estudios de estimación y simulación gesotadística para la caracterización de parámetros geólogo industriales en el yacimiento laterítico Punta Gorda”. Resumen de tesis doctoral. Universidad de Pinar del Río, Pinar del Río, 2002. ISSN 02585979

DORTA-LORENZO, A. J. “Extensión del modelo de OpenMP a memoria distribuida”. Memoria de tesis doctoral. Universidad de La Laguna, San Cristobal de La Laguna, 2008

ESTÉVEZ-CRUZ, E. “Apuntes sobre estimación de recursos y reservas”. Universidad de Pinar del Río, 2009

GEMCOM SOFTWARE INTERNACIONAL Inc. “Gemcom for Windows User Manual. Exploration”. Vancouver, 1998

GRAMA, A.; GUPTA, A. Y OTROS. “Introduction to Parallel Computing”. England, Addison Wesley, 2003. ISBN 0-201-64865-2

INTEL. ARK. “Intel”. [En línea] 2014 [8 de Febrero de 2014]. Disponible en: http://ark.intel.com/es/products/65714/Intel-Core-i7-3517U-Processor-4M-Cache-up-to-3_00-GH

OPENMP. “OpenMP”. [En línea] 2014 [Consultado el: 7 de Febrero de 2014]. Disponible en: http://openmp.org/wp/

THE POSTGRESQL GLOBAL DEVELOPMENT GROUP. Sitio oficial de PostgreSQL [En línea] 2014, [Consultado el: 7 de Febrero de 2014] Disponible en: http://www.postgresql.org

QT PROJECT. “QSemaphore Class Reference”. Qt Project. [En línea] 2014 [Consultado el: 7 de Febrero de 2014] Disponible en: http://qt-project.org/doc/qt-4.8/qsemaphore.html

QT PROJECT. “Qt Creator 2.8”. [En línea] 2014, [Consultado el: 7 de Febrero de 2014] Disponible en: http://qt-project.org/doc/qtcreator-2.8/

QT PROJECT. “Qt Project”. Qt Project. [En línea] 2014, [Consultado en: 17 de Febrero de 2014] Disponible en: http://qt-project.org

QT PROJECT. “QThreadPool Class Reference”. Qt Project. [En línea] 2013. [Consultado el: 7 de Febrero de 2014.] http://qt-project.org/doc/qt-4.8/qthreadpool.html

QT PROJECT. “Threading Basics”. Qt Project. [En línea] 2013. [Consultado el: 7 de febrero de 2014] Diponible en: http://qt-project.org/doc/qt-5.0/qtcore/thread-basics.html

TANEMBAUM, A. S.; WOODHULL, A. S. “Sistemas Operativos Modernos”. Prentice Hall, 1997. ISBN 970-17-0165-8

TORRES, J. “Concurrencia. Sistema de Video Vigilancia from Scratch”. [En línea] 2013 [Consultado el: 7 de febrero de 2014] Diponible en: http://ull-etsii-sistemas-operativos.github.io/videovigilancia-blog/concurrencia.html

www.sociedadelainformacion.com

Edita:

Director: José Ángel Ruiz Felipe
Jefe de publicaciones: Antero Soria Luján
D.L.: AB 293-2001
ISSN: 1578-326x