LMRE: Un entorno multiprocesador para la enseñanza de conceptos de concurrencia en un curso CS1

De Giusti Armando, Frati Emmanuel, Leibovich Fabiana, Sanchez Mariano, De Giusti Laura, Madoz María C.

Instituto de Investigación en Informática LIDI (III-LIDI) – Facultad de Informática –UNLP. Argentina
{degiusti, fefrati, fleibovich, msanchez, ldgiusti, cmadoz}@lidi.info.unlp.edu.ar

Abstract. Se presenta un entorno visual interactivo para la enseñanza de conceptos de concurrencia y paralelismo en un curso inicial de algoritmos. El entorno LMRE (Lidi MultiRobot Environment) es una evolución del Visual Da Vinci utilizado extensamente en la introducción a la programación en varias Universidades.
El artículo analiza la problemática del cambio tecnológico a partir de la introducción de los procesadores de múltiples núcleos y su impacto sobre la programación y describe una definición del entorno, así como las primitivas a utilizar en la programación de aplicaciones concurrentes.
Por último se detallan aspectos de implementación del prototipo actualmente en prueba, así como la evolución del mismo para ser empleado en cursos más avanzados de concurrencia.

Keywords: Algoritmos, Concurrencia, Paralelismo, Entorno, Multirobot, Algoritmos Concurrentes y Paralelos.

1 Introducción

Desde el nacimiento de las computadoras en la década del 50, el aprendizaje inicial de la programación ha estado asociado con el "modelo de Von Neumann" [1] cuya simplicidad y consistencia en el manejo de control y memoria le permitió sostenerse a lo largo de más de 5 décadas.

Sin embargo, resulta claro que la evolución de la tecnología de hardware (y también de software) fueron convirtiendo el "modelo Von Neumann" en una abstracción útil para analizar "threads" de programación secuencial, pero cada vez más difícil de sostener como paradigma real, ante computadoras con múltiples procesadores que trabajan en forma paralela (incluso en una PC de escritorio). [2] [3] [20].

La aparición de los procesadores de múltiples núcleos ("multicores") [4] ha significado el agotamiento definitivo del modelo Von Neumann, ya que los procesadores (aún en sus expresiones más simples) presentan una arquitectura esencial MIMD sin reloj único y con capacidad de procesamiento paralelo y múltiples memorias que pueden ser total o parcialmente compartidas. [5] [6] [7].

Tal como se ve en la Fig. 1 una arquitectura típica de un procesador multicore presenta varios elementos a considerar:

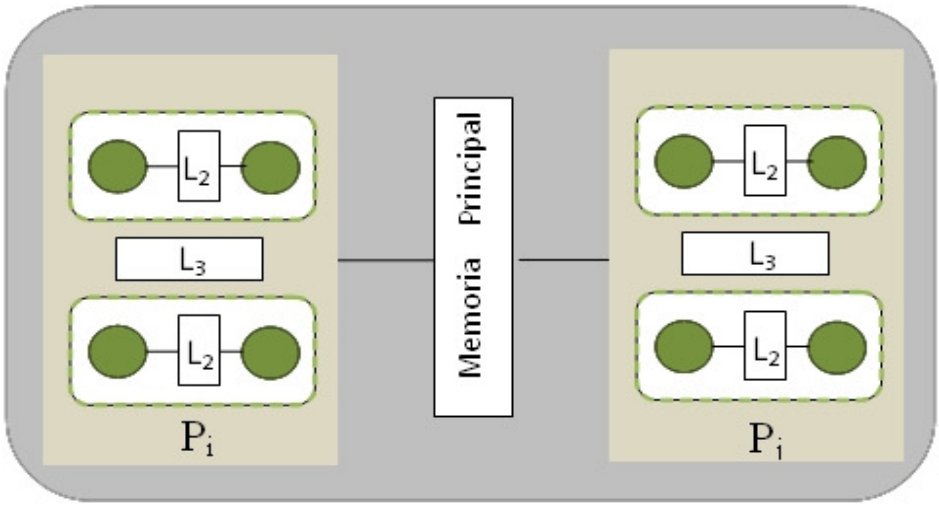

Fig. 1. Diagrama de estructura de un multicore

- Cada núcleo tiene su propio reloj, memoria local y unidad aritmético lógica.
- Existen memorias compartidas globales y locales (por ej. entre dos núcleos), con lo cual el mapa global de memoria es de acceso no uniforme (NUMA).
- Una aplicación a ejecutarse sobre este tipo de procesador, puede resolverse como múltiples hilos de control ejecutándose concurrentemente sobre los diferentes núcleos.
- Para resolver la aplicación se requiere resolver dos problemas básicos: la descomposición y asignación de las tareas concurrentes a los núcleos y la coordinación (comunicación y sincronización) entre ellos [8].

Este breve análisis nos muestra que debemos considerar la "máquina paralela" subyacente para resolver la aplicación, lo cual impacta sobre todos los niveles de software, especialmente los lenguajes, compiladores y sistema operativo. [9] [21] [22].

Por otra parte las métricas clásicas de eficiencia de los algoritmos deberán considerar el número de núcleos utilizados, su rendimiento y la bondad de la descomposición concurrente que se ha realizado de la aplicación para reducir los tiempos ociosos. [10]

En este punto, debemos considerar que los problemas del mundo real son *esencialmente paralelos*, con lo cual las nuevas arquitecturas si bien son más complejas, también nos permiten soluciones que se adaptan más eficientemente al modelo de las aplicaciones.

Asimismo resulta claro que es conveniente introducir conceptos de concurrencia y paralelismo desde los primeros cursos de algorítmica, de modo de formar a nuestros estudiantes en los aspectos esenciales del software de base y de aplicación que se desarrolla en las nuevas arquitecturas de procesadores, intentando explotar todas sus potencialidades. [11] [12].

En este trabajo se presenta un entorno visual interactivo, para su empleo en un curso clásico CS1, en el que se plantean de modo sencillo los conceptos fundamentales de la programación concurrente y se favorece el desarrollo de algoritmos concurrentes/paralelos por el alumno. Las secciones 2 y 3 están dedicadas a describir conceptualmente el entorno LMRE, en la sección 4 se explican las primitivas de programación que el alumno puede emplear, la sección 5 resume algunos aspectos de implementación y en las secciones 6 y 7 se analizan extensiones para cursos superiores al CS1 así como conclusiones y líneas de investigación y desarrollo actuales.

2 Objetivos del entorno a desarrollar

Partiendo del análisis expuesto en la Introducción y de la experiencia en el desarrollo y utilización de Visual Da Vinci [13] [14] [15], se plantea una evolución del entorno inicial de programación de algoritmos por alumnos de un curso CS1, de modo de incorporar el concepto clave de *concurrencia*.

Esto significa que el entorno debe permitir que el alumno trabaje naturalmente con dos puntos básicos:

- La comunicación entre procesos concurrentes (que normalmente significa una forma de *cooperación* entre ellos).
- La sincronización entre procesos concurrentes (que normalmente significa una forma de *competencia o dependencia* entre ellos). Para sincronizar debemos poder expresar mecanismos de exclusión mutua, explícitos y/o implícitos.

Estudiar comunicación y sincronización entre algoritmos (“procesos”) residentes en los núcleos de una arquitectura como la de la Fig. 1 significa conceptualizar y desarrollar 3 modelos fundamentales para el alumno:

- Memoria compartida.
- Mensajes.
- Híbrido. (algoritmos que utilizan memoria compartida y también mensajes).

El entorno LMRE se plantea entonces facilitar la expresión de algoritmos concurrentes (para el alumno de CS1 serán totalmente paralelos, porque se asocia un proceso a un procesador, a fin de simplificar el aprendizaje) que se pueden comunicar y/o sincronizar mediante mensajes y memoria compartida. [16] [17] [18] [19].

3 Características del entorno LMRE

El entorno permite definir una "ciudad" que es una cuadricula de N calles x N avenidas.

También se puede definir un número de K robots que están habilitados para recorrer la ciudad. Cada robot tiene una identificación (ID) única. En principio se establece un número máximo de N robots.

En la ciudad pueden definirse Áreas Exclusivas (AE). En el nivel CS1, en un AE puede circular sólo un robot determinado.

También puede definirse un Área Compartida (AC), a la que tienen acceso todos los robots.

Las "esquinas" de la ciudad pueden tener objetos de dos tipos (en principio flores y caramelos). La distribución inicial de los objetos puede realizarse automáticamente por el sistema o manualmente por el programador.

Los robots tienen capacidad de almacenamiento de objetos, en principio sin una limitación máxima.

Los robots pueden realizar acciones elementales con los objetos (Depositar / Recoger / Contar) y podrán Informar resultados de sus actividades.

Nótese que si N=100, K=1 y toda la ciudad está disponible para el único robot, tenemos la implementación definida para Visual Da Vinci.

Dado que nuestro objetivo es el trabajo de múltiples robots en la ciudad, tendremos que agregar acciones de comunicación por mensajes y de acceso a recursos compartidos (esquinas de la ciudad, objetos, contadores).

Cuando tenemos K robots (K > 1) trabajando en un área compartida (que puede ser toda la ciudad), tendremos que resolver los problemas de exclusión mutua sobre memoria compartida.

Cuando tenemos robots independientes trabajando en áreas exclusivas de la ciudad, que deben colaborar en la resolución de algún problema, tendremos comunicación por mensajes.

Nótese que ambos mecanismos pueden coexistir, si en la ciudad definimos áreas exclusivas y un área compartida. En este caso nos aproximaremos mucho al modelo de arquitectura de la Fig. 1.

4 Primitivas de concurrencia para el Curso CS1

Las 4 primitivas esenciales que provee el entorno para la especificación de la concurrencia son:

Enviar mensaje (EM): permite que un robot envíe un mensaje a otro (identificados por su ID). Al enviar el mensaje, según el modelo sincrónico, el robot se queda en espera de recepción (sincronización) por el robot destino. El envío de mensajes a una ID especial, convierte los mismos en broadcast.

Recibir mensaje (RM): indica que un robot se quedará esperando hasta sincronizar con el envío de mensaje de otro. En la recepción se indica el número de robot del cual se espera el mensaje, o un ID especial que indica que puede ser de cualquiera.

Bloquear recurso (BR): indica que el robot pide exclusión de un recurso, para poder utilizarlo en forma exclusiva. Típicamente en un Área Compartida, la ocupación de una esquina (que permite recoger o depositar objetos) debe ser exclusiva para un robot.

Liberar recurso (LR): indica que el robot deja el recurso libre (por ejemplo la esquina ocupada). También es una primitiva de uso natural en un Área Compartida.

En una estructura como la de la Fig. 2, estas primitivas nos permiten plantear problemas de "procesos independientes que se comunican por mensajes", tal como sería el caso de los 4 robots de las áreas exclusivas realizando una tarea (por ejemplo recogiendo las flores que haya en las esquinas de su área de recorrido) y luego comunicando sus resultados parciales a un quinto robot que informará los resultados.

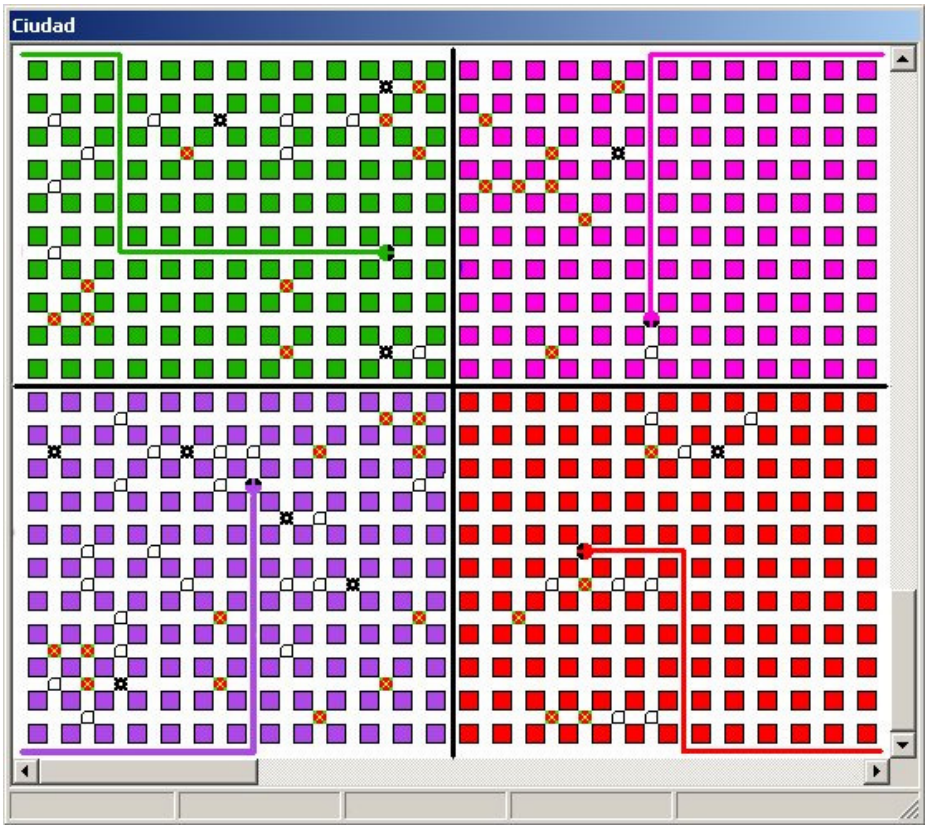

Fig. 2. La imagen muestra la ciudad dividida en 4 areas exclusivas, identificadas por los colores de los robots.

También podemos tener problemas de "recorridos concurrentes" en un área compartida como la de la Fig. 3 donde varios robots intentan cumplir una tarea (por ejemplo depositar caramelos en determinadas esquinas), partiendo de diferentes lugares de la ciudad, cuidando en cada caso intentar ocupar al mismo tiempo la misma esquina.

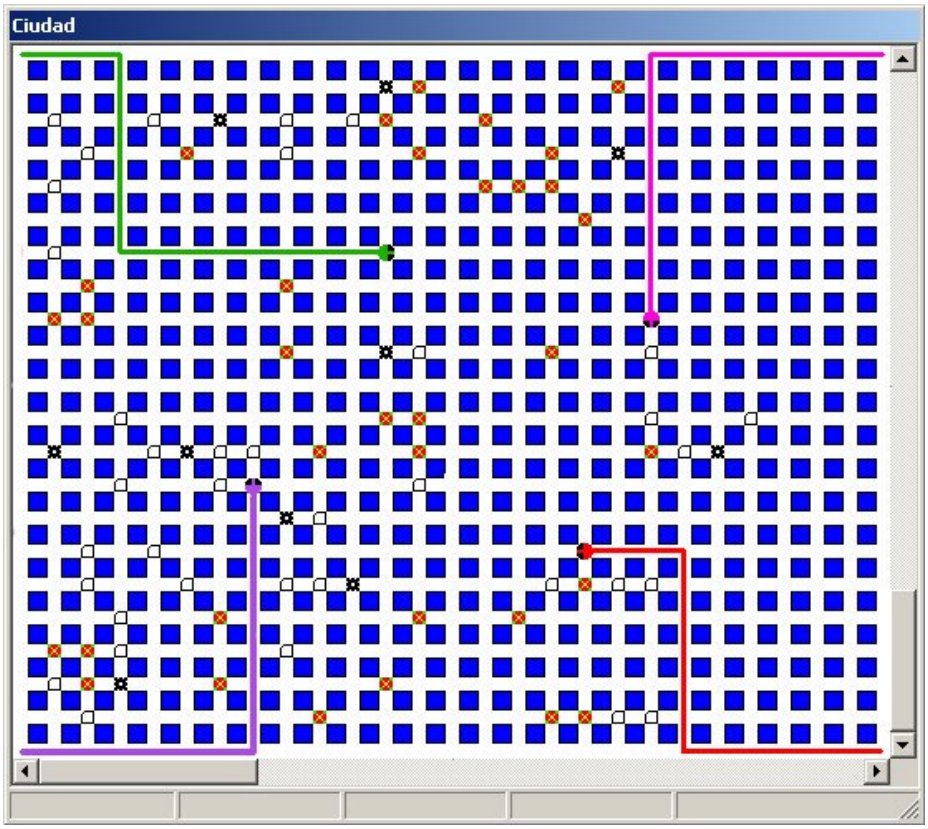

Fig. 3. En esta imagen se ve la ciudad compartida (en color azul) por los 4 robots, diferenciados por su color.

Por último en la Fig. 4 observamos un esquema de Áreas Exclusivas y Área Compartida, que nos permite combinar procesos independientes sobre las primeras y luego que todos los robots vayan al área compartida a completar el trabajo. Claramente esto nos permite visualizar los conceptos de programación híbrida.

Es de hacer notar que el entorno permite que el programador cree múltiples instancias del mismo código, para diferentes robots.

5 Implementación del entorno

Para la implementación del entorno se seleccionó el lenguaje de programación Java. Este lenguaje, además de ser multiplataforma, posee la clase Thread que permite la construcción de programas concurrentes (sin necesidad de ninguna otra herramienta adicional) y proporciona métodos de sincronización entre hilos.
Sin embargo, Java no tiene control sobre las posibles situaciones que pueden ocurrir en los programas concurrentes (bloqueos, violaciones de atomicidad, condiciones de carrera), pero provee mecanismos que permiten al desarrollador controlarlos algorítmicamente. Estos aspectos se deben tener en cuenta durante la implementación del entorno del multirobot, para evitar que estos aparezcan.

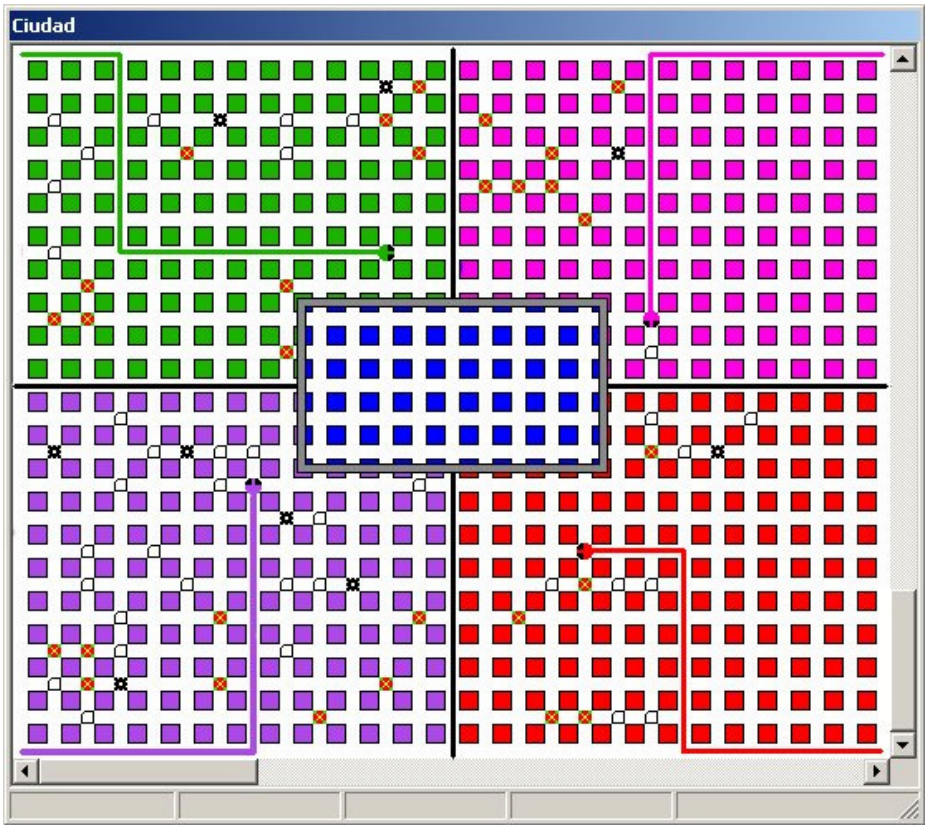

Fig. 4. En la imagen pueden observarse áreas privadas para cada robot (identificadas por su color) y un área compartida en color azul.

6 Extensiones para cursos superiores

Al utilizar el entorno LMRE en cursos más avanzados de concurrencia, se ha planificado agregar funcionalidades de interés académico:

- Los arreglos de bloqueos (semáforos) de modo de poder manejar barreras o esquemas FORK-JOIN.
- La exclusión mutua selectiva. En la ciudad esto conduce a manejar áreas compartidas entre determinados robots y áreas que excluyen robots selectivamente.
- El manejo de prioridades para el acceso a un recurso compartido. La prioridad puede ser un atributo estático del robot o dinámico en función de la situación de contexto y la tarea a realizar por los robots.
- La comunicación por mensajes asincrónicos y su reflejo en el estado de cada robot.

7 Conclusiones y Líneas de Trabajo

Se ha presentado el entorno visual interactivo LMRE para la enseñanza de conceptos de concurrencia y paralelismo en un curso inicial de algoritmos, a partir de la problemática del cambio tecnológico por la introducción de los procesadores de múltiples núcleos y su impacto sobre la programación.

Al describir las funcionalidades del entorno, se han discutido las primitivas a utilizar en la programación de aplicaciones concurrentes y su justificación.

Se presentaron aspectos de la implementación y las extensiones del entorno para cursos superiores al CS1.

Las dos líneas de trabajo más importantes son:

- Incorporar el tiempo como un atributo de las diferentes operaciones a realizar por los robots y permitir de este modo el estudio de los temas de speed-up, eficiencia y rendimiento de los algoritmos.
- Desarrollar una biblioteca de ejemplos clásicos de concurrencia/paralelismo sobre el entorno LMRE.

8 Bibliografía

1. A. W. Burks, H. H. Goldstine, and J. von Neumann, “Preliminary discussion of the logical design of an electronic computing instrument,” Papers of John von Neumann on Computing and Computer Theory, 1947.
2. M. D. Hill and M. R. Marty, “Amdahl’s law in the multicore era,” Computer, vol. 41, pp. 33–38, 2008.
3. J. Held, J. Bautista, and S. Koehl, “From a few cores to many: A tera-scale computing research overview,” tech. rep., Intel Corporation, 2006.
4. D. J. Ernst and D. E. Stevenson, “Concurrent CS: Preparing students for a multicore world,” in Proceedings of the 13th annual conference on Innovation and technology in computer science education (ITiCSE, ed.), ITiCSE, 2008.
5. V. Pankratius, C. Schaefer, A. Jannesari, and W. F. Tichy, “Software engineering for multicore systems: an experience report,” in IWMSE ’08: Proceedings of the 1st international workshop on Multicore software engineering, (New York, NY, USA), pp. 53–60, ACM, 2008.
6. T. Murphy, “High-performance computing in high schools?,” IEEE Distributed Systems Online, vol. 8, no. 8, pp. 1–3, 2007.

7. S.Lathrop and T. Murphy, “High-performance computing education,” Computing in Science and Engineering, vol. 10, no. 5, pp. 9–11, 2008. Introducción a la revista escrita por los editores.
8. De Giusti A., Frati Emmanuel ¿Concurrencia y Paralelismo en el primer curso de Algorítmica?. Proceedings TE&ET 2010. Pag. 276-285. Calafate- 2010.
9. A. Marowka, “Think parallel: Teaching parallel programming today,” IEEE Distributed Systems Online, vol. 9, pp. 1–8, 2008.
10. B. Rague, “Teaching parallel thinking to the next generation of programmers,” Journal of Education, Informatics and Cybernetics, vol. 1, no. 1, pp. 43–48, 2009.
11. A.-C. J. C. T. Force, “Computer science curriculum 2008: An interim revision of CS 2001,” Tech. Report, ACM Press, Dec. 2008.
12. ACM – AIS “IS 2010 Curriculum Guidelines for Undergraduate Degree Programs in Computer Science and Information Systems” ACM Press, 2010.
13. Champredonde R., De Giusti A. “Design and Implementation of the Visual Da Vinci Language.” Proceedings del V Ateneo de Profesores Universitarios de Computación, CACIC 97 – Pag. 980-992 – La Plata 1997
14. De Giusti A., Lanzarini L., Madoz MC “Abstract machines in a first course of computer science” Proceedings of 11th International Symposium “Computer at the University” – Zagreb – Yugoeslavia – Pag. 283-291 – 1990.
15. A. De Giusti, Algoritmos, datos y programas con aplicaciones en Pascal, Delphi y Visual Da Vinci. Pearson Education and Prentice Hall, 1 ed., 2001.
16. S. Carr, J. Mayo, and C.-k. Shene, “Threadmentor: a pedagogical tool for multithreaded programming,” ACM Journal of Educational Resources, vol. 3, pp. 1–30, march 2003.
17. H. Farian, K. M. Anne, and M. Haas, “Teaching high-performance computing in the undergraduate college cs curriculum,” Journal of Computing Sciences in Colleges, vol. 23, no. 3, pp. 135–142, 2008.
18. C. W. Kessler, “Teaching parallel programming early,” in Proceedings of Workshop on Developing Computer Science Education: How Can It Be Done?, p. 6, March 2006.
19. Leibovich, F., De Giusti, L., Naiouf, M. “Parallel Algorithms on Clusters of Multicores: Comparing Message Passing vs Hybrid Programming”. Julio 2011. WorldComp’11.
20. AMD, “Evolución de la tecnología de múltiple núcleo”. 2009. http://multicore. amd.com/es-ES/AMD-Multi-Core/resources/Technology-Evolution.
21. Mc Cool M., “Programming models for scalable multicore programming”. 2007. http://www.hpcwire.com/features/17902939.html
22. Chapman B., “The Multicore Programming Challenge, Advanced Parallel Processing Technologies”; 7th International Symposium, (7th APPT'07), Lecture Notes in Computer Science (LNCS), Vol. 4847, p. 3, Springer-Verlag (New York), November 2007.