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

Laura De Giusti[1], Fernando Emmanuel Frati[1,2], Fabiana Leibovich[1], Mariano Sanchez[1], María C. Madoz[1]

1 Instituto de Investigación en Informática LIDI (III-LIDI) – Facultad de Informática –UNLP, La Plata, Argentina
2 Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET) , La Plata, Argentina

Resumen

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.

Palabras clave: Algoritmos, Concurrencia, Paralelismo, Entorno, Multirobot, Algoritmos Concurrentes y Paralelos.

1. Introducción

Un curso clásico de Algorítmica está centrado en la expresión del control mediante un número reducido de primitivas y la introducción de estructuras de datos elementales.

Normalmente se analiza un único flujo de control, con una métrica elemental de eficiencia dada por el tiempo de ejecución de un dado algoritmo para un determinado conjunto de datos de entrada.

La verificación de los algoritmos resulta bastante directa y su validación es posible respetando algunos principios básicos de la programación estructurada [1]. Conceptos como los de modularización, documentación o reuso se incorporan como "buenas prácticas" de programación. El análisis de los problemas se concentra en encontrar una expresión algorítmica correcta y en lo posible eficiente, para alcanzar la solución.

En todo momento subyace una arquitectura de único thread de control, con reloj único.

Sin embargo, este enfoque deja de lado dos aspectos importantes:

1. La arquitectura de los procesadores actuales es paralela

Una arquitectura típica de un procesador multicore, tal como la que se ve en la Figura 1, presenta varios elementos a considerar:

- 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 un procesador como el de la Figura 1, puede resolverse como múltiples hilos de control ejecutándose concurrentemente sobre los diferentes núcleos.
- Claramente esto requiere (al menos) resolver dos problemas básicos: la descomposición y asignación de las tareas concurrentes a los núcleos y la coordinación (comunicación y sincronización) entre los núcleos para resolver la aplicación.

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 [2].

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 [3].

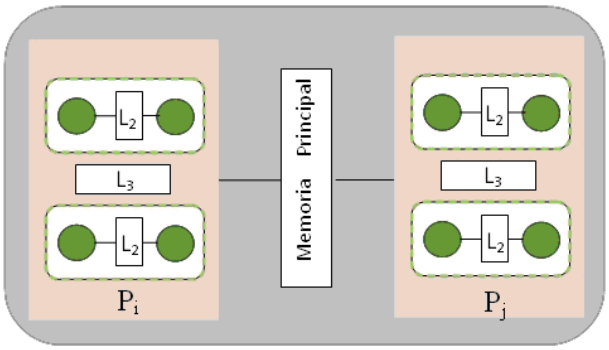

Figura 1. Diagrama de estructura de un multicore

2. Los problemas del mundo real son escencialmente concurrentes

La concurrencia es la propiedad que tienen algunos problemas de ser divididos (total o parcialmente) en tareas más pequeñas que pueden ser ejecutadas al mismo tiempo. Cuando esas tareas son ejecutadas simultáneamente por diferentes procesadores, se dice que el programa se ejecuta en paralelo.

Debemos considerar que los problemas del mundo real son esencialmente concurrentes: bases de datos, lenguajes de programación, sistemas operativos, servidores web, aplicaciones comerciales… todos pueden beneficiarse si se resuelven en forma paralela.

De esta manera, aunque las nuevas arquitecturas son más complejas, también nos permiten soluciones que se adapten más eficientemente al modelo de las aplicaciones.

Resulta claro que es necesario asumir múltiples desafíos en la enseñanza de la programación. En particular replantearnos un curso "clásico" de Algoritmos.

Ejecutar eficientemente una aplicación sobre una arquitectura como la de la Figura 1 conducirá casi siempre a la programación paralela. La programación secuencial será sólo un "caso degenerado" poco aplicable, ya que conduciría a tener N-1 núcleos del procesador ociosos.

En este sentido la última recomendación para la curricula de ciencias de la computación de la ACM/IEEE [4] identifica a la concurrencia como un tópico para atención y propone incorporar nociones de programación paralela a la estructura del primer curso de programación de la carrera sin alterar significativamente los contenidos (programación estructurada, luego programación orientada a objetos y luego programación paralela).

Existen diferentes propuestas destinadas a satisfacer esta recomendación. En [5] los autores reconocen que existe la necesidad de desarrollar herramientas que faciliten tratar con problemas de cómputo de altas prestaciones. Su propuesta radica en un software que facilita el desarrollo y ejecución de programas paralelos en clusters HPC, sobre hardware alquilado a Amazon y utilizan imágenes de máquinas virtuales para cada curso. Aunque el framework que desarrollaron es escalable y rentable, está enfocado a facilitar la gestión de los recursos necesarios para la enseñanza de la programación paralela, y no en facilitar al alumno la adquisición de sus principios.

Otros autores presentan una estrategia de clase basada en herramientas de visualización destinadas a la conceptualización de los programas paralelos [6]. En este caso la herramienta propuesta consiste en una aplicación que permite la introducción de código y la visualización del mismo a través de diagramas UML. Con ella se puede identificar visualmente candidatos a secciones de codigo concurrente (como por ejemplo la inicialización de vectores). Esta propuesta constituye una ayuda en el análisis del código paralelo, pero aún el alumno debe conocer el lenguaje con las primitivas específicas que soportan la concurrencia.

Recientemente en [7] los autores presentan un estudio de los últimos tres años de un curso CS1 en el que han introducido conceptos de concurrencia a través de casos prácticos y sencillos que deben ser implementados en java. En el artículo se remarca la importancia de transimitir a los alumnos los conceptos generales de concurrencia para despertar su interés en esos temas, como también la imposibilidad de abarcar temas avanzados como métricas de eficiencia en una etapa temprana de la carrera.

En este trabajo se presenta una alternativa a estos enfoques a través de 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. La diferencia principal de nuestro enfoque con los casos antes mencionados es que en lugar de forzar a los alumnos a aprender la sintaxis propia de las librerías de comunicaciones, proveemos a través del entorno de un conjunto de primitivas simplificadas con el objeto de facilitar la adquisición de los conceptos de concurrencia.

Las sección 2 está dedicada a describir el curso de CS1 al que apunta el nuevo entorno. En la sección 3 se presenta el entrono LMRE y se explican las primitivas de programación que se agregan al leguaje original DaVinci; la sección 4 muestra tres ejemplos de modelos de programación concurrente usando el

entorno. En la sección 5 se comentan algunos aspectos de implementación y en la sección 6 se presentan las conclusiones y líneas de investigación futuras.

2. Curso cs1 en la UNLP

El curso que se dicta actualmente en la Facultad de Informática de la UNLP abarca dos semestres y tiene el título genérico de "Algoritmos, Datos y Programas". Responde a un modelo clásico centrado en el aprendizaje de la expresión de algoritmos, la introducción a las estructuras de datos lineales y no lineales y la incorporación de conceptos relacionados con modularización, análisis de eficiencia, abstracción y reuso.

En el 2$^{do.}$ semestre se introduce conceptualmente el paradigma de objetos.

Las tareas experimentales se realizan inicialmente con un entorno propio Visual Da Vinci [8] y posteriormente con Pascal.

A partir del año 2010 se ha propuesto un cambio gradual, que tiene 3 etapas:

- Incorporación de la descripción de las arquitecturas de los nuevos procesadores, como evolución de la máquina de Von Neumann e Introducción a los conceptos básicos de Concurrencia.
- Desarrollo de un entorno visual de expresión de Algoritmos concurrentes y paralelos que resulte en una evolución del Visual Da Vinci (VDV) y admita mecanismos de comunicación y sincronización, tanto por memoria compartida como por mensajes.
- Rediseño del programa de la asignatura, planteando la programación concurrente/paralela como el caso general y la programación secuencial como la excepción.

3. Objetivos del entorno LMRE

Partiendo del análisis expuesto en la introducción y de la experiencia en el desarrollo y utilización de Visual Da Vinci [1] [8] [9], 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 Figura 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. [10] [11] [12] [13].

3.1. Características

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

También se puede definir un número de K robots que están habilitados para recorrer la ciudad. Cada robot se declara como una variable del programa que sirve para comunicarse con él. En principio se establece un número máximo de R robots.

En la ciudad pueden definirse áreas de circulación para los robots. Un área de la ciudad es un subconjunto rectangular de esquinas de la ciudad. Estas pueden ser de tres tipos:

- Área compartida (areaC): es el tipo de área por defecto y corresponde a una región de la ciudad de libre acceso, es decir, cualquier robot puede circular por ella.
- Área privada (areaP): una región de este tipo sólo permite que haya un robot en ella. El intento de un robot de ingresar en un área privada de otro, terminará en un error en tiempo de ejecución.
- Área parcialmente compartida (areaPC): este tipo de regiones permiten el acceso de uno o varios robots, con la restricción de que deben haber sido autorizados previamente.

Cada área está identificada visualmente en la ciudad por un marco que encierra el conjunto de esquinas que lo componen, indicando con un color el tipo de

área de que se trata. La Figura 2 muestra una ciudad dividida en estas tres áreas.

Las "esquinas" de la ciudad pueden tener objetos de dos tipos (en principio flores y papeles).

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.

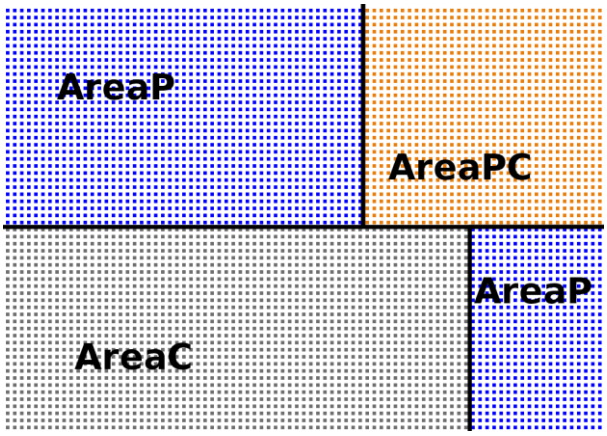

Figura 2. Configuración de áreas de la ciudad. La región azul corresponde a un área privada, la gris a un área compartida y la naranja a un área parcialmente compartida.

Los robots pueden realizar acciones elementales con los objetos (Depositar / Recoger / Contar) y podrán informar resultados de sus actividades. El listado 1 resume el conjunto de primitivas que permiten la interacción con el robot.

Mover	**PosAv**
Derecha	**PosCa**
depositarFlor	**HayFlorEnLaEsquina**
depositarPapel	**HayPapelEnLaEsquina**
tomarPapel	**HayFlorEnLaBolsa**
tomarFlor	**HayPapelEnLaBolsa**
Pos(av, ca)	**Informar(…)**

Listado 1: Sentencias asociadas a cada robot

Nótese que si N=M=100, K=1 y toda la ciudad está dentro de un área privada 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 Figura 1.

3.2. Creación de robots

Los robots tienen una estructura casi idéntica a la del programa principal o subprocesos de un programa Da Vinci. Cada declaración de robot tiene las tres partes de un programa: encabezado, declaraciones y cuerpo.

Cada declaración de robot comienza con la palabra clave *robot* seguido de un nombre. Las declaraciones de procesos locales y variables locales siguen las mismas reglas que las declaraciones del programa principal, con la salvedad que no pueden declararse nuevas áreas o robots.

De esta manera, la creación de robots siempre será explicita. Una característica a destacar es que aún cuando se quiera utilizar el entorno para simular un ambiente como el del Da Vinci original, será necesario crear el único robot desde el programa principal.

El cuerpo de un robot es una secuencia de sentencias, delimitada por las palabras clave comenzar y fin. Estas sentencias se corresponden con los tipos de sentencias de Da Vinci, y sirven para definir el comportamiento del robot en la ciudad, presentados en el listado 1. A estas sentencias se agrega un nuevo conjunto, denominado sentencias de concurrencia.

3.3. Sentencias de concurrencia

A continuación se detallarán las directivas necesarias para el manejo de concurrencia cuando el programa posee más de un robot.

Manejo de colisiones

En este entorno donde las esquinas de la ciudad representan el recurso compartido, se establece que la situación en que dos robots intentan situarse al mismo tiempo en la misma esquina provocará una *colisión*, lo que deriva en un error en tiempo de ejecución.

Las colisiones de LMRE están estrechamente relacionadas con una clase de error muy común en

programas concurrentes llamada *condición de carrera*. En éstas, dos o más procesos intentan acceder simultáneamente a la misma posición de memoria. En programas reales esta situación sólo deriva en error si al menos uno de los accesos se trata de una escritura. Sin embargo, el concepto clave que queremos transmitir es la importancia de sincronizar los accesos que los procesos hacen sobre el recurso compartido. Por este motivo consideramos que en LMRE el error ocurra independientemente del tipo de acceso que los robots hagan sobre la esquina.

Si un robot está circulando por un área privada, estas situaciones no pueden existir. No ocurre lo mismo con las regiones compartidas, donde las colisiones entre los robots que circulan por ella pueden ser muy frecuentes. Por este motivo, el lenguaje debe contar con directivas que permitan bloquear y liberar explícitamente el acceso de otros robots a una esquina. Para ello se definen las siguientes directivas:

- *Bloquear esquina (BE)*: indica que el robot pide exclusión para ocupar una esquina (que permite recoger o depositar objetos).
- *Liberar esquina (LE)*: indica que el robot deja el recurso libre (la esquina ocupada).

Comunicaciones

Independientemente del tipo de áreas por las que circulen los robots, es natural pensar que tendrán que comunicarse entre sí para resolver ciertos problemas, como ser el conteo de objetos en la ciudad.

Una posibilidad podría ser que todos los robots depositen en una esquina de un área compartida los objetos que recogieron y un robot especial tenga la función de recogerlos sólo de esa esquina. Aún así, se debe notar que es necesario "avisarle" a este robot cuando cada robot completó su tarea.

Una alternativa más natural sería que los robots en lugar de depositar los objetos en el área compartida, *comuniquen* sus cuentas parciales al coordinador para que este informe el total.

Esta característica está diseñada para simular el modelo de mensajes sincrónico típico de problemas distribuidos en programas concurrentes. Para ello, definiremos dos directivas nuevas:

- *Enviar mensaje (EM)*: permite que un robot envíe un mensaje a otro (identificados por su nombre). 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 un nombre especial, convierte los mismos en una comunicación colectiva tipo 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 nombre del robot del cual se espera el mensaje. Si se usa un nombre especial el mensaje puede recibirlo de cualquiera.

Se debe notar que el uso de estas primitivas implica que si un robot envía un mensaje, otro debe recibirlo. Si esto no ocurre, la ejecución del programa derivará en otra condición de error habitual en programas concurrentes denominada *deadlock*, donde ninguno de los procesos avanza esperando que ocurra una condición que depende del otro proceso. El listado 2 resume las primitivas antes mencionadas.

Existen otras condiciones que habitualmente tienen palabras clave especiales en lenguajes o librerías específicos para desarrollar programas paralelos (como ser MPI u openMP). Un ejemplo de esto son las barreras o las reducciones.

bloquearEsquina (av, ca)	Otorga acceso exclusivo a la esquina
liberarEsquina (av, ca)	Libera el acceso exclusivo a la esquina
enviarMensaje(robot, msj)	Establece una comunicación entre robots. El robot destino debe ejecutar una operación recibirMensaje para completar la sentencia.
recibirMensaje(robot, msj)	Establece una comunicación entre robots. El robot origen debe ejecutar una operación enviarMensaje para completar la sentencia.

Listado 2: Sentencias para soportar concurrencia

Una barrera es una situación donde cada proceso que participa en esta, debe esperar a que todos los demás lleguen a la barrera: sólo entonces pueden continuar con su ejecución.

Una reducción es una operación colectiva de muchos procesos a uno donde se aplica una operación matemática sobre los datos que se transmiten, obteniendo en el proceso destino el resultado de esta operación (por ejemplo la suma de las cuentas parciales mencionadas en un ejemplo anterior).

Si bien estas primitivas son muy útiles, priorizamos mantener el lenguaje lo más simple posible. Además, si un problema requiere el uso de estas directivas, se pueden construir a partir de las primitivas especificadas. Sin embargo, no descartamos la posibilidad de extender el lenguaje en versiones futuras con estas primitivas.

3.4. Tiempo virtual

Una característica fundamental del Da Vinci, es que se puede configurar la velocidad con que el robot se desplaza por la ciudad, permitiendo realizar ejecuciones instantáneas o que demoran lo suficiente para ver el resultado de cada instrucción que se ejecuta.

Este comportamiento resulta más complicado de realizar cuando coexisten varios robots que deben ser demorados o acelerados y que a su vez están ejecutando diferentes instrucciones. Para conseguirlo, el entorno LMRE cuenta con un "reloj virtual". En este esquema, cada instrucción LMRE tiene un costo en unidades virtuales de tiempo (que en principio es de uno). De esta manera, puede ajustarse la tasa de refresco del entorno de acuerdo a ese reloj virtual y restringir las instrucciones que son ejecutadas por cada robot a las que llegan al momento de tiempo simulado.

Esta característica permite también obtener métricas sobre las soluciones desarrolladas, incorporando la posibilidad de tratar conceptos como eficiencia a partir del entorno. Incluso a futuro puede permitirse la configuración de diferentes costos para cada robot, permitiendo representar escenarios de cómputo heterogéneo.

4. Modelos de programación en LMRE

En una estructura como la de la Figura 2, las primitivas tratadas en las secciones previas 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.

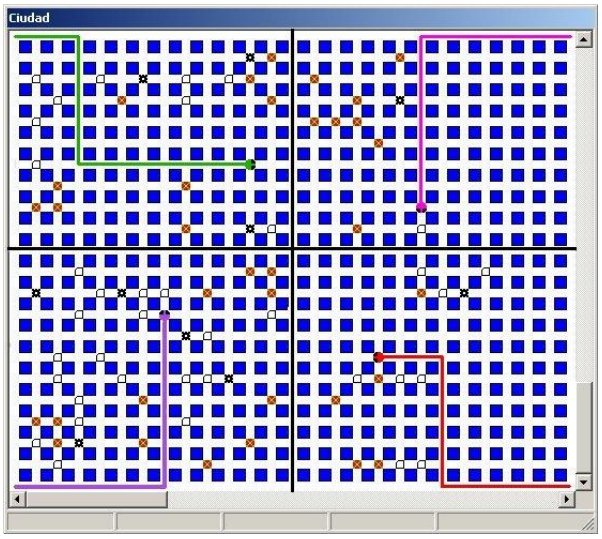

Figura 3. La imagen muestra la ciudad dividida en 4 areas exclusivas.

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

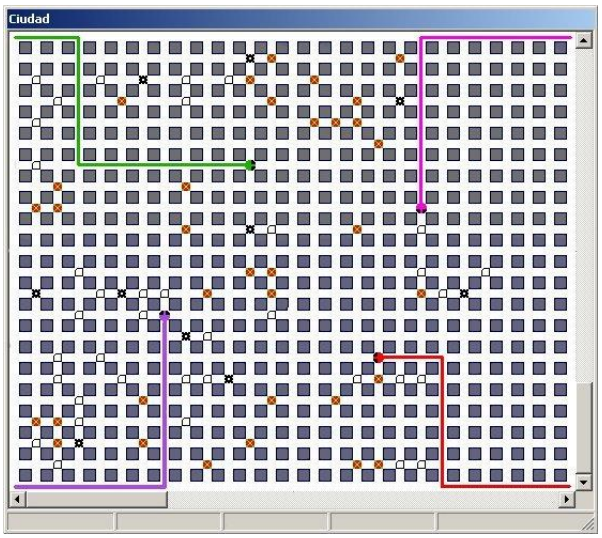

Figura 4. En esta imagen se ve la ciudad compartida por los 4 robots, diferenciados por su color.

Por último en la Figura 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.

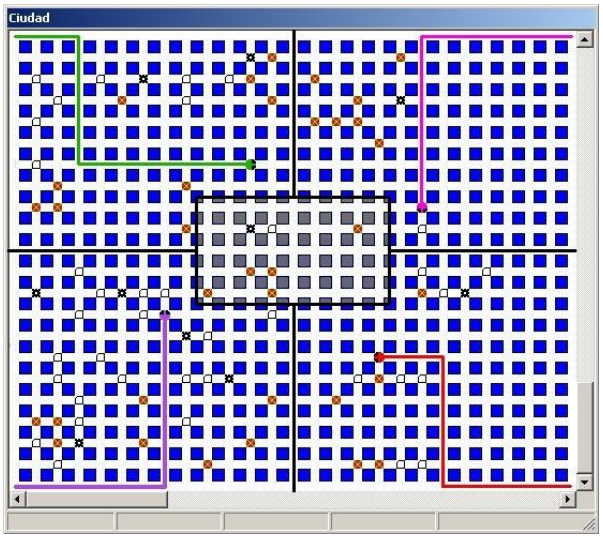

Figura 5. En la imagen pueden observarse áreas privadas para cada robot y un área compartida en color gris.

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

5. Implementación del entorno

El entorno se está desarrollando en el lenguaje de programación Java. La razón por la que se seleccionó este lenguaje tiene que ver con dos aspectos fundamentales: primero, Java es el lenguaje multiplataforma que más difusión tiene, lo cual constituye una ventaja significativa en el hecho que es poco probable que un alumno deba instalar librerías adicionales en la computadora donde quiera ejecutar el entorno debido a la alta probabilidad de que estas ya se encuentren instaladas; en segundo lugar, los alumnos de la Facultad de Informática que participan en el desarrollo del entorno tienen una sólida formación en programación orientada a objetos, y poseen experiencia desarrollando con este lenguaje.

En cuanto a las características que hacen a Java un lenguaje apto para el proyecto, éste 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.

5.1. Desafíos

Aunque el entorno LMRE está pensado para dar soporte a la enseñanza de conceptos de concurrencia en CS1, existe la necesidad de un curso completamente basado en multiprocesadores para expandir los conceptos necesarios para incorporar HPC. La currícula de las carreras de informática de la UNLP posee materias en 3 y 4 año que profundizan y estudian en detalle estos temas. Sin embargo, se considera oportuno contar con una extensión del LMRE que sirva en cursos más avanzados de concurrencia. Para ello 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.

5.2. Proyección 2012, grado de avance

El entorno LMRE se encuentra en desarrollo por un equipo formado por docentes y alumnos de la facultad de informática de la UNLP. Actualmente se ha completado la etapa de migración de las funcionalidades del entorno DaVinci original al nuevo lenguaje, y se está trabajando en incorporar las características detalladas en este trabajo. Se espera tener una versión de prueba para mitad de 2012, lo que permitirá comenzar a realizar evaluaciones con alumnos durante el segundo semestre de dictado de CS1.

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.
- Desarrollar una biblioteca de ejemplos clásicos de concurrencia/paralelismo sobre el entorno LMRE.

Referencias

[1] R. Champredonde and A. De Giusti, "Design and implementation of the visual da vinci language". Tesina de grado Facultad de Informática UNLP, 1997.

[2] V. Pankratius, C. Schaefer, A. Jannesari, and W. F. Tichy, "Software engineering for multicore systems: an experience report", Proceedings of the 1st international workshop on Multicore software engineering, pp. 53–60, 2008.

[3] T. Murphy, "High-performance computing in high schools?", IEEE Distributed Systems Online, vol. 8, no. 8, pp. 1–3, 2007.

[4] A.-C. J. C. T. Force, "Computer science curriculum 2008: An interim revision of cs 2001", tech. rep., ACM Press, dec 2008.

[5] C. Ivica, J. T. Riley and C. Shubert, "StarHPC - Teaching Parallel Programming within Elastic Compute Cloud", Proceedings of the ITI 2009 31st. Int. Conf. on Information Technology Interfaces, 353-356, 2009.

[6] 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.

[7] T. R. Gross, "Breadth in depth: a 1st year introduction to parallel programming", in Proceedings of the 42nd ACM technical symposium on Computer science education, pp. 435–440, 2011.

[8] A. De Giusti, "Algoritmos, datos y programas con aplicaciones en Pascal, Delphi y Visual Da Vinci", Pearson Education and Prentice Hall, 1 ed., 2001.

[9] A. De Giusti, L. Lanzarini and M.C. Madoz, "Abstract machines in a first course of computer science", Proceedings of 11th International Symposium "Computer at the University" – Zagreb – Yugoeslavia – Pag. 283-291 – 1990.

[10] 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.

[11] 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.

[12] 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.

[13] F. Leibovich, L. De Giusti, M. Naiouf, "Parallel Algorithms on Clusters of Multicores: Comparing Message Passing vs Hybrid Programming", WorldComp'11, Julio 2011.

Dirección de Contacto del Autor/es:

Laura De Giusti
III-LIDI - Facultad de Informática UNLP
50 y 120 – 2do Piso
La Plata- Argentina
e-mail: ldgiusti@lidi.info.unlp.edu.ar
sitio web: http://www.lidi.info.unlp.edu.ar

Fernando Emmanuel Frati
III-LIDI - Facultad de Informática UNLP
50 y 120 – 2do Piso
La Plata- Argentina
e-mail: fefrati@lidi.info.unlp.edu.ar
sitio web: http://www.lidi.info.unlp.edu.ar

Fabiana Leibovich
III-LIDI - Facultad de Informática UNLP
50 y 120 – 2do Piso
La Plata- Argentina
e-mail: fleibovich@lidi.info.unlp.edu.ar
sitio web: http://www.lidi.info.unlp.edu.ar

Mariano Sanchez
III-LIDI - Facultad de Informática UNLP
50 y 120 – 2do Piso
La Plata - Argentina
e-mail: msanchez@lidi.info.unlp.edu.ar
sitio web: http://www.lidi.info.unlp.edu.ar

María Cristina Madoz
III-LIDI - Facultad de Informática UNLP
50 y 120 – 2do Piso
La Plata - Argentina
e-mail: cmadoz@lidi.info.unlp.edu.ar
sitio web: http://www.lidi.info.unlp.edu.ar

Laura De Giusti es Doctora en Ciencias Informáticas, docente de cursos de algoritmica y programación concurrente. Es Investigadora y autora de trabajos del área de cómputo de altas prestaciones.

Fernando Emmanuel Frati es Licenciado en Análisis de Sistemas y becario de CONICET desde el año 2008. Actualmente desarrolla su trabajo de tesis doctoral en temas relacionados a cómputo de altas prestaciones.

Fabiana Leibovich es Licenciada en Sistemas y docente en cursos de algoritmica, ingreso y programación concurrente. Es becaria de UNLP y está realizando su doctorado en Ciencias Informáticas en temas relacionados a cómputo de altas prestaciones.

Mariano Sanchez es alumno regular de 2do año de la carrera Licenciatura en Sistemas de la Facultad de Informática de la UNLP. Desde principios de 2011 se desempeña como pasante del III-LIDI.

Cristina Madoz es Especialista en Tecnología Informática aplicada en Educación. Investigadora y autor de trabajos relacionados con el proceso enseñanza/aprendizaje en modalidad no presencial. Docente en cursos de algoritmica e ingresos con más de 25 años de antigüedad.