LIDI Multi Robot Environment

Support software for concurrency learning in CS1

Armando De Giusti, Fernando Emmanuel Frati, Mariano Sanchez, Laura De Giusti
Institute of Research in Computer Science (III-LIDI)
National Commission of Scientific and Technical Research (CONICET)
School of Computer Science, UNLP
La Plata, Argentina
{degiusti, fefrati, msanchez, ldgiusti}@lidi.info.unlp.edu.ar

***Abstract*— An interactive visual environment for teaching the concepts of concurrence and parallelism in an introductory course to algorithms is presented. The Lidi MultiRobot Environment (LMRE) is an evolution of the Visual Da Vinci environment that is extensively used in the introductory course to programming in several universities in Argentina.**

The paper analyzes the problem of the technology change introduced by multicore processors and their impact on programming. Also, an environment function is described, as well as the primitives to be used when programming concurrent applications.

Lastly, implementation-related aspects of the prototype currently being tested are described, as well as prototype evolution aimed at using it in more advanced concurrency courses.

Keywords- Algorithms; Concurrency; Parallelism; Sequential Programming; Parallel Programming

I. INTRODUCTION

A typical course on algorithms is focused on the expression of control through a reduced number of primitives and the introduction of basic data structures.

Typically, an only control flow is analyzed using basic efficiency metrics given by the runtime of a given algorithm for a certain set of input data.

Algorithm verification is quite straightforward, and their validation is possible if some basic structured programming principles are followed [1]. Concepts such as modularization, documentation or reuse are incorporated as "good programming practices". Problem analysis is focused on finding a correct and, if possible, efficient algorithm expression to find the solution.

At all times, there is a control, one-thread underlying architecture, with an only clock.

However, this approach leaves out two important aspects:

A. Current processors have a parallel architecture

The typical architecture of multicore processors, as Figure 1 shows, has several elements to be considered:

- Each core has its own clock, local memory, and arithmetic logic unit.
- There are globally and locally shared memories (e.g., between two cores), which means that memory access is non-uniform (NUMA).
- An application that will be run on a processor as the one shown in Figure 1 can be solved as multiple control threads by being concurrently run on the various cores.
- Clearly, this requires solving (at least) two basic problems: dividing the application into tasks and assigning these concurrent tasks to the cores, and coordinating (communication and synchronization) the cores to solve the application.

This brief analysis shows that the underlying "parallel machine" has to be taken into consideration to solve the application, which impacts every software level, especially languages, compilers, and operating system [2].

On the other hand, classic algorithm efficiency metrics will have to take into account the number of cores used, their performance, and the suitability of the concurrent division of the application to reduce idle times [3].

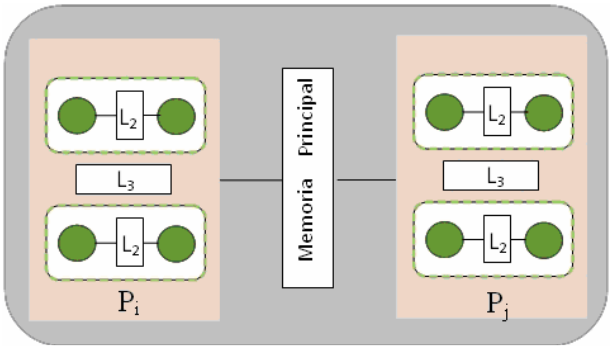

Figure 1. Multicore structure diagram

B. real-world problems are basically concurrent

Concurrency is the property that have some problems to be divided (total or partially) into smaller tasks that can be

executed at the same time. When these tasks are executed simultaneously by different processors, it is said that the program runs in parallel.

The fact that real-world problems are basically concurrent must be considered: databases, programming languages, operating systems, Web servers, commercial applications — they can all benefit if they are solved parallelly.

Thus, even though newer architectures are more complex, they also allow implementing solutions that are more efficiently adapted to the application model.

From this, it is clear that programming teaching poses multiple challenges. In particular, the "typical" courses on algorithms will have to be redesigned.

The efficient execution of an application on an architecture as the one shown in Figure 1 will almost invariably lead to parallel programming. Sequential programming will only be used in isolated cases, since it would result in having N-1 idle processor cores.

In this sense, the last recommendation of the ACM/IEEE [4] for computer science curricula identifies concurrency as a topic of interest and proposes adding parallel programming concepts to the first programming course taken by students without significantly changing its contents (structured programming, then object-oriented programming, and then parallel programming).

There are various proposals aimed at satisfying this recommendation. The authors in [5] acknowledge that there is a need for the development of tools that help dealing with high-performance computation problems. They propose a software application that helps develop and run parallel programs in HPC clusters, on hardware rented to Amazon, and they use virtual machine images for each course. Even though the framework that they developed is scalable and cost-effective, it is aimed at facilitating the management of the resources required for teaching parallel programming, rather than helping students learn its principles.

Other authors present a class strategy based on visualization tools for the conceptualization of parallel programs [6]. In this case, the tool proposed is an application that allows entering the code and viewing it by means of UML diagrams. It can be used to visually identify candidates to concurrent code sections (such as vector initialization). This proposal helps with the analysis of the parallel code, but students still need to know the language with the specific primitives that support concurrency.

More recently, the authors of [7] presented a study of the last three years of a CS1 course in which concurrency concepts were introduced through simple case studies to be implemented in Java. In the paper, the authors highlight the importance of teaching students general concurrency concepts to awaken their interest in those topics, as well as the impossibility of dealing with more advanced issues, such as efficiency metrics, at an early stage of their studies.

In this paper, an alternative to these approaches is presented – an interactive visual environment to be used in a typical CS1 course. In this environment, basic concurrent programming concepts are presented in a simple way, and students are guided to develop concurrent/parallel algorithms. The main difference between this approach and those mentioned above is that, rather than forcing students to learn the typical syntaxis of communication libraries, a set of simplified primitives is provided through the environment to help students learn concurrency concepts.

In Section II, a conceptual description of the LMRE environment is detailed, and the programming primitives that students can use are explained. In Section III, some implementation aspects are discussed, and in Section IV, conclusions and future research lines are presented.

II. CS1 COURSE AT THE UNLP

This course is currently given at the School of Computer Science of the UNLP. It is an annual course that is called "Algorithms, Data and Programs". Each year, it serves more than 1,000 students. It is based on a classic, learning-focused model to teach algorithm expression, introduction to linear and non-linear data structures, and introductory concepts related to modularization, efficiency analysis, abstraction and reuse.

During the 2nd semester, object paradigm concepts are introduced.

Practical activities are initially carried out with a proprietary environment, Visual Da Vinci [8], and then with Pascal and Delphi.

Starting in 2010, a gradual change in 3 stages has been proposed. The stages are:

- Incorporation of new processor architecture descriptions, as an evolution of the Von Neumann machine, and introduction to concurrency basic concepts.
- Development of a visual environment for the expression of concurrent and parallel algorithms as an evolution of Visual Da Vinci (VDV). This new development should support communication and synchronization mechanisms, both through shared memory and messages.
- Curriculum redesign to present concurrent/parallel programming as the standard, and sequential programming as the exception.

A. Purpose of the LMRE environment

Based on the analysis discussed in the Introduction and our experience in the development ans use of Visual Da Vinci [1] [8] [9], an evolution of the initial algorithm programming environment used by students of a CS1 course is proposed in order to incorporate the key concept of concurrency.

This means that the environment should allow students to naturally work with two basic aspects:

- Communication among concurrent processes (which usually involves cooperation).
- Synchronization among concurrent processes (which usually involves competitiveness or dependence). For synchronization to be possible, the expression of

explicit and/or implicit mutual exclusion mechanisms must be possible.

To study the communication and synchronization among the algorithms ("processes") residing in the cores of an architecture as the one shown in Figure 1, students need to acquire the relevant concepts and develop 3 essential models:

- Shared memory.
- Messages.
- Hybrid (algorithms that use shared memory as well as messages).

The LMRE environment is then set to help students express concurrent algorithms (to make learning easier, for CS1 students, these algorithms will be fully parallel, with one process linked to one processor) that can communicate and/or synchronize through messages and shared memory [10] [11] [12] [13].

B. *Features*

The environment allows defining a "city", represented by a grid of N streets x M avenues.

A number of K robots can also be defined; these robots can move around the city. Each robot has a unique identifier (ID). A maximum number of R robots is set.

The city can have Exclusive Areas (EA). For CS1, only one given robot can move in an EA. Shared Areas (SA) can also be defined, accessible to all robots.

City "corners" may have two types of objects (flowers and candies).

The initial distribution of the objects can be done automatically by the system or manually by the programmer.

Robots can store an unlimited number of objects.

Robots can perform basic actions on the objects (Put down / Pick up / Count), and can inform the results of their activities.

It should be noted that, if N=M=100, K=1 and the entire city is available for an only robot, the result would be the implementation defined for Visual Da Vinci.

Since our purpose is working with multiple robots in the city, the features of message communication actions and access to shared resources (city corners, objects, counters) need to be added.

When there are K robots (K > 1) working in a shared area (that can be the entire city), the problems of mutual exclusion on shared memory will have to be solved.

When there are independent robots working in exclusive areas in the city, as their collaboration to solve some problem, there will be message communication.

Both mechanisms can co-exist if the city has several exclusive areas and one shared area. This case would be a close approximation to the architecture model shown in Figure 1.

C. *Primitives*

The 4 essential primitives provided by the environment to specify concurrency are:

Send message (SM): this allows a robot to send a message to another robot (identified by ID). After sending the message, following the synchronous model, the robot waits until the target robot receives the message (synchronization). If messages are sent to a special ID, they become broadcast messages.

Receive message (RM): this indicates that a robot will wait until it synchronizes with the message sent by another robot. To this end, the primitive receives an ID parameter that indicates the ID number of the robot that must send the message, or a special value that indicates that it can come from any robot.

Block resource (BR): this indicates that the robot requests the exclusion of a resource to have exclusive access to it. Typically in a Shared Area, corner occupation (which allows picking up or putting down objects) must be exclusive for a robot.

Free resource (FR): this indicates that the robot frees the resource (for example, the corner that it was occupying). This is also a commonly used primitive in Shared Areas.

In a structure such as the one shown in Figure 2, these primitives allow presenting problems of "independent processes that communicate through messages", as in the case of 4 robots on exclusive areas carrying out tasks (e.g., picking up the flowers found within their areas) and then communicating their partial results to a fifth robot that will report the results.

Figure 2. The image shows the city divided in 4 exclusive areas, identified by the colors of the robots.

There can also be "concurrent runs" problems within a shared area as the one shown in Figure 3, where several robots try to carry out a task (e.g., place candies on certain corners),

starting from various locations in the city, and attempting to occupy the same corner at the same time.

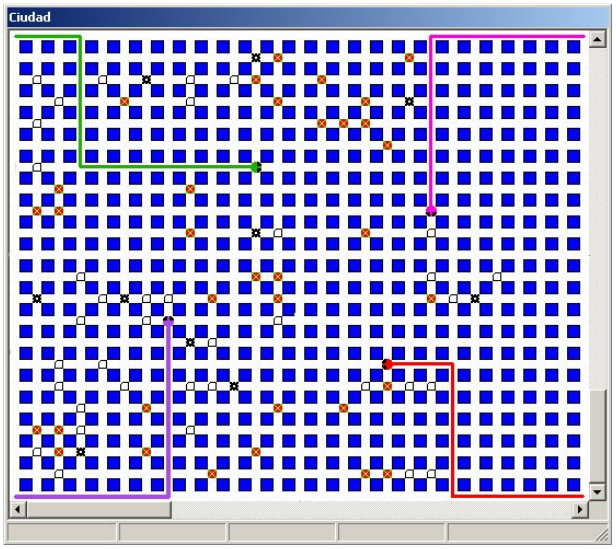

Figure 3. This image shows (in blue) the city shared by the 4 robots, identified by their color.

Finally, Figure 4 shows an Exclusive Area and Shared Area distribution, which allows combining independent processes in the exclusive areas, followed by all robots going to the shared area to finish the job. This clearly shows the concepts of hybrid programming.

Figure 4. The image shows the exclusive areas for each robot (identified by their color), and a shared area in blue.

It should be noted that the environment allows the programmer to create multiple instances of the same code for different robots.

III. ENVIRONMENT IMPLEMENTATION

The environment is being developed in Java programming language. This language was selected based on two main aspects: Firstly, Java is the multiplatform language that is most widely used, providing a significant advantage in the fact that it is unlikely that students will have to install any additional libraries in the computer where they will be running the environment (it is highly likely that they are already installed). Secondly, the students of the School of Computer Science participating in the development of the environment have a solid background on object-oriented programming and have experience in this development language.

As regards the suitability of Java for the project, it offers the Thread class that allows building concurrent programs (no additional tool is required), and it provides thread synchronization methods. Java has no control over possible situations that may occur in concurrent programs (blocking, atomicity violations, race conditions); it does, however, provide mechanisms that allow developers to algorithmically control these. These aspects should be considered when implementing the multirobot environment to prevent their occurrence.

1) Challenges

Even though the LMRE environment is devised to support the teaching of concurrency concepts in CS1, there is a need for a course that is entirely based on multiprocessors to expand on the necessary concepts to incorporate HPC. The curriculum of computer science courses at the UNLP includes academic subjects in the 3rd and 4th years that deal with these topics in detail. Despite this, it would be advisable to have an extension of LMRE that can be used in more advanced concurrency courses. To this end, the addition of functionalities of academic interest is planned:

- Blocking arrangements (semaphores) that allow handling barriers or FORK-JOIN schemes.
- Selective mutual exclusion. In the city, this leads to the incorporation of areas shared among certain robots and areas that selectively exclude robots.
- Priority handling for access to shared resources. Priority can be a static attribute of the robot or a dynamic attribute based on the context and the task to be carried out by the robots.
- Communication through asynchronous messages and their reflection on the status of each robot.

2) Projection for 2012, progress to completion

A prototype of the LMRE environment is currently being tested by a team formed by teachers and students of the School of Computer Science of the UNLP. The migration of the functionalities of the original Da Vinci environment to the new language has been completed, and the functionalities detailed in this paper are currently being tested. In August 2012, the environment will be introduced for systematic use by the students attending the second semester of the course CS1, and in 2013 its use will be generalized, after an assessment of the experience in 2012.

IV. CONCLUSIONS AND LINES OF WORK

The interactive LMRE visual environment has been presented. This environment is proposed for teaching the concepts of concurrency and parallelism in an introductory course to algorithms, considering the technology change caused by the introduction of multicore processors and their impact on programming.

When describing environment functionalities, the primitives to be used when programming concurrent applications were discussed and the rationale behind our choices explained.

Implementation aspects were presented, as well as the extensions of the environment for advanced courses.

There are two major lines of work:

- Incorporating time as an attribute of the various operations to be carried out by the robots to study algorithm speed-up, efficiency and performance.
- Developing a library of typical concurrency/parallelism examples in the LMRE environment.

ACKNOWLEDGMENT

Work partially supported by MICINN (TIN2009-14317-C03-01).

REFERENCES

[1] R. Champredonde and A. De Giusti, "Design and implementation of the visual da vinci language". Graduate Final Work, School of Computer Science, 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 – Yugoslavia – Pp. 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, July 2011.