ECMRE: Extended Concurrent Multi Robot Environment

Juan Castro[1(✉)], Laura De Giusti[1,2], Gladys Gorga[1], Mariano Sánchez[1], and Marcelo Naiouf[1]

[1] School of Computer Science, Computer Science Research Institute LIDI (III-LIDI), UNLP, La Plata, Argentina
camaju_25@hotmail.com,
{ldgiusti,ggorga,msanchez,mnaiouf}@lidi.info.unlp.edu.ar
[2] Scientific Research Agency of the Province of Buenos Aires (CICPBA), La Plata, Argentina

Abstract. ECMRE is an extension of CMRE (Concurrent Multi Robot Environment) that adds features related to current parallel architectures: processor heterogeneity, energy consumption, processor speed change techniques in relation to temperature and/or energy consumption.

ECMRE allows incorporating the topics of concurrency and parallelism in a simple and entertaining manner in beginner classes in the courses of Computer Science by means of a graphic and interactive environment.

An initial test was carried out in a course with 42 students to analyze how they adapt to this new environment and how they can use it.

Keywords: Concurrency · Parallelism · Heterogeneous processors
Parallel algorithms · Energy consumption

1 Introduction

Concurrency has been a central issue in the development of Computer Science, and the mechanisms used to express concurrent processes that cooperate and compete for resources have been in the core curriculum of Computer Science studies since the seventies, in particular after the foundational works of Hoare, Dijkstra and Hansen [1–3]. On the other hand, parallelism, understood as "real concurrency" in which multiple processors can operate simultaneously on multiple control threads at the same point in time, was for many years a possibility that was limited by available hardware technology [4]. Classic Computer Science curricula [5–7] included the concepts of concurrency in various areas (Languages, Paradigms, Operating Systems), but parallelism was almost entirely omitted, except to present the concepts of distributed systems.

Changes in technology have produced an evolution of the major topics in Computer Science, mainly due to the new applications being developed from having access to more powerful and less expensive architectures and communications networks [8]. For this reason, international curricular recommendations mention the need to include the topics of concurrency and parallelism from the early stages of student education, since all architectures and real systems with which they will work in the future will be essentially parallel [9]. However, parallel programming (and the essential concepts of concurrency)

A. E. De Giusti (Ed.): CACIC 2017, CCIS 790, pp. 285–294, 2018.
https://doi.org/10.1007/978-3-319-75214-3_27

is more complex for students who are starting their studies, and new strategies are required to teach the topic.

Given the stimuli to which students are exposed from an early age, be it through video games, computers, mobile phones, tablets, or any other electronic device, the use of interactive tools to teach core concepts to students in a CS1 course [9–11] has become essential [12]. In this sense, the possibility to take the initial steps in the world of programming through a graphic and interactive environment allows reducing the gap that traditionally existed between abstraction and the possibility of seeing a graphic representation of how the concepts being learned are applied in an environment that is conceptually similar to those used in everyday life [8, 13].

CMRE is a graphic environment that has a set of robots that move within a city, and it has allowed teaching the basic concepts of concurrency and parallelism in a beginner's course in Computer Science. In a previous article [14], the idea of adding advanced features commonly found in current parallel architectures (such as heterogeneity, energy consumption, temperature generated) to the environment was discussed. As a continuation of that work, we have implemented such an extension and created ECMRE (Extended Concurrent Multi Robot Environment), which is presented here.

This article is organized as follows: Sect. 2 details the advanced features present in modern parallel architectures that have been included in the environment; Sect. 3 describes the original version of CMRE; Sect. 4 discusses the extensions developed to create ECMRE; and Sect. 5 presents a test carried out with the new environment in a first-year course. Section 6 discusses the conclusions.

2 Advanced Features of Parallel Architectures

Current parallel architectures come with advanced features that should be included when teaching the basic concepts of concurrency and parallelism. In particular, architecture heterogeneity and energy consumption.

2.1 Heterogeneity in Parallel Architectures

Since the early computers, there has been an ever-present desire to increase machine computational power. However, it is currently hard to increase processor speed by increasing their clock rate. Hardware architects face two issues: heat generation and energy consumption. The solution to this problem introduced by designers has been integrating two or more computational cores within a single chip, which is known as multicore processor. Multicore processors improve application performance by distributing work among the available cores [15, 16].

Currently, research is focusing mainly on heterogeneous multicore architectures (i.e., architectures whose cores have different performance and energy consumption characteristics and which may or may not use different sets of instructions), since having different types of cores allows optimizing performance and, when tasks are appropriately distributed among cores, higher performance/energy ratio efficiencies are achieved.

In this type of architectures, heterogeneity is present in various aspects, most significantly in core computational power (computation speed), memory access time, and communication speed among cores. These three aspects determine the time required to execute the instructions in each core, and thus, the same sentence executed by two different cores can take different times. On the other hand, since there is a certain degree of independence of the features that cause heterogeneity, not all instructions are affected in the same proportion. That is, a floating point operation that is run in core A, may take a fourth of the time it takes when run in core B, while a writing operation may take half the time when run in it.

2.2 Energy Consumption in Parallel Applications

Energy consumption is a key aspect of current processors. In general, the performance of a parallel algorithm is not measured only in its execution time, but also in energy consumed. Thus, there will be Flops/Watt or Flops/Joule ratios corresponding to a relation between computation and instant power or total energy [17, 18].

It is important to teach Computer Science students to always use consumption metrics as an indicator of algorithm quality. Additionally, they should also understand the automatic mechanisms developed by processors according to the temperature reached (which is a direct function of the energy consumed in a period of time) [17].

There are performance adjustment techniques used in current processors that consider energy consumption, temperature and other values as indicators for decision-making. Overclocking and underclocking are two of the most widely used techniques to increase or decrease processor clock rate in order to increase performance or standardizing consumption and temperature values when the processor is overloaded.

3 Current Version of CMRE

The main features of CMRE can be summarized as follows [15, 19]:

- There are multiple processors (robots) that carry out tasks and that can co-operate and/or compete. They represent the cores of a real multiprocessor architecture. These virtual robots can have their own clock, and different times for carrying out their specific tasks.
- The environment model (“city”) where the robots carry out their tasks supports exclusive areas, partially shared areas and fully shared areas. An exclusive area allows only one robot to move in it, a partially shared area specifies the set of robots that can move in it, and a fully shared area allows all robots defined in the program to move in it.
- If only one robot is used in an area that encompasses the entire city, the scheme used in Visual Da Vinci is repeated [20, 21].
- When two or more robots are in a (partially or fully) shared area, they compete for access to the corners on their runs, and the resources found there. For this, they must be synchronized.

- When two or more robots (in a common area or not) wish to exchange information (data or control), they must use explicit messages.
- Synchronization is done through a mechanism that is equivalent to a binary semaphore.
- Mutual exclusion can be generated by stating the areas reached by each robot. Entering other areas in the city, as well as exiting them, is not allowed.
- The entire execution model is synchronous and allows the existence of a cycle virtual clock which, in turn, allows assigning specific times for the operations, simulating the existence of a heterogeneous architecture.
- The environment allows executing the program in a traditional manner or with step-by-step instructions, giving the user detailed control over program execution to allow them controlling typical concurrency situations such as conflicts (collisions) or deadlocks.
- In the step-by-step mode, the effect of the operations can be reflected on physical robots, communicated through Wi-Fi. The physical robots have Linux as operating system, which allows running an http server implemented on NodeJS [22]. Thus, the environment communicates with the robots (each physical robot corresponds to a virtual one in the environment). These are point-to-point, two-way communications, i.e., the environment sends instructions to the physical robot and then the robot sends its response to the environment stating that the instruction given has been fulfilled.

4 CMRE Extension

ECMRE (Extended Concurrent Multi Robot Environment) is an extended version of its predecessor (CMRE) and, as such, it adds the concepts of heterogeneous multicore architectures, energy consumption, processor temperature, and overclocking and underclocking techniques to it.

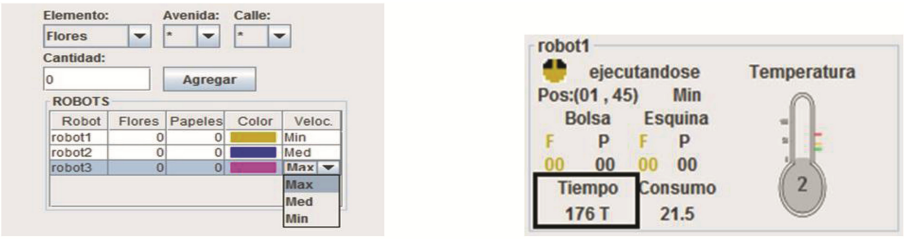
Students can work on ECMRE by representing different types of multicore architectures and watch in a graphic and interactive manner the information related to operation time for each robot and variations in consumption and temperature values corresponding to algorithm runs. By analyzing this information, students can modify their algorithms (for example, they can balance workloads) to obtain solutions that are efficient as regards energy consumption and temperatures reached.

4.1 Representing Parallel Architecture Heterogeneity

In this section, the main adaptations made to CMRE to address speed and energy consumption aspects in ECMRE are described.

Processor Performance. In ECMRE there is a *Details* area (Fig. 1(a)) to adjust the general parameters for the application and for each robot in particular. It includes a table called ROBOTS that has a column that allows defining the speed for each robot. This simplifies working with the robots, which could present variations in their performance

because they will execute algorithm instructions at different speeds. There are 3 speeds that can be selected, and they are related as follows:



(a) (b)

Fig. 1. Areas in ECMRE: (a) *Details*, where each robot is configured, (b) *Execution Information*, where the execution time for the robot is displayed.

- *Max* is the maximum speed available.
- *Med* is half the maximum speed.
- *Min* is half the medium speed.

In ECMRE, T is defined as a unit of time (equivalent to 100 ms) to measure robot/processor performance. We decided to use a subset of 10 primitive instructions (*blockCorner, put-downFlower, put-downPaper, right, sendMessage, freeCorner, move, receiveMessage, pick-upFlower* and *pick-upPaper*) classified based on their complexity, and an execution time consistent with robot speed was assigned to them.

Also, ECMRE has an *Execution Information* area that displays updated information for each robot during the execution of the algorithm (Fig. 1(b)). One of the entries there corresponds to the execution time measured in T units. This information is extremely useful to assess algorithm performance.

The ability of controlling the speed for each robot combined with the assignment of times to primitive instructions in ECMRE allows working with robots/processors with different performances, simulating one of the features of heterogeneous multicore architectures.

Energy Consumption. ECMRE adds a new section called Robot/Processor where energy consumption options can be set (in Joules) for a set of primitive instructions (Fig. 2(a)). Based on this, each robot stores its own consumption information and updates it during algorithm execution, which is displayed in the Execution Information area (Fig. 2(b)).

Processor speed is a factor that affects consumption (as speed increases, so does consumption). In ECMRE, the consumption generated by robot r to execute instruction i is given by the consumption of i previously specified multiplied by a coefficient that represents the speed of r at that moment (0.25 for *Min*, 0.5 for *Med* and 1 for *Max*).

Fig. 2. ECMRE (a) Energy consumption by instruction in the Robot/Processor section, (b) *Execution Information* area, where robot energy consumption information is displayed

4.2 Temperature Representation

In the Robot/Processor section mentioned above, the temperature (in Celsius) produced when executing each instruction of the set of primitives can also be set for each robot (Fig. 3(a)).

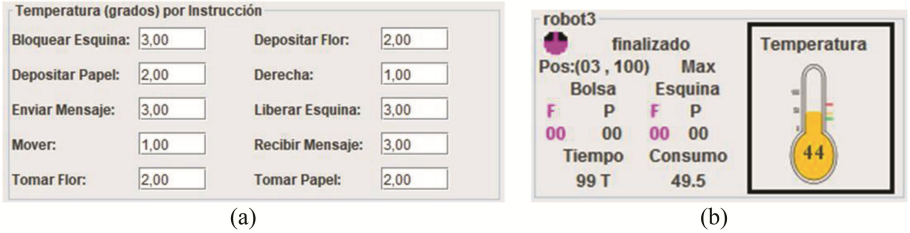


Fig. 3. ECMRE (a) Temperature by instruction in the Robot/Processor section, (b) *Execution Information*, where robot temperature information is displayed. (Color figure online)

The required logic has been implemented so that each robot records and updates its temperature and then this information can be viewed while the algorithm is being executed. To this end, a thermometer was added to the *Execution Information* area of the robot (Fig. 3(b)) to display the temperature with a numeric value. The thermometer changes its color (gray, green, orange and red) as temperature increases or decreases, where gray represents the minimum temperature and red, the maximum.

This feature is also affected by robot speed (as speed increases, so does temperature). In ECMRE, the temperature of robot r after executing instruction i is given by its previous temperature plus the temperature generated by instruction i previously specified, and this total is multiplied by a coefficient that represents the speed of r at that moment (0.92 for *Min*, 0.95 for *Med* and 0.98 for *Max*).

4.3 Representation of Overclocking and Underclocking Techniques

On the other hand, ECMRE allows for the possibility of robots using *overclocking* and *underclocking* techniques while the algorithm is being run. In the *Robot/Processor* section in ECMRE, the following parameters that enable this functionality are set (Fig. 4):

Fig. 4. ECMRE - Performance adjustment parameters in the Robot/Processor section

- *TCase Max:* maximum processor temperature. If this value is exceeded:

– An underclock condition will be attempted to help decrease processor temperature.
– If underclock is not admitted, or if the current speed is already the minimum speed, the robot will stop until its temperature goes back to normal.

- *Use Overclock:* this enables the use of an overclocking operation in the robot.
- *Use Underclock:* this enables the use of an underclocking operation in the robot.
- *Verification Interval (VI):* this indicates how often the increase in energy consumption should be checked and, if necessary, the corresponding overclocking/underclocking operation applied. The value is expressed in units of time T.
- *Expected Consumption Variation (ECV):* this indicates the variation in energy consumption (in joules) expected for the time defined in *VI*. This value is used to apply overclocking/underclocking operations.

The performance adjustment algorithm in each robot works as follows: For each instruction that is executed, it assesses the following:

- If robot temperature is higher than the specified maximum temperature (TCase Max):

– If the robot supports underclocking and its current speed is not the minimum speed, an underclock operation is applied to help decrease temperature. Note that if robot speed is already Min, it cannot be further reduced.
– Otherwise, the robot stops until its temperature cools down to 25°C and then resumes processing.

- If the increased consumption recorded during the verification interval (VI) exceeds the expected consumption variation (ECV):

– If the robot supports underclocking and its current speed is not the minimum speed, an underclock operation is applied to help decrease energy consumption and temperature. Note that if robot speed is already Min, it cannot be further reduced.

- If the increased consumption recorded during the verification interval (VI) is lower than the expected consumption variation (ECV):

– If the robot supports overclocking and its current speed is not the maximum speed, an overclock operation is applied to increase performance. Note that if robot speed is already Max, it cannot be further increased.

Overclock and underclock operations are not applied while consumption variation time is lower than the value set for *VI*. Every time an overclock or underclock operation is applied, or when the robot stops to cool down, consumption variation and elapsed time values are reset to 0.

4.4 Processor Logs

The new functionalities added to ECMRE may require an analysis after the algorithm is run to assess if the result obtained meets expectations. If execution time or energy consumption are higher than expected, modifications to the implemented solution may be proposed, or tasks could be reassigned among participating processors to achieve better results. To facilitate this analysis, ECMRE includes a new section called *Processor Log* that shows a user-friendly summary (through tables and charts) of the execution of an algorithm.

ECMRE records processor information while the algorithm is being run. In the *Robot/Processor* section, the option *Log Frequency* can be set to define how often (number of instructions) this information is recorded for each robot at runtime.

Recorded events include: processor stopped due to overload, overclock, and underclock. For each recorded log entry, the following robot information is stored: event (LOG, STOPPED, OVERCLOCK and UNDERCLOCK), speed before and after the event, temperature, and energy consumption.

The *Processor Log* section includes the following 5 subsections: Temperature, Temperature Chart, Consumption, Consumption Chart, and Consumption by Instruction Chart. For each of these, the user can select the robot to be analyzed.

5 Test Session with Students

ECMRE was presented in the course Programming Workshop of the School of Computer Science of the UNLP. This is a first-year course that consists of 3 modules. The third of these modules deals with the introduction to basic concurrent programming concepts, which is done through the use of CMRE.

The class started with a brief review of the concepts of concurrency and parallelism addressed by the environment, such as multicore architectures (both homogeneous and heterogeneous), energy consumption, temperature, performance adjustment techniques in processors, and load balancing. For each item, the concept and its significance in relation to current computer architectures were reviewed, and the new elements added to CMRE (temperature, consumption and speed) to create ECMRE, were introduced.

Once the theoretical review was completed, a practical activity was presented to be carried out under the supervision of the educator and the group of students. The students did not interact directly with the environment because it was still under development. The practical activity consisted in solving a model problem using ECMRE so that students could see how execution time, energy consumption and temperature changed based on different robot configurations. To this end, the tables, charts and logs described in the previous section were used.

At the end of the practical activity, each student answered a brief, anonymous survey intended as a first feedback from students, who will be the end users of the tool. This survey consisted of 5 questions answered on a Likert scale that goes from 1 (fully disagree) to 5 (fully agree). The survey was taken by the 42 students who attended the class on the day of the experience; the results obtained are listed in Table 1.

Table 1. Results of the survey taken by students of the Programming Workshop.

Question	Results obtained				
	Fully agree	Agree	Neither agree nor disagree	Disagree	Fully disagree
ECMRE helps learning the concepts presented	36%	57%	7%	0%	0%
Having a practical tool that allows viewing the theoretical content learned during the class through specific examples is useful	55%	38%	7%	0%	0%
The contents in ECMRE are organized and its use is intuitive	24%	52%	24%	0%	0%
The icons and charts used in the application are of the right size and match their associated function	48%	38%	14%	0%	0%
Having this application as a supplement to theoretical classes is beneficial	52%	36%	12%	0%	0%

6 Conclusions

The concepts of heterogeneity, energy consumption and temperature in parallel architectures are highly relevant, and we have presented an extension of CMRE (ECMRE) that allows including them.

ECMRE appears as a very useful tool to introduce these concepts in beginner classes in Computer Science courses of studies. To achieve this, two stages have been carried out. On the one hand, CMRE was modified to allow including these features in robots and, on the other, graphical tools were added to allow students view and then analyze this information in an easy and entertaining manner.

Thus, the complexity level of possible scenarios is increased, which poses a much more ambitious challenge that matches the technological reality of current processors.

References

1. Hoare, C.: Communicating Sequential Processes. Prentice Hall, Upper Saddle River (1985)
2. Dijkstra, Edsger W.: Finding the correctness proof of a concurrent program. In: Bauer, Friedrich L., et al. (eds.) Program Construction. LNCS, vol. 69, pp. 24–34. Springer, Heidelberg (1979). https://doi.org/10.1007/BFb0014652
3. Hansen, P.B.: The Architecture of Concurrent Processes. Prentice Hall, Upper Saddle River (1977)
4. Dasgupta, S.: Computer Architecture: A Modern Synthesis, Vol. 2: Advanced Topics, vol. 2. John Wiley & Sons, New York (1989)

5. ACM Curriculum Committee on Computer Science: Curriculum 1968: recommendations for the undergraduate program in computer science. Commun. ACM **11**(3), 151–197 (1968)
6. ACM Curriculum Committee on Computer Science: Curriculum 1978: recommendations for the undergraduate program in computer science. Commun. ACM **22**(3), 147–166 (1979)
7. ACM Two-Year College Education Committee: Guidelines for associate-degree and certificate programs to support computing in a networked environment. The Association for Computing Machinery, New York (1999)
8. Hoonlor, A., Szymanski, B.K., Zaki, M.J., Thompson, J.: An evolution of computer science research. Commun. ACM **56**, 74–83 (2013)
9. ACM/IEEE-CS Joint Task Force on Computing Curricula: Computer Science Curricula 2013. Report from the Task Force (2013)
10. ACM/IEEE-CS Joint Task Force on Computing Curricula: Computer Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering. Report in the Computing Curricula Series (2004)
11. ACM/IEEE-CS Joint Interim Review Task Force: Computer Science Curriculum 2008: An Interim Revision of CS 2001. Report from the Interim Review Task Force (2008)
12. De Giusti, L., Leibovich, F., Sanchez, M., Chichizola, F., Naiouf, M., De Giusti, A.: Desafíos y herramientas para la enseñanza temprana de Concurrencia y Paralelismo. In: Congreso Argentino de Ciencias de la Computación (CACIC), pp. 1585–1595. Fundación de Altos Estudios en Ciencias Exactas, Mar del Plata (2013)
13. AMD: Evolución de la tecnología de múltiple núcleo. http://multicore.amd.com/es-ES/AMD-Multi-Core/resources/Technology-Evolution (2009)
14. De Giusti, L., Leibovich, F., Chichizola, F., Naiouf, M., De Giusti, A.: Incorporando conceptos en la enseñanza de Concurrencia y Paralelismo utilizando el entorno CMRE. In: Congreso Argentino de Ciencias de la Computación (CACIC), pp. 1212–1221. Red de Universidades con Carreras en Informática (RedUNCI), Junín (2015)
15. Gepner, P., Kowalik, M.F.: Multi-core processors: new way to achieve high system performance. In: Proceeding of International Symposium on Parallel Computing in Electrical Engineering 2006 (PAR ELEC 2006), pp. 9–13 (2006)
16. Cool, M.M.: Programming models for scalable multicore programming (2007). http://www.hpcwire.com/features/17902939.html
17. Balladini, J., Rucci, E., De Giusti, A., Naiouf, M., Suppi, R., Rexachs, D., Luque, E.: Power characterisation of shared-memory HPC systems. Computer Science & Technology Series–XVIII Argentine Congress of Computer Science Selected Papers, pp. 53–65 (2013)
18. Brown, D.J.: Toward energy-efficient computing. Mag. Commun. ACM **53**(3), 50–58 (2010)
19. De Giusti, A., De Giusti L., Leibovich, F., Sanchez, M., Rodriguez Eguren, S.: Entorno interactivo multirrobot para el aprendizaje de conceptos de Concurrencia y Paralelismo. In: IX Congreso de Tecnología en Educación y Educación en Tecnología, pp. 74–81. TE&ET, Chilecito Argentina (2014)
20. Champredonde R., De Giusti A. "Herramienta visual para la enseñanza de programación". Congreso Argentino de Ciencias de la Computación (CACIC), pp. 520–531. Red de Universidades con Carreras en Informática (REDUNCI), San Luis (1996)
21. Champredonde, R., De Giusti A.: Design and implementation of the visual DaVinci language. In: Congreso Argentino de Ciencias de la Computación (CACIC), pp. 980–992. Red de Universidades con Carreras en Informática (REDUNCI), La Plata (1997)
22. Nodejs page: https://nodejs.org/api/http.html. Accessed 21 July 2017