LEARNING THE PRINCIPLES OF PARALLEL COMPUTING WITH GAMES

Mauricio Alvarez Mesa, Pau Bofill,

Friman Sánchez, Montse Farreras

Universitat Politècnica de Catalunya
email:{alvarez, pau, fsanchez, mfarrera}@ac.upc.edu

Abstract
The trend towards parallel computers requires a fundamental change in the way software is developed in order to maintain performance scalability. Because of that, it is required that most software developers have a solid knowledge on how to develop parallel programs. In this paper we present a methodology for learning parallel computing that gives priority to the general principles rather than technologies that use them. Parallel computing is presented as a specific case of the general coordination problem and, based on that, the fundamental issues of coordination systems are presented. Coordination is modelled as a cooperative game, in which learners (players) contribute to a common goal. Two games are presented as an example: the "orange game", and a game based on the "dining philosophers" problem. Those games only use ordinary materials (not computers) such as cards, drawing paper, colours and oranges, and allow to illustrate problems in coordination systems like mutual exclusion and deadlocks.

Key words: Workspaces for Active Learning, Peer Instruction, Active Learning in large classes, Parallel computing, Concurrency.

I INTRODUCTION

Computer science and engineering are facing an important change due to the emergence of multicore computer architectures. Following the current trend, in the near future almost all computers will be parallel computers [13]. This generates a radical change in the way software is developed, requiring that most programmers learn parallel programming. This topic has been regarded as complex, hard and, even, it is said that it requires "to think in a way that humans find difficult" [17].

In this paper we propose a methodology for learning parallel computing that starts from the fundamental principles rather than the technologies that use them. This methodology is based on the "Great Principles Framework" [6] which describes a set of fundamental

scientific principles of computing. Using this framework we model parallel computing as a coordination problem and use games to illustrate its principles.

Coordination deals with the problem of getting multiple entities cooperating toward a single result. It can be simulated as a game in which players reach a common objective when they coordinate in the right way. We have selected two games that can be used to illustrate fundamental issues of parallel computing such as mutual exclusion and deadlock avoidance.

The proposed games allow the participants to learn how to coordinate with each other to solve a common problem. Using games, participants can discuss and analyse the impact of strategy, rules and resources. This process allow learners to find solutions based on their prior knowledge of coordination and to create abstractions based on their own experience.

In a more general way, we propose that games can be used as a model for explaining the principles of (parallel) computing. We left as an open discussion how this methodology can be applied to other disciplines.

II WHY LEARNING PARALLEL COMPUTING IS IMPORTANT?

II.1 Multicore revolution

In the last 35 years the performance scalability of computing systems has been achieved through the scaling of integrated circuits technology. It has allowed to increase, at the same time, the speed of integrated circuits and the density of components[4]. The latter is widely know as the "Moore law"[16] and states that the number of basic components (i.e. transistors) on a chip will double every 24 months[1]. During that time software has not required an important modification to get advantage of the performance offered by technology scaling. Software developers have used this increasing performance to raise the level of abstraction in programming languages and to develop more powerful programs.

Nowadays, due to several restrictions in the technology used to fabricate microprocessor chips it is not possible to make faster computers at the same rate than in the past. But, the increase in components density (Moore's law) remains valid. Instead of making faster and more complex "single core" processors, chip makers are now fabricating slower and simpler "multi-core" processors. Multicore processors are multiprocessor systems integrated on the same chip. As a result of the multicore trend, it is expected than the number of cores per chip will increase with every technology generation [5, 3].

1Originally expressed as doubling every year, later corrected as doubling every two years, but its common use is that the performance of a computer system will double every 18 months

This represents a dramatic change from the software point of view. In order to benefit from the performance scaling offered by multicore architectures software needs to be redesigned to get advantage of parallel processors. Using multiple processors to solve a single problem is the main problem of the parallel computing domain. In the past, parallel computing was used only for some specific applications that require higher performance. Programming parallel architectures has been the task of a few expert developers that have the required skills to do that.

Currently there is an important ongoing research effort to create new tools and programming paradigms that facilitate the development of parallel software [1]. But ,it is clear that parallel computing has to become an essential part of the knowledge base of computing engineers. An important question that arises from there is, how to teach and learn parallel computing in the most effective way.

II.2 Teaching Parallel Computing

When dealing with the problem of how to teach parallel computing, educators face three main questions: What, when and where?

Because of the complexity of parallel computing, it is not clear what need to be learned. Parallel computing requires knowledge from different fields such as data structures and algorithms, computer architecture, programming languages and compilers and run-time systems. There are different proposals of what need to be learned such as formal methods that allow to describe correct parallel algorithms [14] or traditional synchronization topics in popular programming languages like Java [12].

When dealing the question of when and where, two main proposals have appeared: one is to devote a whole course to the topic of parallel computing and concurrency, and the other one, to include this topic on other courses that deal with some related content such as operating systems and databases [11].

III START FROM THE PRINCIPLES

Our approach is based on the idea that that parallel computing, as any other part of computing and sciences in general, can be easily learned with a clear definition of the principles and with a learning environment that inspires curiosity and intuition.

III.1 General principles of computing

Our methodology is based on the "Great Principles Framework" [6]. This framework helps to identify the essential knowledge in the computing field by defining a set of (overlapping) categories of principles. They can be used as a effective learning tool due to the fact that the set of active principles evolve more slowly than the technologies that are based on them [7]. Principles are grouped in seven categories: computation, communication, coordination, recollection, automation, evaluation and design.

The "Great Principles Framework" is based on the notion that there are principles that transcend computers and apply to computation in all fields [6]. In this framework computing science is seen as the study of fundamental properties of information processes: "Computers are the tool, not the object of study" [8]. This notion has an important consequence from the learning perspective: learning the fundamental principles of computing do not require the use of computers [2].

III.2 Parallel programming as a coordination problem

In the "Great Principles Framework" parallel computing can be included in the "coordination" category. Coordination is defined as the cooperation among multiple agents towards a single result. This definition fits very well with the purposes of parallel computing in which multiple computing elements (processors, nodes, systems) work together in order to solve a single problem. An important feature of seeing parallel computing as a coordination problem is that the learning process can benefit from the intuitive and pre-existing knowledge that learners have about coordination in natural systems [15]. Coordination activities are common in animals like flying birds, ants, bees and others, and also in human relations in which they are used to orchestrate interactions between several agents to produce a mutual outcome.

From the computing perspective, the learning objective is to understand coordination algorithmically in order to automate coordination processes in computing systems [9]. Achieving coordination between humans is relatively easy because humans have many different communication strategies, but achieving coordination between computing systems is more challenging because computers communicate with each other in a very restricted way [18]. Analysing coordination mechanisms in humans (and other animals) can help to understand the limitations that computers have in order to make a correct coordination. One of the main issues that make parallel computing difficult is that the limited way in which computers communicate with each other make that small programming errors have catastrophic effects and that those errors are very difficult to detect. A clear understanding of the underlying principles of coordination can help to avoid those mistakes in the early design phases.

III.3 Modelling coordination with games

Coordination systems can be modelled as a game [9]. In a coordination game, several agents follow predefined rules, use some resources and follow their own strategies to contribute to a common objective. The objective is only reached when the agents coordinate in the right way. We want to highlight the difference between games where one individual wins from coordination games that end when a common objective has been reached. In the latter, cooperation is required to avoid situations in which the game do not progress due to the blocking generated by non-cooperative players.

In this type of games, agents (players) have to deal with some fundamental issues of coordination systems such as deadlocks, racing, feedback loops and choice-making problems. By using games, players (learners) can discover the meaning of those issues without the use of computers at all.

IV CASE STUDIES: THE MUTUAL EXCLUSION PROBLEM

Mutual exclusion is one of the fundamental issues in parallel computing. It can be defined as the problem of guaranteeing the exclusive access to a single shared resource when there are several competing agents [18]. When mutual exclusion is not guaranteed, some erroneous behaviour such as deadlocks, livelocks and races can appear. Deadlocks are situations in which a set of agents are involved in a circular wait. Livelocks are a similar problem in which the agents are entangled in a circular wait but they are changing their state constantly without advancing. Races appear when the solution of a computation depends on the order of parallel computations.

In this section, we present two games in which deadlocks and mutual exclusion problems can arise. Players are encouraged to find, intuitively, appropriate solutions to avoid them.

IV.1 Deadlocks and the “orange game”

We use the “orange game”, from the Computer Science Unplugged Project [2], to illustrate the problem of deadlocks and how cooperative strategies solve them. In this game, a group of players are seated in a circle. Resources consist of two oranges per player labeled with their initials (except for one player that has only one orange). The aim is for each person to end up holding the oranges labeled with their own letter. The initial state is a random distribution of the oranges between the players. There are only two rules: only one orange may be held in a hand, and an orange can only be passed to an empty hand of an immediate neighbour in the circle. At some point in time, some

players may attempt to “win” the game leaving others in a deadlock situation. In order to avoid the deadlock those players need to disarm his move and continue playing.

After a round, participants can discuss and analyze the impact of strategy, rules and resources. They can develop and share their strategies, find new ones, or even propose to modify the rules and add or remove some resources. This include for example changing the organization of agents (topology) or restricting the communication between them. Based on that discussion, participants will be asked to derive a general definition of the problem that the game is illustrating and a general procedure for solving (or avoiding) it.

IV.2 Mutual exclusion and the “dining philosophers” problem

Mutual exclusion (and deadlocks, livelocks and starvation problems) can be illustrated with a variant of the “dining philosophers” problem, which is a classic synchronization problem in parallel computing [10]. In this game, a group of agents (philosophers) are seated on a table and a fork is placed between each pair of them. The “life cycle” of philosopher consists of thinking, becoming hungry, eating, and thinking again. In order to start eating, the hungry philosopher needs to get the two adjacent forks. The problem is to design a strategy for each of the philosophers that satisfy some defined property such as “starvation-freedom”, which means that each hungry philosopher eventually gets to eat [18].

A game to reproduce the dining philosophers problem is set as follows. Players sit on a circular way and in the center there will be a bin with colored balls (or oranges with labeled initials). A stick is placed between each pair of players. The objective of the game is to allow all the players to collect their own balls (or “eat” their own oranges).

The game start with a set of rules, and after a round, players can modify the rules to find different solutions. For example, communication between players can be restricted by not allowing them to talk or even to see each other.

After playing several rounds of the game, participants can discuss their strategies and their efficiency. A discussion can follow to analyze different ways in which deadlock situations can be avoided and the general applicability of these solutions.

V CONCLUSION

In this paper we have presented a methodology for learning parallel computing using games. We suggest that learning will be easier when a set of general principles are

presented initially to the learners and when all the subsequent learning activities are designed around that principles.

We argue that basic principles of parallel computing can be learned in a conceptual and intuitive way, independent of the technologies that implement them. We include parallel computing within the general coordination problem and model coordination as a cooperative game. In this way, abstract concepts, such as mutual exclusion and deadlocks, can be learned intuitively by playing games and before any technological or application details have been presented.

We are working now on a collection of games that can illustrate other issues in parallel computing and similar problems in other computing areas. We are preparing the materials to include some of these exercises in real courses at the Computer Engineering School from the Universitat Politècnica de Catalunya.

V ACKNOWLEDGEMENTS

We want to acknowledge all the ALE-2009 organizing team and specially to Pau Bofill for their support and motivation that to make this paper.

REFERENCES

[1] Krste Asanovic, et al The landscape of parallel computing research: A view from berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Dec 2006.

[2] Tim Bell, Ian H. Witten, and Mike Fellows. *Computer Science Unplugged: An enrichment and extension programme for primary-aged children*. Computer Science

Unplugged, 2005.

[3] G. Blake, R.G. Dreslinski, and T. Mudge. A survey of multicore processors. *IEEE Signal Processing Magazine*, 26(6):26–37, november 2009.

[4] S. Borkar. Design challenges of technology scaling. *IEEE Micro*, 19(4):23 –29, jul. 1999.

[5] Shekhar Borkar. Thousand core chips: a technology perspective. *In DAC '07: Proceedings of the 44th annual conference on Design automation*, pages 746–749. ACM, 2007.

[6] Peter J. Denning. Great principles of computing. *Commun. ACM*, 46(11):15–20, 2003.

[7] Peter J. Denning. Computing is a natural science. *Commun. ACM*, 50:13–18, July 2007.

[8] Peter J. Denning. The profession of it: Beyond computational thinking. *Commun. ACM*, 52:28–30, June 2009.

[9] Peter J. Denning and Craig Martell. Coordination: out of many. Technical report, Department of Computer Science, George Mason University, 2007.

[10] E. W. Dijkstra. Hierarchical ordering of sequential processes. *Acta Informatica*, 1:115–138, 1971.

[11] Alan D. Fekete. Teaching about threading: where and what? *SIGACT News*, 40:51–57, February 2009.

[12] Stephen J. Hartley. Alfonse, you have a message. *In Proceedings of the thirty-first SIGCSE technical symposium on Computer science education*, pages 60–64, 2000.

[13] David Kirk and Wen mei Hwu. *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann, 2010.

[14] Leslie Lamport. Teaching concurrency. *SIGACT News*, 40:58–62, February 2009.

[15] Gary Lewandowski, Dennis J. Bouvier, Tzu-Yi Chen, Robert McCartney, Kate Sanders, Beth Simon, and Tammy VanDeGrift. Commonsense understanding of concurrency: computing students and concert tickets. *Commun. ACM*, 53:60–70, July 2010.

[16] Gordon Moore. Cramming more components onto integrated circuits. *Electronics Magazine*, 1965.

[17] Herb Sutter and James Larus. Software and the Concurrency Revolution. *Queue*, 3(7):54–62, 2005.

[18] Gadi Taubenfeld. *Synchronization Algorithms and Concurrent Programming*. Prentice Hall, 2006.