An Integrated Course on Parallel and Distributed Processing[*]

José C. Cunha **João Lourenço**

{*jcc, jml*}*@di.fct.unl.pt*

Departamento de Informática
Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa
Portugal

Abstract

Most known teaching experiences focus on parallel computing courses only, but some teaching experiences on distributed computing courses have also been reported. In this paper we describe a course on Parallel and Distributed Processing that is taught at undergraduate level in the Computer Science degree of our University.

This course presents an integrated approach concerning concurrency, parallelism, and distribution issues. It's a breadth-first course addressing a wide spectrum of abstractions: the theoretical component focus on the fundamental abstractions to model concurrent systems, including process cooperation schemes, concurrent programming models, data and control distribution, concurrency control and recovery in transactional systems, and parallel processing models; the practical component illustrates the design and implementation issues involved in selected topics such as a data and control distribution problem, a distributed transaction-based support system and a parallel algorithm.

We also discuss how this approach has been contributing to prepare the student to further actions regarding research and development of concurrent, distributed, or parallel systems.

Introduction

Most known teaching experiences focus on parallel computing courses only [9, 10, 12, 18, 20]. They report on several approaches concerning the hardware and software support.

Some teaching experiences on distributed computing courses have also been reported [7, 16, 19] encompassing network computing or distributed algorithm design. A proposal for an integrated approach to teach concurrency issues in operating systems, database systems, and distributed systems can be found in [2] but it doesn't discuss parallel computing, and its focus is more on the operating system level.

In this paper we describe a course on Parallel and Distributed Processing that is taught at undergraduate level in our Computer Science (CS) degree, and presents an integrated approach concerning concurrency, parallelism, and distribution issues. It is a breadth-first course addressing a wide spectrum of abstractions, but it assumes a previous exposure to classical concurrency issues at the operating system level and basic knowledge of computer networks. It is attended by students in their final (fifth) year aiming to prepare them for further actions at the university or the industry requiring a deep understanding of the complex behavior of real concurrent, distributed and parallel systems. Although most classical CS courses cover individual topics such as concurrent programming, network programming, transaction-based processing, or parallel algorithm design, they usually consider each topic in a very narrow scope. From a conceptual point of view, many common issues appear among the above topics which must be clearly identified and put into a more broad scope, such that a better understanding of real applications and systems can be achieved. We think this integrated view can be presented in a final course of a CS degree.

The theoretical component focus on the fundamental abstractions to model concurrent systems, including process cooperation schemes, concurrent programming models, data and control distribution, concurrency control and recovery in transactional systems, and parallel processing models. The practical component aims at illustrating the design and implementation issues involved in a few selected and significant topics: (i) a data and control distribution problem, which is typically a distributed algorithm involving distributed process synchronization based on logical clocks; (ii) a distributed transaction-based support system, typically involving both concurrency control and recovery techniques; (iii) a parallel algorithm and its application, typically involving its implementation and performance evaluation.

A distinctive characteristic of our approach is the emphasis on a close integration of the following aspects: (i) the theoretical issues involving concurrency, distribution, and parallelism which are presented in an unified way; (ii) a tight

[*]Accepted for publication in "*The 29th SIGCSE Technical Symposium on Computer Science Education*".

interaction between theoretical abstractions and their practical implementation; and (iii) the effective use of an unified platform—the PVM [3] system—to support the implementation.

In the following three sections we discuss each of the above aspects. We conclude with a discussion on the characteristics of this approach, and describe further experiences.

Abstractions for Concurrency, Distribution, and Parallelism

Our goal is twofold: (i) to identify the main fundamental abstractions for concurrency, communication, and synchronization; (ii) to introduce distribution and parallelism concepts and techniques.

The theoretical lectures address the following topics:

1. *A discussion on the characteristics of concurrent, distributed, and parallel systems.* The goal is to identify the distinctive aspects of so-called concurrent, distributed and parallel systems as well as the underlying common concepts. This is made by discussing the whole spectrum of computation layers: classes of applications, formal models, programming languages, operating systems, and hardware architectures. A breadth-first approach is highly desirable because it gives a global view of the issues, and prepares the way to further exploration of specific topics. The student will get a clear understanding of the logical problem specification, and the approaches that support the implementation of high-level abstractions.

2. *An overview of concurrent, distributed, and parallel programming models.* Known approaches for the specification of concurrency and communication are discussed. Processes, threads, objects, shared and distributed name spaces, and associated communication models are presented and illustrated with examples, drawn from representative languages and models.

3. *A discussion on the fundamental aspects of data and control distribution.* Models of data and control distribution, partitioning and replication are discussed. Synchronization of distributed processes and event ordering management (logical and vector clocks, and causal precedence) is deeply studied. Several classes of distributed algorithms are discussed, ranging from mutual exclusion, election, deadlock handling, termination detection, to high-level protocols for atomic and causal message broadcasting, and coherent management of replicated data.

4. *A discussion of concurrency in transaction-based systems.* Transactions in database and object-oriented settings are discussed. Both concurrency control and fault-handling and recovery techniques are discussed, using centralized and distributed approaches.

5. *An overview of parallel processing systems.* A discussion of several classes of parallel applications and algorithms, the identification of significant parallel programming models, and the implementation issues concerning the operating system and the parallel architecture.

Integrating Theory and Practice

Practical support for teaching Parallel and Distributed Processing can be entirely based on the high-level programming language level. Languages such as Occam [15], or Linda-based languages [4, 8] can be used with the advantage of clearly illustrating the adequacy (or lack of it) of a specific programming paradigm to solve distinct problems.

Alternatively, a more generic (and flexible) approach consists in using a language such as SR [1], offering the possibility of exploiting several distinct concurrency and communication semantics in an unified framework.

However, in order to support such a wide spectrum of concepts as discussed in the theoretical lectures, we need a very flexible programming platform, preferably at an intermediate level, offering an adequate compromise between some degree of transparency, and some level of control of system-level abstractions. This requirement results from the need to illustrate typical distributed system concepts, typical transaction-based concepts, and typical parallel processing concepts, in a single framework.

Among the existing platforms, we find the PVM system [3] to be the most suitable to meet the above goal. Practical sessions are organized according to three main small projects, each closely corresponding to one major topic that is developed by the theoretical lectures. In the following we illustrate three examples that have been actually performed in our course:

1. *A project on distributed algorithm design and implementation.* Typically, a classical distributed algorithm is selected among the ones discussed on the lectures, and the students must devise a logical distributed architecture that matches the algorithm. Issues such as process decomposition and cooperation, synchronization and event ordering with no global clock, cost in terms of message traffic, and fault-handling naturally arise, and are handled in the setting of this specific problem.

2. *A project on transaction-based processing.* Typically, a simplified transaction-based system is designed on top of an existing UNIX file system, including a concurrency control method, based on locking, and a deadlock handling strategy. Centralized and distributed solutions are analyzed.

3. *A project on parallel processing.* Typically, a parallel algorithm is selected, and several alternative designs concerning granularity and process and data distribution are discussed. Performance measurement is performed in order to evaluate the actual implementation.

Evaluation of the PVM System as a Tool to Support Practical Laboratories

The PVM programming interface is simple and easy to use. The students quickly master the fundamentals of process control and communication mechanisms provided by the system. Practical PVM-based laboratories are easily installed on any UNIX-based local network.

For each of the above projects, an incremental development approach is followed: first the student designs and implements some basic mechanisms and associated primitives that are needed to support the specific high-level abstractions required by each problem; at a second step, the student actually uses such primitives to implement a solution to the problem; finally, an evaluation of the selected solution is made, concerning both its functionality and its performance. Alternative solutions may then be proposed but usually there is no time to actually implement them.

As an example, consider the project on distributed algorithm design. First the students implement an extended version of the PVM communication primitives such that logical timestamps and logical clock management is supported. Then, using such primitives, the students implement the distributed algorithm. A similar approach is followed concerning the transaction-based project. First support for lock management is implemented, and then deadlock handling is added. Concerning the project on parallel algorithm design, first a master-slave scheme is implemented, and then it is actually used to solve the specific problem.

From our experience, we claim this is a very fruitful approach because it allows a clear identification of the required abstraction mechanisms, at distinct levels, their implementation on top of a specific platform (PVM), and their use to solve the proposed problem. This allows the student to get a very realistic feeling about the theory, the design abstractions, and the implementation constraints.

Although many courses use specific parallel machines, greater flexibility can be achieved by having a heterogeneous local computer network including some multiprocessor nodes. We use a UNIX LAN with access to two multicomputers, each with sixteen Transputer-based nodes. Concerning the software environment, there is an advantage of using PVM and the C language for all the programming projects in such a short course because it provides a more uniform environment instead of a variety of languages and environments. A more advanced course can explore multiple parallel programming models.

Discussion

This course has been lectured for several years and we have some evidence that it really helps the student to get a relatively deep understanding of the theoretical and practical issues involved in the covered topics.

In a fifteen-week term, the course has 30 hours of theoretical lectures, and 45 hours of practical sessions. The average students are typically able to complete the practical session projects with only some extra effort beyond class time. An average number of 40 students have attended the course each year. They are divided in two classes during the laboratory sessions. They are in their fifth year of the Informatics Engineering degree and have a classical background in computer architecture and organization, operating systems, computer networks, and programming and software engineering. So they are well prepared to concentrate on design and implementation issues from an integrated, system-level point of view.

As there is not enough time to explore all course matters with depth, both theoretically and practically, the course has a broad breadth-first scope aiming at the following main goals:

1. To prepare the student for an integrated view of concurrent, parallel and distributed systems technology as found in real systems and applications. Real systems and applications encompass multiple dimensions related to the course topics, e.g. they exhibit different levels of concurrency, with several possible forms of parallelism, which are typically integrated into a distributed and heterogeneous environment.

2. To enable further advanced studies which complement this course: in the final course project during the second term of the fifth year of the engineering degree, or at master level.

As there are no support texts fully encompassing the course topics we are using multiple sources [5, 13, 14, 17] and some significant papers concerning algorithm design. An integrated view of all these elements is given to the students in a set of lecture notes [6].

In [18] a prediction is made that in the long-term specific parallel computing courses may well tend to disappear, and be subsumed within other computer courses ranging from algorithms, to compilers, operating systems, and software engineering. Although we tend to agree with this view, this is not the case for an integrated approach such as the one we propose.

Our experience with the course even shows a great need to increase the focus on the integrated view. This concerns for example the presentation of distinct abstractions into more unified models, e.g. shared-memory, distributed-memory, and distributed-shared memory; process-based, object-based and thread-based; centralized and distributed data and control; total and partial orderings. It also concerns the illustration of specific case studies where the above multiple dimensions can be found in real applications and systems.

Further Experiences

In the past five years, about 20 final student projects were successfully completed. These projects took place during the second fifteen-week term following our course, and involved a large diversity of topics such as distributed debugging, parallel genetic algorithms, parallel volume visualization algorithms, parallel and distributed Prolog, and runtime support systems for the Transputer machines. Some of these students have made our M.Sc. course in Computer Science where they attended a more advanced course on Parallel and Distributed Processing, and completed dissertation projects under related topics. Through these actions we found that some specific topics which were lectured in the course can be more deeply explored by the students.

In 1996 we have started an exciting experience by involving our undergraduate students in a large project for designing and implementing a parallel and distributed computational steering environment for genetic algorithms. The students were divided into groups, each responsible for the design and implementation of a component in the environment, under our close supervision:

1. Parallel genetic algorithms: four students were responsible for the implementation of a PVM-based prototype for the parallel execution of genetic algorithms, including a visualization component that graphically displays the evolution of the parallel computation. We also selected two test applications involving optimization problems in the area of the environmental sciences, as case studies to evaluate the prototype.

2. Parallel and distributed debugging: four students were in charge of developing a distributed debugging architecture for C programs in the PVM environment. This is being used as the steering component of the whole environment, in order to dynamically control the parameters of the genetic algorithm prototype.

From the pedagogical point of view this project has been showing us several interesting aspects:

1. The students were really involved into a real collaborative effort which forced them to carefully analyze the interfaces between components, and to face the corresponding design and implementation difficulties. This also puts particular requirements upon the developed documentation describing each component.

2. Intermediate design or implementation errors in some components were made evident when trying to integrate them into the system.

3. The students were highly motivated and stimulated by this environment which is expected to be used by non-computer scientists.

All of these students had previously attended the integrated course on parallel and distributed processing.

Besides our undergraduate degree we have also applied this approach in three intensive courses that we have lectured within the scope of an European Tempus project [11]. In these very short courses (two-weeks) both the theoretical and the practical sessions have focused on a single specific topic (distributed algorithm design and implementation). Again, we have used the PVM system for the practical sessions, with success.

Conclusions

We discussed our experience with a course based on an integrated view to present multiple dimensions found in so-called concurrent, parallel and distributed systems. This will enable the student to understand both conceptual and practical design and implementation issues that are found in real applications and systems, where multiple heterogeneous and autonomous hardware and software components establish complex interactions affecting both the performance and correctness of the whole system.

We have shown how this course has allowed the students to pursue further successful actions, at undergraduate and master levels, towards a more deep development of parallel and distributed processing topics within the university environment. We plan to pay particular attention to the proposal of more industry-oriented final student projects.

Acknowledgements

Thanks are due to all the students that have participated in these activities.

This work was partialy supported by the EC TEMPUS Project S_JEP 07989-94.

References

[1] G. R. Andrews. The Distributed Programming Language SR — Mechanisms, Design, and Implementation. *Software: Practice and Experience*, 12(8), 1982.

[2] J. Bacon. *An Integrated Approach to Operating Systems, Database, and Distributed Systems*. Addison-Wesley, 1993.

[3] A. Beguelin, J. J. Dongarra, G. A. Geist, R. Manchek, and V. S. Sunderam. A User's Guide to PVM Parallel Virtual Machine. Technical Report ORNL/TM-118266, Oak Ridge National Laboratory, 1991.

[4] N. Carriero and D. Gelernter. Linda in context. *Communications of the ACM*, 32(4), April 1989.

[5] K. M. Chandy and J. Misra. *Parallel Program Design*. Addison-Wesley, 1988.

[6] J.C. Cunha. Distributed systems. Lecture notes, Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, 1991.

[7] E. Dillon, C.G. Santos, and J. Guyard. Teaching an Engineering Approach for Network Computing. In *Proceedings of the 27th SIGCSE Technical Symposium on Computer Science Education*, volume 28, pages 229–232, Philadelphia, Pennslvania, March 1996. ACM.

[8] B. S. Elenbogen. Parallel and Distributed Algorithms:Laboratory Assignments in Joyce/Linda. In *Proceedings of the 27th SIGCSE Technical Symposium on Computer Science Education*, volume 28, pages 14–18, Philadelphia, Pennslvania, March 1996. ACM.

[9] L. Jim and L. Yang. A Laboratory for Teaching Parallel Computing on Parallel Structures. In *Proceedings of the 26th SIGCSE Technical Symposium on Computer Science Education*, pages 71–75, Nashville, Tennessee, March 1995. ACM.

[10] D. J. John. NSF Supported Projects: Parallel Computation as an Integrated Component in the Undergraduate Curriculum in Computer Science. In *Proceedings of the 25th SIGCSE Technical Symposium on Computer Science Education*, volume 26, pages 357–361. ACM, 1994.

[11] J. Kwiatkowski, M. Andruszkiewicz, E. Luque, T. Margalef, J. C. Cunha, J. Lourenço, H. Krawczyk, and S. Szejko. Teaching Parallel Processing: Development of Curriculum and Software Tools. In *Proceedings of International Conference on Integrating Technology into Computer Science Education*, volume 28, Barcelona, Spain, June 1996. ACM.

[12] E. Luque, J. Sorribes, R. Suppi, E. Cesar, J. L. Falguera, and M. Serrano. Parallel Systems Development in Education: a Guided Method. In *Proceedings of International Conference on Integrating Technology into Computer Science Education*, volume 28, Barcelona, Spain, June 1996. ACM.

[13] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[14] N. Lynch, M. Merritt, W. Weihl, and A. Fekete. *Atomic Transactions*. Morgan Kaufmann, 1994.

[15] D. May. OCCAM. *SIGPLAN Notices*, 18(4), April 1993.

[16] N. Plouzeau and M. Raynal. Elements for a Course on the Design of Distributed Algorithms. *SIGCSE Bulletin*, 24(2), 1992.

[17] M. Raynal. *Distributed Algorithms and Protocols*. John Wiley & Sons, 1988.

[18] N. C. Schaller and A. T. Kitchen. Experiences in Teaching Parallel Computing: Five Years Later. *SIGCSE Bulletin*, 27(3):15–20, September 1995.

[19] C. Stewart. Distributed Systems in the Undergraduate Curriculum. *SIGCSE Bulletin*, 26(4):17–20, December 1994.

[20] W. E. Toll. Decision Points in the Introduction of Parallel Processing into the Undergraduate Curriculum. In *Proceedings of the 26th SIGCSE Technical Symposium on Computer Science Education*, volume 27, pages 136–140, Nashville, Tennessee, March 1995. ACM.