An Undergraduate Course in Concurrent Programming Using Ada

Kwok-bun Yue
Division of Computing and Mathematics
University of Houston - Clear Lake
2700 Bay Area Boulevard
Houston, TX 77058-1098
E-Mail: Yue@cl.uh.edu

Abstract

This paper describes a senior level course in concurrent programming using Ada. Unlike other similar courses in the subject area, it is not part of an operating systems course, nor is it tied to a particular hardware architecture. The course is software oriented and it discusses in depth a concurrent programming language, Ada, so that students are able to actually develop effective concurrent programs to solve problems in a wide range of applications. Ada is selected because of its popularity, superb portability, numerous hardware platforms, and rich concurrent constructs. Classical issues in concurrent programming are presented in the context of Ada. General issues in designing concurrent programming languages are elaborated using Ada, together with other concurrent programming languages such as CSP, Occam, and Linda. Finally, general principles of designing parallel programs are also discussed. Therefore, the course provides both the depth in a concurrent programming language for program development and the breadth in concurrent programming theory for insight. Using Ada throughout the course strengthens student's expertise in Ada and provides an useful reference point for understanding concurrent programming theory. The course is heavily based on handouts, examples, homework and programming assignments. A rich set of instructional materials are available from the author.

1. Introduction

Concurrent programming has rapidly become an integral part of computer science education. As Carriero and Gelernter put it, "parallelism will become, in the not too distant future, an essential part of every programmer's repertoire." [3]. Frequently, concurrent programming is either offered in the graduate level, or is covered as a part of other undergraduate courses, usually the operating systems course. For example, Sanders and Hartman described the integration of concurrent programming in the Fortran and programming languages courses [16]. There is much discussion of teaching concurrent programming within an operation systems course [12, 14, 18]. There are also many parallel programming courses that are hardware based, with the course contents closely tied to one or many particular hardware architectures [15].

While these efforts are productive, we believe the growing importance and maturity of the area justify an independent software oriented concurrent programming course at the senior level. By separating it from the operating systems course, both the depth and the breadth of the subject can be reached. This also eases the pressure of the ever growing contents of the undergraduate operating systems course and allows it a better coverage of other important topics. This paper discusses the framework and organization of such a course offered at the University of Houston - Clear Lake (UHCL).

2. Course Objectives

There are two major applications of the concurrent computation model. The first application is to exploit hardware parallelism to solve problems more quickly by mapping concurrent processes to underlying parallel hardware architectures for potential speedup. The term *parallel programming* is generally used when physical hardware parallelism exists. In this respect, the goal is to speed up problem solving and many topics may be discussed, for example, parallel models of hardware architectures (such as message passing, shared memory and data parallel), design of parallel algorithms, complexity analysis of parallel algorithms, formal methods in developing parallel programs and parallel processing.

The concurrent computation model can also be effectively used in modelling and constructing effective solutions in areas such as embedded systems and real-time applications, where it is more suitable than the traditional sequential computation model. The main

concern here is the appropriate design of solutions, and not the potential speedup. In fact, many embedded systems and real-time applications are implemented on uni-processors. In this respect, topics such as concurrent computation models, concurrent programming languages, timing, priorities and scheduling, control and synchronization theory, etc, may be discussed.

It is obviously not possible to include all these topics in a single course. Our first decision is to make the course software oriented. After their graduation, it is more likely for students to develop concurrent programs in a concurrent language than to deal with a particular parallel hardware architecture directly. Hardware issues in parallel programming are thus left to other available courses at UHCL, such as the graduate courses in high performance architectures and parallel processing.

Several objectives of the course are established to define the course contents. Firstly, the course should discuss in depth a selected concurrent programming language so that students are able to actually develop concurrent solutions in this language for a wide range of applications. This is in contrast to courses that solely concentrate on the general theory of concurrent computational models without an in-depth coverage of a concurrent language. We feel that at the undergraduate level, actual development of concurrent programs is very important in helping students to understand the underlying concepts. A deep understanding of a concurrent programming language will expose many implementation issues common in developing concurrent solutions, improve the student confidence in their skills and provide insight in general issues of concurrent programming. By concentrating at the concurrent programming language level, we avoid being tied to the intricacy of a particular hardware architecture, which may not be readily transferred to other hardware architectures. It is more advantageous to have expertise in a language that can be mapped to different kinds of hardware architectures.

Secondly, students should understand enough classical issues in concurrent programming to allow them to develop concurrent solutions. Thirdly, students should have a good understanding of concurrent programming languages in general. Students may not be working with the selected language after their graduation and may need to learn another language quickly. A broad knowledge of concurrent programming languages is essential in developing transferable insight in concurrent programming. Such a grasp of the underlying theory is especially important because of the rapid technological advances in computer science. Furthermore, students will be more aware of the strengths and shortcomings of the selected language and thus be more capable of developing effective solutions in the language.

Fourthly, students should have a good idea of general design principles of concurrent programs, independent of the language used. That should cover the basic mechanism for composing parallel programs. Ideally, a formal method approach, such as by Chandy and Taylor [5], with a good treatment of complexity analysis, is desirable, if time permits.

3. Course Contents

Ada is selected as the language of choice for the course. UHCL is located next to the NASA Johnson Space Center at Houston with a large Ada community. There is a SIGAda in Clear Lake. Ada is a required core course for every undergraduate program in computer science. Beyond the local rationale, there are good general reasons for selecting Ada. Many have reported using other concurrent languages in a concurrent programming course: Pascal-FC [6], Concurrent Distributed Pascal [10], Joyce-Linda [13], CSP [15] and custom designed language environments [11]. Although they have many merits in themselves, none can match Ada in terms of popularity, practicality, portability and availability in a wide range of platforms. Portability is especially convenient for students since they can develop their programs on many platforms. In fact, Feldman has demonstrated that sophisticated Ada programs can be written using a number of packages and tasks to be entirely portable [8].

Besides, Ada is a modern programming language with rich constructs and was designed with software engineering principles as goals, both important for an instructional language for computer science courses. Although there are some well reported problems in the rendezvous model in Ada, Ada's tasking is powerful enough to solve a wide range of applications effectively. Furthermore, the new Ada-94 standard has solved many of these problems as well as further strengthened Ada expressiveness.

The course is divided into three parts. Since all students have already taken an introductory Ada programming course, it is not necessary to review Ada sequential programming features. The first half of the course contains a very detailed discussion on all concurrent constructs in Ada, in parallel with traditional issues in concurrent programming. Classical problems such as buffers, mutual exclusion, readers and writers,

and producers and consumers are also discussed with Ada solutions of various sophistication. A discussion of the relative merits of the Ada rendezvous model ends the first half of the course.

There are many good textbooks for this part (for example, [2, 3, 9, 17]). The book by Burns [3] is probably the most authoritative and complete treatment of concurrent programming in Ada. However, it is relatively hard for undergraduate students to read and contains no exercises. Although we have selected the book by Barnes [2] because it is also the textbook for the prerequisite Ada programming course, any of the aforementioned books should be satisfactory.

In the remaining half of the semester, issues in designing concurrent programming languages are first elaborated, for examples, choices in asynchronous or synchronous communication, process identification, creation and termination, scheduling controls and data flows. It also includes a discussion of classical synchronization constructs such as semaphores and monitors, as well as several other concurrent languages, such as CSP, Linda and Occam. Linda is especially emphasized as it is based on an asynchronous communication model, as opposed to the synchronous communication model in Ada. New features in Ada-94 are also covered.

Many examples are developed both in the syntax of these concurrent languages and that of Ada. Examples with Ada syntax allow students to focus on high level comparison and contrast with Ada, illustrating the choices made in the designs of the languages. Examples in the syntax of other concurrent languages allow the students to have a better understanding of these languages. This is the approach of Ben-Ari and in fact his well-known book is used as the textbook for this part [1]. The paper by Elrad and Nohl is also useful for the discussion of scheduling controls [7].

The last part of the course is the discussion of the concurrent program design principles as the book by Carriero and Gelernter [4]: the three concurrent programming paradigms, specialist parallelism, agenda parallelism and result parallelism, and their supporting data structures. The discussion in [4] is excellent, but it is relatively difficult for the average undergraduate students to understand. Besides, the language used is C-Linda, not Ada. Hence, handouts and examples in "Ada-Linda" are developed to illustrate the concepts.

4. Assignments

The success of the course depends largely on whether or not the students can assimilate and practice the knowledge they learnt in class, which is relatively new to most of them. Therefore, weekly homework assignments are assigned to the students. These assignments usually contain four to eight questions that require a considerable amount of reading and thinking. Suggested solutions are immediately distributed once the assignments are collected, enabling the students to learn by comparison and contrast.

Since one of our major objectives is to train students to be proficient in actually developing concurrent programs, programming assignments are integral to the course. In a typical semester, there are five to six programming assignments, most involving Ada. The first assignment, such as a concurrent version of quicksort, introduces students to the Ada rendezvous model. The second assignment is usually a simple hypothetical embedded system, such as a vending machine, that requires the students to understand task activation and termination, to use the select statement and to be aware of issues in integration to the embedded system. The third assignment usually deals with dynamic task types and the generation of worker tasks. A typical example is the polynomial evaluation of P(X) for many X values. An optional assignment is usually on resource management, simulation or generic. Issues such as mutual exclusion, timing and scheduling are essential in these assignments. Together, the first four assignments help the students to gain expertise in all major Ada concurrent programming features and to understand classical issues in writing concurrent programs.

The fourth assignment is usually solving a relatively easy problem, such as matrix multiplication in some concurrent programming languages, such as Ada and Linda. The assignment enhances student understanding in the differences of these languages as well as general issues in the design of concurrent programming languages. The last assignment is usually solving a problem using different parallel programming design techniques.

Because of the heavy dose of homework and programming assignments, it is unreasonable for the students to develop the entire program solutions. Furthermore, many bolts and nuts of the solutions are not related to concurrent programming. Ada packages provide a convenient way to give the student a skeleton with which to start working with so that they can concentrate on the essential elements. For example, in the assignment of developing a concurrent quicksort procedure, packages are provided for the basic input and output operations, necessary type definitions as well as

the driver for testing the quicksort procedure. The only Ada code the student needs to implement is the concurrent quicksort procedure itself, which can be fitted into two pages, documentation included. Like homework assignments, suggested solutions to programming assignments are distributed.

5. Conclusions

This paper has briefly described a software oriented senior level course in concurrent programming that provides both the depth in Ada for students to solve a wide range of applications, and the breadth in various important topics in concurrent programming theory. The course is participation oriented in the sense that students need to work through heavy classwork exercises, homework assignments and programming assignments. For those interesting in offering a similar course, a collection of instructional materials, including the course syllabus, twelve instructional modules with notes, 19 homework assignments with solutions, ten programming assignments with solutions, examinations and solutions, and example program code, are available from the author. In the future, We plan to gradually extend the course to base on Ada-94, the new Ada standard.

Acknowledgement

This work is partially supported by the Defense Advanced Research Projects Agency via Phillips Laboratory (PL), Air Force Material Command, United States Air Force under SFRC# F29601-93-K-0127.

References

[1] Ben-Ari, M, *Principles of concurrent and distributed programming*, Prentice-Hall, 1990.

[2] Barnes, J., *Programming in Ada*, Addison-Wesley, 1991.

[3] Burns, A., *Concurrent programming in Ada*, Cambridge University Press, 1985.

[4] Carriero, N. & Gelernter, D., *How to write parallel programs: a first course*, MIT Press, 1990.

[5] Chandy, K. & Taylor, S., *An introduction to parallel programming*, Jones and Bartlett, 1992.

[6] Davies, G., Teaching concurrent programming with Pascal-FC, *SIGCSE Bulletin*, 22 (2), (1990), 38-41.

[7] Elrad, T. & Nohl, D., The analysis and comparison of scheduling controls in concurrent languages through classification, *Proceedings of the 23rd Technical Symposium on Computer Science Education*, (1992), 89-93.

[8] Feldman, M., The portable dining philosophers, *Proceedings of the 23rd Technical Symposium on Computer Science Education*, (1992), 276-280.

[9] Gehani, N., *Ada concurrent programming*, Prentice-Hall, 1984.

[10] Higginbotham, C. & Morelli, R., A system for teaching concurrent programming, *Proceedings of the 22nd Technical Symposium on Computer Science Education*, (1991), 309-316.

[11] Jipping, M., el at., Concurrent distributed Pascal: a hand-ons introduction to parallelism, *Proceedings of the 21st Technical Symposium on Computer Science Education*, (1990), 94-99.

[12] Leach, R., An advanced operating systems project using concurrency, *Proceedings of the 21st Technical Symposium on Computer Science Education*, (1990), 39-44.

[13] McDonald, C., Teaching concurrency with Joyce and Linda, *Proceedings of the 23rd Technical Symposium on Computer Science Education*, (1992), 46-52.

[14] Mims, T. & Hoppe, A., Utilizing a transputer laboratory and Occam in an undergraduate operating systems course, *Proceedings of the 22nd Technical Symposium on Computer Science Education*, (1991), 317-323.

[15] Olszewki, J., CSP Laboratory, *Proceedings of the 24th Technical Symposium on Computer Science Education*, (1993), 91-95.

[16] Sanders, D. & Hartman, J., Getting Started with parallel programming, *Proceedings of the 21st Technical Symposium on Computer Science Education*, (1990), 86-88.

[17] Shumate, K., *Understanding concurrency in Ada*, McGraw-Hill, 1988.

[18] Silver, J., Concurrent programming in an upper level operating system course, *Proceedings of the 20th Technical Symposium on Computer Science Education*, (1989), 217-221.