Fall-12: Using the SALSA Programming Language to Introduce Concurrency in CS2

Travis Desell
Department of Computer Science
University of North Dakota
Grand Forks, ND, USA
Email: tdesell@cs.und.edu

***Abstract*—The SALSA programming language was used in a two week module to introduce concurrent and distributed programming concepts to computer science II students at the University of North Dakota. Computer science II is taught using Java, which made for an easy transition to using SALSA for concurrent programming, as it has a similar syntax and allows the use of Java objects. The actor model was introduced, along with the and necessary concepts of concurrency, asynchronous message passing and distributed memory. As an evaluation, a survey was given to students before and after the module, with the results of the survey highlight the fact that early computer science students do have a natural understanding of many concurrent and distributed programming concepts. Further, they can make their minds up quickly, gaining confidence much easier than they gain actual knowledge. It was also observed that the students learned concepts better by applying them in programming assignments than by being presented them in lecture. This work provides motivation for longer, applied learning modules on concurrent and distributed programming in future early programming courses.**

***Keywords*-parallel and distributed computing education; actor model; computer science II**

I. Motivation

The object and threads programming model used by Java and many other programming languages presents some difficulties for beginning computer science students, namely how to appropriately deal with deadlocks and how to synchronize data access (to prevent memory errors). Further, programs written using objects and threads can be difficult to debug, as it is challenging to determine which threads are causing these problems as there is no representation of this in the source code. These challenges are reflected in the NSF/TCPP Curriculum Initiative document, where concurrency defects and tools to detect them are suggested to be taught at the Data Structure and Algorithms level.

Objects and threads have a model where multiple tasks can be performed on the same object at the same time. As such, it is not particularly intuitive, as it is not representative of how things happen concurrently in the real world. Alternately, the actor model combines a thread of control, along with its state and behavior into a single unit of concurrency. Each actor maintains a mailbox of messages and will process them one after the other. As such, *the hypothesis is that using the actor programming model, which is more "true to life", may prove more intuitive for beginning students*. Further, the actor model uses distributed memory, which prevents concurrent memory accesses, and actors communicate via asynchronous message passing which makes it very difficult to program deadlocks – easing the problem of concurrent programming.

II. Methodology

Computer Science II has been traditionally taught in Java at UND, to introduce students to concepts of object oriented programming such as inheritance, polymorphism, generics and software design. Other topics of the course include, in addition to an overview of Java's libraries and syntax, an introduction to data structures such as stacks, queues and trees, as well as sorting algorithms such as quicksort and mergesort, which also introduce the students to recursion, and their analysis using big O notation.

Computer Science II is taught with a strong focus on applied learning by using extensive in-class programming assignments to reinforce the information taught in the lectures. One class a week is lecture style, while the remaining two classes are in the lab, along with an additional 2 hour evening programming lab. Because of this, students typically complete 2-3 programming assignments a week based on applying the information taught in the lecture. In general, while students have found the amount of programming to be challenging, they prefer the active learning and recognize the necessity of becoming stronger programmers to succeed in the computer science field.

III. Teaching Concurrency Using SALSA

With the structure of Computer Science II, there was time for one hour lecture and 4 lab hours each week. As opposed to most other weeks in the course, where students had 2-3 programming assignments to complete in the lab sessions, students were given a single programming assignment due to the fact that the concepts involved were more advanced and that they were using a new programming language.

The first week's lecture focused on introducing the syntactic differences between SALSA and Java, as well as the concepts of concurrency, asynchronous message passing and distributed memory. Students were given skeleton code for a parallel trapezoidal rule solver and asked to implement a version which could run any number of worker actors which could calculate slices of the trapezoidal rule, whose results would be summed up by the master actor.

The second week focused on introducing non-determinism and concurrency defects. In actor programming languages, most concurrency defects occur due to the fact that when messages are sent asynchronously, the order they are received in is not necessarily the order they were sent in, which is a direct example of non-determinism. Students were given a faulty dining philosophers SALSA program, which resulted in deadlock, and were asked to fix the program so that no philosopher starved.

The average grade over all students for the non-SALSA lab assignments was a 78.3; with the trapezoidal assignment having an average grade of 76.8 and the dining philosophers having an average grade of 63.2 – however for both assignments the grade distribution was very bimodal. Most students successfully completed the lab and received a 100, while the rest did not submit the lab and received a 0. As the dining philosophers assignment was the last of the semester, the lower grade may be due to the fact that students simply did not do the assignment due to studying for tests in other courses, rather than the fact they struggled with the material. This does bring out a common trend I have noticed, especially in lower level computer science courses. The abilities of the students are extremely bimodal. There are typically quite a few students who immediately grasp concepts and excel, while their remains a similar amount of students who struggle with beginning programming concepts for the entire semester. Some of this may also have been reflected in the grades.

IV. Survey Examination

A survey was given to students before and after the module on programming in SALSA. The Computer Science II course consisted of 34 students (22 were Computer Science majors). Of these students, 24 students took the pre-survey and 27 students took the post-survey, with 21 students taking both the pre- and post-survey.

The first part of the survey addressed student interest and experience in concurrent and distributed programming. Students reporting no interest went from 1 to 4, some interest from 17 to 13 and significant interest from 6 to 10. The important thing noticed is the importance of how these topics are taught to beginning computer science – *a two week module was sufficient to some students for making up their minds about further studying this subject*. Student major may have also had some effect on student interest, as 3 of the 5 students who reported no interest in the post survey were not computer science majors. For experience, students reporting no experience went from 18 to 7, some experience from 6 to 20 and significant from 0 to 0. Again, a short two week module had a significant effect on beginning computer science learners, as most students reported that the module gave the some experience in concurrent/distributed programming. The students were also reasonable about their self assessed knowledge, as none reported that they gained significant experience from the module.

The rest of the survey consisted of questions divided into five categories: concurrency, asynchrony vs. synchrony, determinism, distributed vs. shared memory, and concurrency defects. The categories were chosen due to their relatedness to programming in the SALSA programming language and what was taught during the two week module. Each category consisted of 3 to 4 (generally real-world) scenarios which addressed concepts for those categories. The rationale behind this approach was that many concurrent and distributed computing topics generally occur in the real world, so students may actually have some understanding of them, but not knowledge of the appropriate terminology. In general, it was found that students typically improved on the scenarios they had to actually apply (especially one on the dining philosophers), as opposed to those which were just discussed in lecture.

V. Discussion

Perhaps the most important information to take from this work is that early programming students do have a basic understanding of concurrent and distributed programming topics, without having any formal instruction, as most things in the real world operate concurrently. Second, and as a warning to those seeking to introduce these topics into their curriculum, increasing student confidence in a topic is significantly easier than increasing their actual knowledge – which can prove dangerous as students can easily become more sure of their incorrect knowledge.

Given this experience, there is reason to believe that with some changes it could be a very successful method for introducing concurrent and distributed programming to early programmers. The module only lasted for two weeks, and while this was sufficient to increase student confidence, the level of student knowledge was not improved on a similar level. However, topics in which students directly had to apply via programming assignments showed the largest improvement. This module also illustrated the bimodal abilities of early CS students. For some students I could have gone into much further depth, while others would need a full semester or longer to apply and grasp the concepts.

This work was positive in that it does show that students at this level can learn and understand these topics – however it should be noted that some students may still be at the level of code modification and testing, so they can simply get things to work without understanding the *why* of how they worked. However, it does mean that more time (especially applied time) must be devoted to these topics if students are going to understand them at this level. As such, future examination of this subject using SALSA and other distributed/concurrent programming languages/paradigms should involved more applied programming, to avoid the pitfall of students gaining confidence than actual knowledge.