Creative Learning of Concurrent Programming

Y. Lucet

November 27, 2012

Abstract

The project aims at achieving the highest level of learning (in Bloom's revised taxonomy) when teaching concurrent programming. To allow students to creatively solve concurrent programming challenges, a tool will be implemented to check automatically the validity of algorithms that run several codes in parallel. The tool will determine whether the algorithm is correct, and if not, it will provide feedback on the type of error and give an example of sequential code execution that gives the wrong output. Using that feedback, students will be able to understand their mistake, improve their solution, and propose another solution that fixes the error. The tool will be used by students in COSC 315 (Introduction to Operating System) and COSC 407 (Introduction to Parallel Programming), and by instructors in lower level courses mentioning concurrent programming challenges like COSC 222 (Data Structures).

1 Objective

All computers today contain multiple cores that allow the execution of several programs in parallel. In fact, running code in parallel may be the only way to take advantage of the latest processors computing capabilities [8]. However, computer science curricula still teach sequential programming i.e. almost all computer science courses assume the computer can only perform one operation at a time. For example, most programming courses at UBC O (COSC 111, 121, 222) introduce no (or very little) notion of concurrent programming. To remedy that situation and train students with parallel computing skills, I developed a parallel computing course (COSC 407), which was offered for the first time in January 2012. Following other instructors experience [7], I also introduced more parallel computing concepts in the operating system course (COSC 315) that I also taught in January 2012. Together these courses offer a solid introduction to parallel computing.

After teaching both course I realized one major difficulty of learning parallel programming is the lack of simple user-friendly tools to check ones solution. Sequential programming algorithms are much easier to verify: the computer can only execute one instruction at a time. Hence, simple sequential programs are easy to verify with pen and paper. Tools like debuggers offer a natural view of the execution. The situation is opposite with parallel programs. Even simple programs are not easy to verify when the computer can execute a number of instructions at each time step. One has to consider all the possibilities to validate the algorithm.

For example, consider two pieces of code called threads that have each two instructions and run in parallel. The code can be combined in six different ways to produce four different results. For code involving more than a couple of instructions and a couple of threads, the possibilities grow exponentially: 5 threads having 5 lines of code each produce $5^5 = 3,125$ possible combinations to validate. It becomes very quickly extremely difficult to determine whether a program is valid.

The current proposal aims to solve that difficulty by building a program that will validate simple code with specific rules. The objective is to facilitate student learning not to build a robust code validation tool. So it makes sense to cap the number of thread to 5 and the number of lines of code to 8 (that still leaves about $400,000$ combinations). That size is sufficient to understand and experience the core issues in concurrent programming.

Visualization of algorithms has attracted a lot of attention in the 1990s with results showing that careful integration of visualization tools is required to observe a positive effect [2]. In fact, feedback is needed to observe enhanced learning due to visualization [5]. More recently, a teaching tool was implemented to visualize concurrent programs [6]. Our focus here is different. We do not want another visualization tool but an automatic assessment tool that provides very quick feedback to the student.

Classical problems in concurrent programming can be found in The Little Book of Semaphore [3]. The book defines a simple syntax to solve popular concurrent programming problems, and include solutions. The challenge is to guide students toward the solution without showing them the solution.

Showing them the solution only teaches them how to reproduce and apply a technique (the lower levels of learning in Blooms taxonomy [1]). It is much more challenging for students to find the solution by themselves (highest level of learning in Blooms taxonomy). A very efficient strategy to achieve higher learning is to provide students with immediate feedback so they can take quick corrective action. That strategy was successfully deployed in COSC 222 (Data structure) for which numerous applets are freely available to check student answers [5]. We plan to create such a tool for parallel programming.

Once built, the tools will be introduced to the students first as part of directed assignments with a TA available in a tutorial-like format. Then the tool will be made available to study on a self-spaced mode. It will allow to challenge students and deepen their problem solving skills on intrinsically concurrent problems.

On the practical software development side, the project will build on the CONcurrent programming VIsualisation Tool (Convit) tool [4], which is freely available with an open source license. Convit already provides the visualization and debugger. It needs to be complemented with a robust exhaustive solution checker. Reusing the code will allow a quick implementation that is achievable by a strong undergraduate student in computer science using the requested funding.

2 Relation to Founding Principles

The project gives students the tool to solve challenging problems by themselves. Academic excellence is definitely one of its goals and as such it fits within the first founding principle of the Irving K. Barber School of Arts and Sciences. By achieving a higher level of learning, it advances excellence in teaching and learning, which is the third founding principle. It also provides a better teaching method and learning experience (fifth founding principle), and encourages creativity and innovation (sixth founding principle). Indeed, students will not be limited to the instructor solution, and will be convinced that their solution, while less elegant, still correctly solves the problem.

3 Immediate Benefits to Students

The tool will be deployed in COSC 315 Operating System, and in COSC 407 Parallel Computing to help student solve several assignments. Instructors in lower level courses (COSC 222 and even COSC 121 when mentioning thread safe functions) will also use the tool while discussing the challenge of writing concurrent code and the design decisions in existing libraries.

4 Milestones

The tool will be composed of a parser to recognize a limited set of simple commands, a master scheduler to enumerate all the possibilities, and one or more slave verifiers to evaluate each sequential combination of the code.

1. March 2013: Hire an undergraduate student in computer science for 16 weeks
2. May 2013: Read Convit code and understand Parser subroutine
3. June 2013: Add a Scheduler to generate every possible combination
4. July 2013: Add a Verifier to validate each combination
5. August 2013: Validate the tool and performance tuning
6. January 2014: Deployment of the tool in COSC 315
7. May 2014: Dissemination of the results

5 Evaluation/Assessment Plan and Dissemination

The tool will be deployed in COSC 407 and in COSC 315 in January 2014. Feedback will be collected and the improvements, bug fixes, and modifications will be performed in summer 2014. Then the tool will be made available to interested instructors at UBC. The success of the project will be evaluated based on the experienced of the instructor while using the tool in class (no student questionnaire will be used at this point), and the number of instructors who adopt the tool in their courses. Preliminary results will be presented at the 2014 Learning Conference and the 2014 Western Canadian Conference on Computing Education. Final results will be presented at the ACM SIGCSE[1] conference (March 2014), or ACM ITiSCE[2] conference (June/July 2014). The tool will be made available under an open source license.

An extension of the project will see a formal evaluation of the tool. It will necessitate an application to the research ethics board. Such quantitative evaluation is beyond the scope of the current proposal.

6 Budget

- $7,000 to hire an undergraduate student with programming experience for 16 weeks,
- $1,000 for a computer and the necessary software to program and test,
- $2,000 to present the tool at the 2014 Western Canadian Conference on Computing Education and at an international conference on computer science education.

References

[1] L. W. Anderson, D. R. Krathwohl, P. W. Airasian, K. A. Cruikshank, R. E. Mayer, P. R. Pintrich, J. Raths, and M. C. Wittrock, *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*, Pearson, 2000.

[2] M. D. Byrne, R. Catrambone, and J. T. Stasko, *Do algorithm animations aid learning?*, Tech. Rep. GIT-GVU-96-18, Georgia Institute of Technology, 1996.

[3] A. B. Downey, *The Little Book of Semaphore*, Green Tea Press, 2005.

[4] H.-M. Jarvinen, M. Tiusanen, and A. Virtanen, *Convit, a tool for learning concurrent programming*, in AACE E-Learn Conference, 2003.

---

[1] Association of Computing Machinery, Special Interest Group on Computer Science Education.

[2] 19th Annual Conference on Innovation and Technology in Computer Science Education

[5] A. Korhonen and L. Malmi, *Algorithm simulation with automatic assessment*, SIGCSE Bull., 32 (2000), pp. 160–163.

[6] V. Manickam and A. Aravind, *If a picture is worth a thousand words, what would an animation be worth?*, in Proceedings of the 16th Western Canadian Conference on Computing Education, WCCCE '11, New York, NY, USA, 2011, ACM, pp. 28–32.

[7] C.-K. Shene, *Multithreaded programming can strengthen an operating systems course*, Computer Science Education Journal, 12 (2002), pp. 275–299.

[8] H. Sutter, *The free lunch is over: A fundamental turn toward concurrency in software*, Dr. Dobb's Journal, 30 (2005).