Integrating Parallel and Distributed Computing Into Undergraduate Courses at All Levels

Steven Bogaerts, Kyle Burke
Department of Mathematics and Computer Science
Wittenberg University
Springfield, OH
{sbogaerts, kburke}@wittenberg.edu

Eric Stahlberg
CCR Bioinformatics Core
National Cancer Institute
Bethesda, MD
stahlbergea@mail.nih.gov

Abstract— **This document is a proposal for a 1/2 hour presentation at the First NSF/TCPP Workshop on Parallel and Distributed Computing Education (EduPar-11), on our ongoing efforts as part of our three-year National Science Foundation (NSF) grant with principal investigators Eric Stahlberg, Melissa Smith, and Steven Bogaerts. The grant facilitates a unique collaboration between a small private undergraduate institution, Wittenberg University, and a large research institution, Clemson University. This document summarizes the contents of our proposed presentation of this ongoing work, as applied to five courses: CS1, CS0, Computational Models and Methods, Programming Languages, and Algorithms.**

Keywords-education; early parallelism; CS1; CS0

I. INTRODUCTION

Our theme in integrating parallelism and distributed computing (PDC) concepts into our curriculum is to focus on *integration*, as opposed to *insertion*. Historically, any PDC in an undergraduate curriculum appeared in the context of operating systems, and perhaps algorithms or an upper-level elective. It is undeniable that PDC is of such significance today that it cannot be relegated solely to such courses.

We propose to share our methodology and experiences of integrating PDC into undergraduate courses at *all* levels as part of our three-year NSF grant. This includes upper-level courses like Algorithms and Programming Languages, mid-level courses like Computational Models and Methods, and even low-level courses like CS1 and CS0. Each of these courses carries ample opportunities to consider PDC in a manner that is both level-appropriate and in harmony with the traditional topics of the course.

II. CS1

Our PDC course module for CS1 was used for the first time in Fall 2010. There were two basic parts to the module: 1) background material, and 2) the use of PDC as a *medium* for the explanation of typical CS1 concepts, rather than yet another topic to fit into a full course. The background material is a high-level overview of:

- How non-technical tasks are often naturally done in parallel.
- Various physical class activities in which the students play the part of cores, messages, etc, to vividly illustrate basic concepts. Some of these activities are based on work in [1].
- The terminology of PDC.

With this background material covered, students can then explore computing in parallel and see how this reinforces traditional CS1 topics. We used the multiprocessing module in Python for this. To see how the module can serve as a medium for traditional CS1 topics, while at the same time teaching core PDC concepts, consider the following examples:

- The multiprocessing module uses various high-level classes to accomplish parallelism. This serves as a medium for explaining concepts like modularity, abstraction, instantiation, methods, instance variables, and object state.
- The multiprocessing module depends on the Python parameter-passing mechanism to make possible actions like communication via a shared queue. So students can view this as another example for learning parameter-passing.

We will share our experiences in applying this module in CS1, as well as planned refinements for Fall 2011. In particular, we will share how the material may have been introduced a little too early (around week 4), our original motivation for this, the result, and planned revisions.

III. CS0

We are currently investigating opportunities in Wittenberg University's "Computing in the Arts and Sciences" course. This is a very basic course, involving some simple programming in Scratch, along with the use of Excel and Access. Students that take this course are usually in other disciplines and will take no further computing courses. So we feel it is crucial to give them a level-appropriate introduction to PDC concepts. This introduction will consist of two parts:

- *The background material, very similar to that described for CS1*: parallelism in nature, physical activities demonstrating core concepts, and basic terminology. This can enable the students to be conversant in these core PDC concepts, though their actual technical experience in them will be minimal.
- *Programming in Scratch*: Scratch is designed to be a very gentle introduction to programming in a drag-and-drop environment. Interestingly, PDC concepts are also very naturally integrated into Scratch programming. The programs consist of multiple "sprites" (e.g. a dancing bear, a young man) which are able to receive commands simultaneously and communicate with each other in both blocking and non-blocking forms. This is fertile ground for parallel programming in an environment comfortable for extremely novice computer users. We look forward to exploring these ideas further this semester, and will have more to report by the time EduPar-11 is held.

IV. Computational Models and Methods

We are also developing PDC curricular materials for "Computational Models and Methods", a core course in Wittenberg's computational science minor and applied mathematics major. This course features the use of Mathematica and follows CS1 in sequencing. Students who take this course are drawn from many disciplines, including math, chemistry, physics, biology, economics, geology and, of course, computer science. The course itself emphasizes student preparation for systems analysis, mathematical techniques and numerical methods that are the foundations for many larger scientific applications. The PDC components in development for this course include:

- *Building on background information in PDC presented in CS1*: review of parallelism core concepts of multi-processing and terminology to reinforce the foundations presented in CS1.
- *Expanding awareness of common aspects of parallel computing*: using parallel computing constructs in Mathematica, students will gain increased appreciation of the core PDC concepts that are shared among programming languages for readily parallel algorithms and simulations
- *Introducing key concepts for distributed and shared data computing*: using Mathematica's intrinsic parallel functions, students will gain experience with parallel computing using larger arrays of data including data distribution, atomic operations, load balancing and task partitioning.

V. Programming Languages

As a part of our Programming Languages curriculum, High Performance Computing is included as a programming paradigm, using Chapel as the instructional language. In Spring 2010, the course had about three weeks devoted to learning PDC programming language tools. During that time, we covered different parallel syntax structures at both the process level and data structure level, explored the difference between global and task-local variables, learned to use built-in optimization with the reduce and scan operations, used domains as indices for large data sets, and practiced controlling the amount of parallelization used.

Demonstrations were given in class, but students were also expected to complete programming projects on their own. During the three weeks, we applied parallel programming to perform matrix operations and even test instances of the Collatz conjecture, with new projects to be explored in the Spring of 2011. Already in the first year, students were extremely excited to see the speedup in code and devoured each new PDC tactic. Chapel was especially applicable to the course since the language is still in a prototype stage and missing features can be discussed as a general Programming Languages topic.

VI. Algorithms

In Algorithms, again our goal is to weave relevant PDC concepts into the curriculum instead of tacking on more material. Thus, for each new algorithm covered in class, we first study sequential versions, followed by explanations of increasingly better parallel solutions. Some basic topics covered are work efficiency versus time efficiency, and distributed computing models such as CREW (Concurrent Read; Exclusive Write), CRCW, etc. During the course, students are exposed to parallel versions of many algorithms, including max/min/sum and other reduce operations, sorting (including merge, selection, insertion and bubble), integer multiplication, nearest-neighbor, graph traversing and Boruvka's algorithm for finding a minimal spanning tree.

In order to provide hands-on experience with algorithm design and implementation, students program in Chapel and run the code on our cluster. As part of the projects, they answer questions about run times and changes in efficiency when adding processors, comparing their results to the theoretical expectations.

Acknowledgment

We gratefully acknowledge the support of the National Science Foundation through grant CCF-0915805, SHF:Small:RUI: Collaborative Research: Accelerators to Applications – Supercharging the Undergraduate Computer Science Curriculum.

References

[1] Maxim, B. D.; Bachelis, G.; James, D.; and Stout, Q. 1990. Introducing Parallel Algorithms in Undergraduate Computer Science Courses. In *Proceedings of the Twenty-First SIGCSE Technical Symposium on Computer Science Education*, p. 255. New York, NY, USA: ACM Press.