Making Parallel Programming Accessible to Inexperienced Programmers through Cooperative Learning

Lori Pollock and Mike Jochen
Computer and Information Sciences
University of Delaware
Newark, DE 19716
{pollock, jochen}@cis.udel.edu

Abstract

This paper describes how we utilized cooperative learning to meet the practical challenges of teaching parallel programming in the early college years, as well as to provide a more real world context to the course. Our main contribution is a set of cooperative group activities for both inside and outside the classroom, which are targeted to the computer science discipline, have received very positive student feedback, are easy to implement, and achieve a number of learning objectives beyond knowledge of the specific topic. These activities can be applied directly or be easily adapted to other computer science courses, particularly programming, systems, and experimental computer science courses.

1 Introduction

As parallel computing is becoming a cost effective way to achieve large performance gains, these high performance computing systems are changing the nature of research and development across all areas of science and engineering. Computing theory and practice have begun a rapid transition to parallel architectures, opening new opportunities in algorithms and software and hardware design. However, skilled scientists are needed to exploit the opportunities that parallelism presents, which mandates modifying our curricula to teach computer science, engineering, and other science students to use this technology effectively. We have developed a sophomore level foundations course in parallel programming, for which C/C++ programming and data structures are the only prerequisites. Providing students with parallel programming skills early in their studies enables integration of parallelism into core junior-level courses such as algorithms, operating systems, and programming languages, as well as many of the elective upper level courses, including simulation, computer architecture, databases, and artificial intelligence. However, our experiences have revealed several challenges that must be addressed to successfully teach parallel programming to inexperienced programmers. Parallel programming requires a very different approach from traditional sequential programming, as the programmer must think of performing tasks in parallel, communicating information, coordinating actions, and balancing workload between parallel processes. Making the switch from thinking in sequential mode to thinking in parallel is a big step for many students, particularly, when they are trying to understand why their program is not working.

Furthermore, given the cost effectiveness and general availability of clusters, we chose to use the standard message passing interface, MPI[6], as our parallel programming paradigm. Besides its advantages of portability and free availability, it remains the most frequently used and most well accepted parallel programming paradigm. The disadvantages of MPI on a cluster are the cryptic error messages, and the lack of stable and useful debugging tools. This kind of environment is particularly challenging for inexperienced programmers. In addition, a parallel computing course is by nature an experimental computer science course, where students are not only concerned about the correctness of their program, but also focused on how well their program performs under different conditions (e.g., number of processes, sizes of data sets,...). Students have typically not had to go that extra step at this level to analyze why their program is not performing well, in this case, in parallel, and determine how to improve its performance. This requires critical analysis of their programs. These observations led us to reexamine the course pedagogy.

The first two instantiations of this course were designed

similar to the majority of computer science courses in colleges and universities today. Classroom time consisted of lectures, albeit, with considerable question and answer interaction. Students worked individually on small programming exercises outside the classroom to grasp the practical hands-on experience with the various features of parallel programming and MPI. Due to memory and disk space limitations, small data sets were used. Unlike other courses, the students performed experimental performance evaluations and were required to write scientific reports describing their experiments similar to other sciences. Students struggled with writing these reports and the analysis of their performance results as it was quite new to them in computer science, and writing a report for a programming assignment appeared to be busy work to them. Because assessment was based on the individual programming assignments and test scores, students were not permitted to work together or help one another debug their program. This approach not only did not reflect the real world of parallel computing in the research and development labs, but it did not provide a supportive learning environment that addresses the challenges of parallel programming for inexperienced programmers.

In this paper, we describe how we utilized cooperative learning to meet the practical challenges of teaching parallel computing in the early college years, as well as to provide a more real world context of parallel programming throughout the course. Cooperative learning through peer groups has long been promoted to increase student depth of learning, comfort level, confidence, motivation, higher order thinking, and learning skills[4, 1, 5, 7, 3]. It has also been shown to increase retention (especially of women and non-traditional students) and reduce gender bias in the classroom[8, 9, 2].

The main contribution of this paper is a set of cooperative group activities, for both inside and outside the computer science classroom, which are targeted to the computer science discipline, have received very positive student feedback, are easy to implement, and achieve a variety of learning objectives in an experimental computer science course. We believe that all of these activities are also appropriate for other programming, systems, and experimental computer science courses beyond parallel programming.

2 Group Activities in the Classroom

We experimented with a number of different classroom activities in groups of 3-4 students. Groups were sometimes formed randomly by counting off by 4's, other times by the formal project groups, and still other times formed by grouping the students sitting close to one another. For the classroom activities, all methods worked equally well. Classroom group work included mystery program readings, program solution sharing and analysis, group problem solving reviews for exams, problem specification clarifications, and discussion groups.

Mystery program readings. The main goal of this activity was to demonstrate the use of several new parallel constructs to students through examples. However, the activity had several additional learning objectives, including the ability to read, understand, and explain parallel programs, and the verbal critique of a parallel program with peers.

Each group was given 5 mystery MPI programs, and assigned responsibility for one of them. Each mystery program exhibited a different parallel construct or parallelization method. Each program was less than 2 pages in length. Students were told to come prepared to class with their MPI books. Each group was asked to converge on a group answer to the following set of questions for their assigned program: (1) For each MPI command that you have not yet seen, use your textbook to discover and write in English exactly what each instance of that command is doing. Note that MPI is a library, and each instance of an MPI command is a call to the library with some set of parameters that determines the effect of executing the command. (2) Each program had one or two points in the program which had been marked with a star. For each of these marked points, the group was to draw a picture of each process's memory contents, assuming that there are 2 processes. (3) Write a short description of what the program does, not only statement by statement, but a short paragraph of the overall program task achieved.

Each group chose a different person from the group to present the results of each of the tasks to the entire class. The presentations to the class were made on slides that the groups prepared in class. For some students, this was the first time they had ever discussed a program with other students (legally!). The challenge of solving the mystery of each program without any clues as to the overall goal of the programs was very enticing to many of the students. When some groups finished earlier than others, they began to discuss the other groups' mystery programs without any coaxing. By limiting the discussions to small groups, students were less intimidated to speak, and many students who would not speak up in class were participating actively in the discussions. This approach also allowed the entire class to cover 5 examples as opposed to 1 or 2 without the instructor presenting a single example.

Program solution sharing and analysis. While few people enjoy grading programming assignments, this task can actually be a valuable learning experience since the grader needs to examine and assess the various solutions identified for the same programming problem.

This classroom group activity had the goal of giving the students the opportunity to gain that same kind of assessment experience where different solutions developed by the students themselves were compared in terms of correctness, performance, and cleverness.

On days that individual parallel programming assignments were returned to the students, we formed our small groups, and each person in the group explained their program and experimental results to the group. The group then voted on the "best" solution to represent their group based on a set of criteria which focused primarily on correctness, generality, performance and clarity. The group developed a justification for their choice, and identified the most common alternatives and their pitfalls with respect to the group's best solution. The class was told before they started to avoid negative comments, and to present their critiques in a positive light. A spokesperson from the group presented the group's best solution to the class. The class then discussed the tradeoffs of all of the presented best solutions. Considerable discussion of different ways to perform specific subtasks was spurred by these discussions.

Besides exposing each student to a number of different solutions to the same problem, the students gained experience in assessing and comparing different solutions, and presenting a case for choosing one solution over another one. This activity also created an atmosphere within the class that it was acceptable, comfortable, and even very beneficial, to discuss the students' different solutions to the programming problems throughout the remainder of the semester.

Group problem solving reviews. Like many courses, this course included a midterm and final exam for individual assessment. In the past, the typical review session for an exam consisted of the professor answering student questions and presenting minilectures to review important points. Instead, we spent two class periods in our project groups working on the problem-solving sections of the past year's exam. Students who attended the sessions were motivated to work as a team in order to boost their grades if they indeed performed well as a group on the problems. If they were not satisfied with their group's performance, or did not attend, they had the option not to have this grade counted toward their final course grade.

This activity gave the students a flavor for the kinds of problems on the exam, and eventually both the questions and answers from the past year's exam problems. This activity helped considerably in relieving their anxieties over exams. The students liked the opportunity to discuss different approaches to the problems in a non-intimidating setting with a professor to give feedback on the spot as they "studied". They also liked the opportunity to work as a team to raise their grade with no threat of hurting their grade by group work.

Problem specification clarifications. A big part of programming and consulting jobs is gaining a clear understanding of the problem specification, mutually acceptable to the customer and programmer. Problem-based learning tries to mimic this by having students work in groups on ill-defined, complex problems. Students are given the task of posing questions, gathering knowledge, organizing ideas, and continuing to define new learning issues as they progress through the problem. While one of our group programming projects performed outside class focused on taking a problem-based learning approach, we also devoted some class time to specification clarification sessions.

In these sessions, the project groups were given an ill-defined parallel programming problem, and charged with posing a set of questions that they thought needed to be answered in order to more clearly define the problem specification. For each question, they were charged with developing a reasonable set of assumptions and parameters for experimentation with respect to the problem specification and evaluation of potential solutions. Each group chose a different person from the group to present the questions, assumptions, and parameters for experimentation to the entire class. A complete list of questions was created from the group lists, and the parameters for experimentation were discussed as a class.

This experience gave students exposure to one aspect of problem-based learning in which they had to better define an ill-defined problem, and develop reasonable assumptions. This is the first time many of these computer science students had been presented with an ill-defined programming problem, without all the exact specifications laid out for them in fine detail.

Discussion groups. Short discussion group meetings were held throughout the semester during class to break up classtime and get students actively discussing the issues related to the current topic. Each discussion period was initiated with a set of 3-4 questions for which the group was to agree on a group answer. Often, this involved examining a program segment and describing the approach, the possible motivation for the approach, and the potential pitfalls of the approach. Other times the questions were more open-ended to create discussion, with the comfort of a group of peers that agreed on an argument to present to the class.

3 Group Projects

The hands-on parallel programming experience in this course was achieved through a mix of small focused, concretely specified individual programming assignments and ill-defined, open-ended group projects. The indi-

vidual and group assignments were distributed evenly throughout the semester, with the intent of getting the students comfortable with working on group projects early in the semester. The goal of the individual programming assignments was to gain experience with specific parallel programming concepts through the effective application of MPI features. In this paper, we focus on the three group projects.

Open-ended real-world program and experiment. The main objectives of the first group project, which was assigned after one individual programming project, were (1) to gain the experience of the project group environment common in the computer industry, (2) to creatively solve a somewhat open-ended problem after investigating multiple solutions with varying tradeoffs, and (3) to learn skills, apply knowledge, and seek new knowledge through a problem-based approach.

The initial specifications for the project were merely the input and output format, overall task to be performed, and goals of correctness, performance, and generality. Each group was to develop an edge detection processor which when given an image, returns an altered version of that image that delineates all the edges within the original image. Due to the large computation times for sequential versions, this is a commonly parallelized application. The project had 3 deliverables with deadlines of one week for the first two deadlines, and 2 weeks for the third deadline, so the entire project lasted about one month. The first deliverable was a written report that answered a set of 8 questions, which directed the students to perform some research on available algorithms for edge detection, explore the tradeoffs of the algorithms, select an algorithm to implement, discuss and agree upon a way to handle image borders, determine how to perform file input/output, explore the potential performance issues, and identify the desirable characteristics in a good test suite.

The second deliverable was a prototype implementation, either as a correct sequential program, or a first correct version of a parallel system. For this deliverable, there were no expectations of good performance, only correctness. The third deliverable focused on meeting all the requirements of the original specification, focusing on obtaining good parallel performance, conducting an empirical study of performance under different parameters, and writing a group experimental report and user manual. Each deliverable was evaluated and graded with the same weight. Each student also performed a peer review for each member of their group.

For many students, this was their first encounter with a group project that involved multiple deliverables, and a deliverable that merely had the goal of researching the possible solutions and making a plan without any programming. Students liked the feedback after each deliverable, and used the feedback to improve their plans for the next deliverable. They also entered into interesting discussions over which algorithms to use, and how the best sequential algorithm often did not lead to the better parallel algorithm. They had an opportunity to go back and modify their ideas from the earlier deliverables, and sometimes completely scrapped those ideas due to implementation difficulties or realization that there was a better way to gain performance in parallel. The project mimicked the real world programming experience much more closely than typical programming assignments with a single deadline. The experience was different from a software engineering course, as the lines of written code was not substantial, and little time was spent on software engineering design.

Research project and presentation. The goals of the second group project were to explore the role of parallelism in a variety of application domains, to learn how to research a particular computer science topic in a focused manner, and to collectively organize and present a talk that overviews the findings. For the first deliverable, the groups performed a quick search for information on a set of application domains in order to make their selection of a domain of interest. They needed to determine whether they could present adequate information on the problem, the common sequential solution, motivation for parallelization, the common parallelization method, and any experimental performance results demonstrating the gains from parallelism.

The second deliverable was a draft of their slides for their presentation, and a plan of who would present each part of the presentation as each member had to share in the oral presentation. Detailed comments on the drafts were given in a group meeting with the professor or via written comments. This feedback not only improved the final presentations tremendously from a previous instantiation of the course where drafts were not used, but the confidence of the student speakers was increased, especially those who had not previously talked in front of an audience. The third deliverable was the presentation to the class; most talks were performed with laptops. The talks were geared to 15 minutes shared among the 2-3 group members.

Each student completed an evaluation form for each group presentation. While these evaluations were not used for grading, the feedback was summarized and anonymously given to each group. The evaluation of others was included as part of each student's grade in order to promote attendance and participation in the evaluations.

Parallel programming contest. The third group project was an open-ended parallel programming assign-

ment posed as a programming contest. The only specification for the contest was to write a parallel sort program in MPI, with specific input and output file specifications to ease the grading. A set of contest rules was set up to insure fairness in judging performance. Scoring was based on (40%) correctness, (40%) performance, (8%) creativity, (8%) generality, and (4%) documentation. Entries were judged by a panel of 6 professors, research scientists, and teaching assistants. The grades were assigned independent of the judges and results of the contest. The top two teams were given t-shirts.

The contest setting for this project really encouraged the teams to work together to design their best possible parallel program, with careful consideration of tradeoffs between the various parameters used for judging. Students were already comfortable working in project groups and had a good grasp of how to exploit the strengths of each of their team members.

4 Lessons Learned

The student evaluations at the midterm and end of the semester were very positive for all of the cooperative classroom activities. They overwhelmingly believed that the mystery program readings and the group problem solving reviews for exams were very helpful, and suggested that they be used in other courses. The groups certainly created open discussion during class, as the students presented solutions as a group and then became more comfortable with speaking in class. Fortunately, each individual activity took much less time than lecture preparation on the same material.

Students liked the mix of individual and group projects, but believed they learned more from working in groups for the projects than programming in isolation. The expected group management problems occurred, but they all had workable solutions since the groups were given adequate opportunity to voice their concerns early. The group projects permitted more complex problems to be considered than what could ordinarily be covered with individual assignments. The only feedback on the programming contest was several groups informally expressing their enjoyment for it and their comments on their strong group effort, since the contest was run to completion only after course evaluations had already been completed. It did appear that the competitive aspect of the contest heightened the interest of many students.

The main lesson we learned was that student assessment needs to include a mechanism for more individual responsibility within group projects. All members of a group could start with the same grade, but then that grade needs to be adjusted for each member by some percentage that reflects their individual contribution measured by peer evaluations and self assessment.

5 Concluding Remarks

Our experience in creating and implementing these cooperative learning activities specific to a parallel programming course has not only demonstrated to us many of the positive implications of cooperative learning, but more importantly, shown us how easy it can be to create such activities for computer science. These activities can play a particularly important role when you want to challenge students to go beyond writing the basic correct program in order to analyze and improve the performance of their programs, or inexperienced programmers are presented with an unfriendly working environment, or a major goal is to mimic the real world environment with ill-defined, complex problems.

References

[1] Bonwell, C. C., and Eison, J. A. Active Learning: Creating Excitement in the Classroom. *ASHE ERIC Higher Education Report No. 1* (1991).

[2] Chase, J. D., and Okie, E. G. Combining Cooperative Learning and Peer Instruction in Introductory Computer Science. *Proceedings of ACM SIGCSE* (2000).

[3] Dougherty, R. C., Bowen, C. W., Berger, T., Rees, W., Mellon, E. K., and Pulliam, E. Cooperative Learning and Enhanced Communication: Effects on Student Performance. *Journal of Chemical Education* (1995).

[4] Johnson, D. W., Johnson, R. T., and Smith, K. A. Cooperative Learning: Increasing College Faculty Instructional Productivity. *ASHE ERIC Higher Education Report No. 4* (1991).

[5] McConnell, J. J. Active and Group Learning and Their Use in Graphics Education. *Computers and Graphics* (1996).

[6] Message Passing Interface Forum. MPI: A message-passing interface standard. *International Journal of Supercomputer Applications 8*, 3-4 (1994).

[7] Silberman, M. L. *Active Learning: 101 Strategies to Teach Any Subject.* Allyn and Bacon, 1996.

[8] Tenenberg, J. Using Cooperative Learning in the Undergraduate Computer Science Classroom. *Proceedings of the Midwest Small College Computing Conference* (1995).

[9] Walker, H. M. Collaborative Learning: A Case Study for CS1 at Grinnell College and UT-Austin. *Proceedings of ACM SIGCSE* (1997).