Teaching Parallel Computing to Freshmen*

Donald Johnson, David Kotz, and Fillia Makedon

Department of Math and Computer Science
Dartmouth College
Hanover, NH 03755-3510
{djohnson,dfk,makedon}@cs.dartmouth.edu

Abstract

Parallelism is the future of computing and computer science and should therefore be at the heart of the CS curriculum. Instead of continuing along the evolutionary path by introducing parallel computation "top down" (first in special junior-senior level courses), we are taking a radical approach and introducing parallelism at the earliest possible stages of instruction. Specifically, we are developing a completely new freshman-level course on data structures that integrates parallel computation naturally, and retains the emphasis on laboratory instruction. This will help to steer our curriculum as expeditiously as possible toward parallel computing.

Our approach is novel in three distinct and essential ways. First, we will teach parallel computing to freshmen in a course designed from beginning to end to do so. Second, we will motivate the course with examples from scientific computation. Third, we use multimedia and visualization as instructional aids. We have two primary objectives: to begin a reform of our undergraduate curriculum with an laboratory-based freshman course on parallel computation, and to produce tools and methodologies that improve student understanding of the basic principles of parallel computing.

1 Introduction

The High Performance Computing and Communications Program [NSF91] cites many grand challenges, including weather prediction, molecular structure models, and human genome mapping. Meeting these challenges will require two things of computer scientists: an understanding of the sciences and an ascent to unprecedented levels of algorithmic and computational power. The latter will require computational scientists to apply highly parallel

*This research was supported under grant DUE-9352796 by the National Science Foundation ILI-LLD program.

resources to solve large problems. The curricular need, then, is to educate CS students in the relation of the sciences and computation, and in all aspects of parallel computation. We believe that the greatest success toward these objectives on the educational front will come from starting early, with freshman courses, and employing examples from the sciences and the best tools for conceptualizing the computational process through computer visualization.

We plan to begin teaching parallel computing early in the curriculum — indeed, to freshmen — through intensive use of visualization and multimedia tools. The eventual goal is to integrate parallel computing into the entire curriculum. In the remainder of this paper, we describe the philosophy behind this decision and then describe the course we are now developing.

2 Philosophy

We believe that parallelism is the future of computing and computer science and that it should therefore be at the heart of the CS curriculum. Dartmouth, like many other schools, is presently taking the evolutionary path of introducing parallel computation in special junior-senior level courses. This *top-down strategy* has been the approach of most similar efforts to introduce parallel computation into the undergraduate curriculum [BEW88, FG91, HS91, Hyd89, Joh92, Mer92], although some have introduced parallel computation into parts of other courses [SH90, Sil89, Har92, Whi88, Yea91].

We plan a more radical approach, to introduce parallelism *bottom-up*, at the earliest possible stages of instruction. This will provide a foundation for the higher level courses, such as languages, operating systems, and architecture, that will soon be covering aspects of parallel computing. It will also help to steer our curriculum toward parallel computing.

Thus, we propose to develop a new freshman-level course on data structures that integrates parallel computation naturally, and retains the emphasis on laboratory instruction. This course will eventually replace our current data structures course, CS 15. CS 15, our second CS course, is the logical choice because the first course (introduction to programming) is skipped by many of our best students through placement exams, and because the CS major branches in three directions after CS 15. Thus, the student clientele for the data structures course we propose includes the freshmen and sophomores who have completed (or placed out of) the introductory course in computer programming. They have learned Pascal or C++, using the Macintosh as a platform.

This approach is novel in three distinct and essential ways. First, we will teach parallel computing to freshmen (to our knowledge, no one has developed a completely new course emphasizing parallel computation at the freshman level). Second, we will motivate the course with examples from scientific computation. Third, we will use multimedia and visualization as instructional aids. While the use of visualization to teach parallel computation is not new to computer science education [Hay88, Mye86, Nap90, Rob92, Sch92, Wag89], the effective integration of interactive visualization and multimedia into a rewarding experience for freshmen is unusual.

3 A New Course: Parallel Data Structures and Algorithms

Students enter CS 15 having learned to program in Pascal or C++, the use of arrays and lists, and some basic algorithms in their first CS course. CS 15 is currently organized around the concept of the abstract data type (ADT). A typical implementation of the syllabus is as follows: students first write a package that implements linked lists with arbitrary data. Then they proceed through a series of assignments that employ this ADT: a polynomial manipulation package, an implementation of a heap, and an implementation of binary search trees (with either red-black- or splay-tree balancing). Lecture material and written assignments provide instruction in algorithm invention and analysis, e.g., for sorting. There is also an extensive hashing experiment.

The principles covered by our new course will be (1) the need for parallel machines *vis-a-vis* scientific problems, (2) the solution of basic algorithm and data structure problems in both the serial and shared-memory models, (3) case studies from the sciences, (4) abstract data types, and (5) basic parallel programming techniques. By designing a new course from scratch we believe we can cover parallel computing along with the traditional concepts in a coherent fashion.

3.1 Tentative Outline of the Course

I Introduction
- A. Parallelism in problems from the sciences
- B. Processing units, memory modules, interconnections
- C. Synchronization of computation
- D. Complexity analysis, serial and parallel

II Data Structures
- A. Arrays and lists
- B. Manipulating arrays and lists in parallel
  1. List ranking and prefix sum
  2. Matrix multiplication
- C. Index structures
  1. Binary search trees
  2. B-trees
  3. Concurrent access to B-trees
  4. Hashing

III Case studies of parallel algorithms and data structures
- A. Ray tracing
- B. Solving linear systems
- C. Databases
- D. Genome sequence comparison

Examples of Possible Programming Assignments

1. List ranking (parallel)
2. Polynomial evaluation (serial and parallel)
3. Matrix multiplication (parallel)
4. Convex hull (serial and parallel)
5. Simulation of a physical problem (parallel)
6. Group database projects (parallel, supported by some extensive software infrastructure in order to be feasible)

3.2 Methodology

There are three implementation considerations for teaching this course: choosing a parallel programming platform, choosing a textbook, and the use of visualization and multimedia.

Parallel Programming. To avoid the "informational clutter" of a survey course, we will carefully select *one* programming model, language, and platform, to minimize implementation details and maximize the conceptual content. Our tentative plan is to use the Parallaxis programming language on the Macintosh. The Parallaxis programming language [BBES91] supports the SIMD programming paradigm, can simulate many topologies, is similar to the Pascal language that many students already know, runs a variety of platforms (including Macintosh, Unix, and the Maspar parallel processor), and is free. The SIMD model is, we believe, the easiest to teach due to its single thread of computation. The Macintosh platform is familiar to Dartmouth students, is easily accessible on campus, supports many multimedia tools, and is readily available to other schools.

Textbook. We do not know of any existing freshman-level text that fits. For this reason, we will develop our own electronic lecture notes to accompany each of ten *modules* of instruction, one for each week of a class. Each module will be composed of sections that include introduction and background, theory, examples, exercises for the student, solutions, and references. In many cases we will include hypermedia, with clips of video, audio, and animation.

Multimedia. The use of visualization and animation has been shown to help students learn computer science [Hay88, Nap90]. Multimedia materials will be used in lecture and for self-study, for the animation of parallel data structures, visualization of algorithms, experimentation with various architectures, monitoring of parallel programs, and demonstration of scientific visualization, with a focus on algorithmic over architectural issues. These tools include animation programs, hypermedia, video clips, and audio. The goal is to create a well-managed and seamless exploratory environment [BK89, Bro88, Sta91], including tools for program visualization [Mye86, SBN89]. The course will thus be composed of ten multimedia modules.

As an example of a multimedia module that employs video, text, graphics, algorithm animation, data structure visualization, and hypermedia, consider a case study on genome sequence comparison. We would begin by describing the Human Genome Project. A short video clip of an expert explaining the importance of fast searches for approximate matches between gene sequences would be combined with a graphical simulation of a DNA molecule. This presentation would provide background for the study of the computational problem of genetic sequence comparison, for which there are several parallel techniques. The module would provide one or more matching algorithms, along with animations, for the student to study and to examine their behavior on a fairly large genome database.

4 Status

We plan to teach this course for the first time next year. We are currently in the process of developing the course syllabus and software-support tools. Over the summer we will develop the classroom-demonstration and laboratory modules.

There are many issues still to be resolved, including the appropriate balance and representation of sequential and parallel topics, the importance of real parallel hardware *vs.* simulations, the representation of diversity in parallel systems without loss of pedagogical coherency, the choice of debugging and performance monitoring tools, the challenge of visualizing concurrency and parallelism, and the selection of scientific visualization examples for motivational purposes.

References

[BBES91] Ingo Barth, Thomas Bräunl, Stefan Engelhardt, and Frank Sembach. *Parallaxis Version 2 User Manual.* Universität Stuttgart, February 1991. Computer Science Report Number 2/91.

[BEW88] Ralph M. Butler, Roger E. Eggen, and Susan R. Wallace. Introducing parallel processing at the undergraduate level. In *19th SIGCSE*, pages 63–67, 1988.

[BK89] Jon Bentley and Brian W. Kernighan. A system for algorithm animation. *Computing Systems*, 23(3):137–146, July 1989.

[Bro88] Marc H. Brown. *Algorithm Animation.* MIT Press, Cambridge, MA, 1988.

[FG91] Allan L. Fisher and Thomas Gross. Teaching the programming of parallel computers. In *22nd SIGCSE*, pages 102–107, 1991.

[Har92] Stephen J. Hartley. Experience with the language SR in an undergraduate operating systems course. In *23rd SIGCSE*, pages 176–180, 1992.

[Hay88] Helen Duerr Hays. Interactive graphics: A tool for beginning programming students in discovering solutions to novel problems. In *19th SIGCSE*, pages 137–141, 1988.

[HS91] Janet Hartman and Dean Sanders. Teaching a course in parallel processing with limited resources. In *22nd SIGCSE*, pages 97–101, 1991.

[Hyd89] Daniel C. Hyde. A parallel processing course for undergraduates. In *20th SIGCSE*, pages 170–173, 1989.

[Joh92] David J. John. Integration of parallel computation into introductory computer science. In *23rd SIGCSE*, pages 281–285, 1992.

[Mer92] Marsha J. Meredith. Introducing parallel computing into the undergraduate computer science curriculum: a progress report. In *23rd SIGCSE*, pages 187–191, 1992.

[Mye86] Brad A. Myers. Visual programming, programming by example and program visualization: A taxonomy. In *SIGCHI Proceedings*, pages 59–66, April 1986.

[Nap90] T. L. Naps. Algorithm visualization in computer science laboratories. In *21st SIGCSE*, pages 105–110, 1990.

[NSF91] *Grand Challenges: High Performance Computing and Communications.* A Report by the Committee on Physical, Mathematical and Engineering Sciences, NSF/CISE, 1800 G Street NW, Washington, D.C. 20550, 1991.

[Rob92] James Robergé. Creating programming projects with visual impact. In *23rd SIGCSE*, pages 230–234, 1992.

[SBN89] David Socha, Mary L. Bailey, and David Notkin. Voyeur: Graphical views of parallel programs. *ACM SIGPLAN Notices*, 24(1), 89.

[Sch92] D. Schweitzer. Designing interactive visualization tools for the graphics classroom. In *23rd SIGCSE*, pages 299–303, 1992.

[SH90] Dean Sanders and Janet Hartman. Getting started with parallel programming. In *21st SIGCSE*, pages 86–88, 1990.

[Sil89] James L. Silver. Concurrent programming in an upper-level operating systems course. In *20th SIGCSE*, pages 217–221, 1989.

[Sta91] John Stasko. Using direct manipulation to build algorithm animations by demonstration. In *Proc. of the ACM SIGCHI '91 Conference on Human Factors in Computing Systems*, pages 307–314, May 1991.

[Wag89] Leslie J. Waguespack. Visual metaphors for teaching programming concepts. In *20th SIGCSE*, pages 141–144, 1989.

[Whi88] G. M. Whitson. An introduction to the parallel distributed processing model of cognition and some examples of how it is changing the teaching of artificial intelligence. In *19th SIGCSE*, pages 59–62, 1988.

[Yea91] Dorian P. Yeager. Teaching concurrency in the programming languages course. In *22nd SIGCSE*, pages 155–161, 1991.