An Undergraduate Course in Parallel Computing for Scientists and Engineers[1]

Manavendra Misra
Dept. of Mathematical and Computer Sciences
Colorado School of Mines
Golden, CO 80401
Ph. (303)-273-3873
email: mmisra@mines.colorado.edu

Abstract

Parallel Computing is traditionally treated as a highly specialized area of study, and courses in this area are therefore designed with the advanced Computer Science graduate student in mind. This thinking excludes non computer scientists from obtaining formal instruction in parallel problem solving techniques that might prove useful to them in their careers. To rectify this shortcoming in curricula, we have designed a course to teach parallel computing to science and engineering undergraduates. A prototype of this course was taught at CSM in Spring 1993. A unique feature of this course was the involvement of project mentors to guide students on individual projects. The mentors provide students with exciting projects which have to be solved on parallel machines. The students start designing the problem solution early in the semester, and learn appropriate skills from the course "just in time" to apply them to the project. This paper describes the prototype course, and plans for the future.

1 Introduction

The course described in this paper attempts to rectify what the author perceives as a severe shortcoming in traditional curricula. In most universities, parallel computing is treated as a highly specialized area of knowledge that is taught only to computer science graduate students. This philosophy makes the parallel computing courses inaccessible to non computer science students. Many graduates from these scientific and engineering areas will feel the need to use parallel and high performance computers to solve complex problems in their areas of expertise. In fact, most computational scientists who use parallel machines are from fields

[1]Parallel hardware for future offerings of this course will be procured through NSF ILI-IG grant DUE-9450874

other than computer science. Since so many non-computer scientists use high performance parallel machines to solve problems, the lack of formal training in parallel problem solving skills often results in sub-optimal solutions to the problems. Students from science and engineering disciplines would therefore benefit immensely from a course in parallel computing designed with their needs in mind.

Another drawback of teaching parallel computing only to graduate students is that it is too late to prevent students from developing a habit of using sequential problem solving techniques for all problems, even if the problems are inherently parallel. It is therefore important to provide scientists and engineers an early introduction to parallel problem solving.

A number of other educators have tried to address one or both of the aspects of the problem identified above. Most of the discussions in [3, 4, 6, 7, 8, 9, 10, 11, 12, 15, 17] address the instruction of parallel computing to computer science undergraduate students. A sequence of courses on High Performance Scientific Computing have been developed at the University of Colorado (CU) at Boulder as a result of an Educational Infrastructure Grant from the CISE division of the NSF [5]. The first course familiarizes students with three high performance machines, and teaches the students how to use software tools for visualizing data. The second course in the sequence teaches the students how to solve three scientific problems on each of the three machines. The author is currently involved in an effort to extend the diversity and number of application problems in this project. The author is also involved in a DOE sponsored initiative called UCES [2] (Undergraduate Computational Engineering and Sciences project). The goal of the UCES project is to develop modules for teaching computational science to undergraduate students. The DOE has also sponsored a similar project at the graduate level. This project is called CSEP (Computational Science Education Project), and it has resulted in an electronic text-book that can be retrieved via the World Wide Web [3].

The above discussion points towards a strong need for courses that teach parallel computing to science and engineering students at an early stage of their education. A prototype course motivated by this need was taught at the Colorado School of Mines in the Spring of 1993. This paper describes the prototype course, discusses the lessons learnt from teaching the course, and describes how the course will be taught in the future, beginning in Fall 1994.

2 The Spring 1993 Prototype

The need described in the previous section motivated the development of a prototype course in "Scientific Supercomputing" at the Colorado School of Mines in the Spring semester of 1993. The course was developed and taught by the author, with considerable help from Professor Jean Bell of the Mathematical and Computer Sciences department. In this section, we shall discuss various aspects of this course, and the lessons learnt from it.

[2] The contact person for the project is Tom Marchioro (tlm@ameslab.gov).

[3] CSEP Materials can be viewed at http://csep1.phy.ornl.gov/csep.html using NCSA Mosaic.

2.1 Course Organization

The prototype course was taught as a combined graduate/undergraduate class. It was attended by undergraduates at the senior level, as well as beginning graduate students from a number of disciplines. This combination of backgrounds and levels resulted in a very productive environment, as will be disussed later in this section.

It was strongly felt that students would be best motivated by learning how real-life interesting problems could be solved using high performance computing. The prototype course was therefore built around individual semester projects that required the students to use parallel computing to solve fairly complex problems. Unlike courses where the instructor choses specific applications, students were encouraged to choose projects in areas that they found most interesting. Thus, the emphasis of instruction changed from teaching details of specific applications, to parallel computing issues that enhance general problem-solving techniques. Since the students were working on projects that they found exciting, it was expected that there would be a greater eagerness to learn.

Involvement of Project Mentors: A course like this requires the students to choose a project very early in the semester. However, it is unrealistic to expect students who don't have much of a computing background to decide on a parallel computing project of the appropriate complexity at an early stage. To facilitate the process of project selection, researchers from different areas of expertise were invited to give a short talk on some research project in their area that might benefit from parallel computing. Each student (in consultation with the instructor and the respective researcher) then chose one of these application areas for his/her term project. The researcher then played the role of client and mentor for the student project. Researchers working in robotics, neural networks applications in chemistry and mining, databases, computer vision, renewable energy resources, and geophysics acted as mentors for projects in the course (see Table 1). Since then, other faculty members in engineering, geophysics, and geology have expressed an interest in mentoring student projects in the course.

During the first two weeks, the students were given an introduction to parallel computing, and the prospective project mentors gave short presentations of the problems from their areas of research that they thought were appropriate for student projects. At the end of the first two weeks of class, the students turned in a problem-statement that briefly described the project that they chose. After this, students started work on the design of a parallel solution to the problem, with help from the course instructor and the project mentor. It was expected that the process of designing a parallel solution, and the ensuing implementation of their design would result in the students discovering that they lacked certain skills and knowledge to solve the problem on parallel machines. Instruction in the course therefore concentrated on providing these very skills. In this way, we intended to provide a setting in which the students could see how the knowledge learned in this course can be applied directly for solving real-life problems.

Midway through the semester, the students were expected to have completed the design of their solution. They briefed the instructor on their progress by submitting updated versions

of their report document (the students maintained a document that started off as their problem statement, and was updated periodically till it was in the form of the final report). The students then began implementing this solution on the parallel machine of their choice. At the end of the semester, the students were graded on a final project report, and an in-class presentation of the results of their project–including a discussion on the design and analysis of the algorithms used, and performance analysis of the solution (see Appendix A).

Instruction in the class was imparted through some lecturing, and in large part through group discussions. The small class size, and a lively group of students who came from different backgrounds made these discussions very fruitful and educational. This way, the computer scientists could help the non-computer scientists with some of the computing aspects, while the non-computer scientists provided an application flavor that some of the computer scientists were not used to. Also, the fact that we had both graduate and undergraduate students in the same room helped the undergraduates as they could get to see a graduate perspective on their projects.

2.2 Software Issues Covered

The following software issues were covered in the prototype course:

- Using Unix workstations: directory structures, email, network news, ftp, remote logins, make, etc. Many of the non-computer science majors did not have a familiarity with these concepts, and they were necessary to effectively use the supercomputers remotely (see Section 2.4).
- Design and analysis of sequential and parallel algorithms, performance analysis of a parallel program, optimizing performance. The students were taught how to time program runs on the single processor workstations, as well as on the parallel machines. They were introduced to concepts of speedup, efficiency, and processor-time optimality.
- A discussion on parallelizing compilers, parallel debuggers, integrated program development environments, graphical user interfaces, and their effect on program development. Although we did not have access to most of these tools for the machines we used, the students discussed how their project would benefit from these tools, and they also acquired an understanding of how difficult it is to build these tools.
- Writing parallel programs for shared versus distributed memory machines, the kinds of synchronization primitives required in each case.
- efficient data mapping onto different parallel architectures.

2.3 Computer Architecture Issues Covered

The following architecture oriented issues were covered in the prototype course:

- Taxonomy of parallel computers. Vector processors, SIMD computers, MIMD computers.
- The appropriateness of each class of machine for certain problem types. The analysis of the granularity of parallelism in a problem, and how it affects the choice of parallel machine.
- The distributed memory, message passing paradigm versus the shared memory paradigm.
- Interconnection networks, parallel computer architecture issues, and their effect on program development.
- The architectures of the three machines used in the course.

2.4 Resources Used

When the prototype course was taught in Spring 1993, CSM did not possess parallel hardware that could be used for the course. We did, however, have workstations with Internet access. Therefore, it was decided to use the resources available at the Supercomputer Centers through the Internet. Three machines were chosen so as to provide students access to representative computers in three different classes of high performance computers. Each computer was located at a different Supercomputer Center:

1. The Cray Y-MP was the representative chosen for the class of vector supercomputers. Access to the Y-MP was provided by the National Center for Atmospheric Research (NCAR). Due to the proximity of NCAR to our campus, we were also able to organize a tour of the NCAR facilities for the students. This was particularly motivating for the students as they got to see some of the machines that they were working on, in addition to the kinds of problems that were being solved on them by other scientists.
2. The Connection Machine CM-2 was chosen as the representative of the class of SIMD machines. Access to the CM-2 was provided by the Pittsburgh Supercomputer Center.
3. The nCUBE-2 was chosen as the representative of the MIMD class of machines. Access to the nCUBE-2 was provided by the San Diego Supercomputer Center.

The students were asked to solve sample problems on each machine during the first half of the course. They then had to analyze the problem they were solving for their term project, and decide which machine was best suited for the solution of the problem.

Textbook: The first edition of George Almasi and Alan Gottlieb's text book *Highly Parallel Computing* was used for the class. Unfortunately, the students found this to be too advanced a text for them. It seems like there is no single book that is perfect for a course of this nature. In Fall 1994, we expect to use *Introduction to Parallel Computing* by Lewis and El-Rewini, along with lecture notes developed by the instructor.

During the course, a heavy emphasis was placed on the use of electronic mail and network newsgroups for exchanging information. Unfortunately, a number of the students were not very comfortable with Unix, and the facilities that it provides. This meant that some time had to be devoted towards teaching these concepts. In future offerings of the class, one week will be reserved for teaching these concepts (see Section 3.1). An internal newsgroup was set up just for the course. This newsgroup facilitated communication between students and between the professor and students. The instructor also posted announcements of recent developments in the field of parallel computing (some of which were taken from newsgroups like comp.parallel, and from mailing lists like the UParCC [4] mailing list). Students could also use the newsgroup to discuss common problems, and help other students out.

2.5 Projects completed

Table 1 lists the projects completed by the students in the prototype course, along with the target machine used for each project, and the project mentors.

Title of project	Target Machine	Mentor
Prediction of Protein Structure using Neural Networks	Cray Y-MP	Prof. Lorien Pratt (CSM)
Recognition of Mining Machine Events using Neural Networks	nCUBE	Prof. Aaron Gordon (CSM)
Parallel Implementation of Robot Behaviors	nCUBE	Prof. Robin Murphy (CSM)
Parallel Implementation of the Seismic Migration Problem	CM-2	Prof. John Scales (CSM)
An Analysis of Parallel Relational Databases	nCUBE	Prof. Jean Bell (CSM)
Parallel Implementation of 2-D Wavelet Transforms	CM-2	Prof. Manavendra Misra (CSM)
Wind Simulation	Cray Y-MP	Neil Kelly (National Renewable Energy Lab)
Parallel Neural Network Simulation	CM-2	Prof. Lorien Pratt (CSM)

Table 1: The list of projects completed in the prototype course. The table shows the projects, the target machines the projects were implemented on, and the names of the project mentors.

3 Future Plans

The prototype course provided valuable experience in teaching a broad based course in parallel computing for undergraduates. The lessons learnt from the course will help determine

[4] UParCC stands for **U**ndergraduate **Par**allel **C**omputing **C**onsortium.

the changes that will need to be incorporated in future offerings of the course.

A feeling that was shared by some of the students was that often times, slow access over the internet, and overloaded machines at the Supercomputer Centers was an inconvenience. Many students felt that an on-campus parallel machine dedicated to the course would benefit them immensely. With this in mind, it was decided that we would invest in some affordable parallel hardware. The CSM administration provided a seed grant to purchase a 10-node transputer based machine, with a Sun front-end. We have also recently been awarded an NSF ILI-IG grant to purchase a larger machine capable of handling a full class, along with appropriate software.

Choice of Hardware: The two driving considerations in the choice of equipment for the course were: (a) budgetary considerations and (b) the suitability of the equipment in an undergraduate teaching environment. Although a number of commercially available parallel computers were considered for the course, most were outside our budget limitations. It was important for the purposes of this course to provide an adequate degree of parallelism to as many student users as possible at any given time, subject to the limits imposed by our budget. It was also important to have a parallel system that could be reconfigured so that students could compare performances across a number of different interconnection networks. Based on the above criteria, it became apparent that transputer based parallel machines would be best suited for this situation. Discussions with educators at other Universities who had tried to achieve similar goals as us [1, 14, 16, 18, 19] and with members of the UParCC group helped reinforce this decision.

Other than the transputer machines, there was an option of buying used parallel machines that would be within our budget. Another option was to purchase a shared-memory parallel machine (from a vendor who was going out of business) for a fraction of its original price. These options were discarded because of the possible maintenance problems they might have entailed.

Once a decision was made to buy a transputer machine, a number of vendors were contacted. In response to a posting on the UParCC mailing list and on comp.parallel, educators provided addresses and comments on vendors with whom they had interacted. Using this information, Parsytec, Transtech, Meiko, Computer System Architects (CSA), and Alta Technology Corporation were contacted. To provide maximum accessibility to the parallel system, it was decided to have the processors housed in an independent chassis that was connected to a Unix workstation through an interface. Since students had access to workstations connected to the network all over campus, they could login remotely into the front end (FE) workstation and thus get easy access to the parallel machine. Alta provided the lowest price quotations for such a set-up.

It was decided that a machine with 64 processors, that could be accessed by up to four independent users at a time, and whose interconnections could be reconfigured easily through software control, would be ideal for the required environment. This way, a student could configure an individual 16-node section in one of many interconnection schemes–a 4×4 two-dimensional mesh, a 4-D hypercube, a 16 node shuffle-exchange network, a 16 node linear

array, etc. Alternatively, a student could request all 64 nodes to solve a larger problem, or run tests on performance that require a large number of nodes. It was decided to configure all 64 nodes with 4MB of RAM, as it is envisioned that most student projects will be data intensive (seismic data analysis, image processing, neural networks). A system based on the Ultra/XL cards from Alta meets these specifications in the most cost-effective manner. Peer reviews of Alta's interface card have also been excellent [2].

Visualization: One major shortcoming of the prototype course was the lack of emphasis placed on the visualization of results. Students had a hard time making sense of the large amount of data that sometimes resulted from their projects. Some of them managed to use Mathematica for simple visualization tasks, but it became apparent that a course such as this should definitely have a visualization component to it. The latest version of Matlab from *The Math Works, Inc.* combines powerful numerical processing with excellent graphics and is ideal for this course. Matlab will be used in addition to public domain packages (e.g. Ximage, XCollage, GVLware) for visualization. We already have a site license for Mathematica, which can also be used for simple data visualization tasks.

Unix skills: The original schedule of the prototype course did not have a large amount of time earmarked for teaching Unix skills. It turned out that a number of the students did not feel very comfortable with these skills, and so the new schedule reserves a week for teaching these skills.

More group work: In the prototype course, students did individual projects over the semester. It might be worthwhile to form groups of two students from different backgrounds (and possibly groups with one graduate and one undergraduate student) to facilitate learning from each other.

Cooperation with other similar efforts: The author is involved with the UCES project, as well as with the enhancement of the CU Boulder HPSC project [5]. Such collaborations provide opportunities for the exchange of project modules, as well as a chance to exchange ideas with educators with similar goals.

3.1 Course Description and Schedule

The description of the course as it appears in the catalog is as follows:

MACS 463 : Parallel Computing for Scientists and Engineers. This course is designed to introduce the field of parallel computing to all scientists and engineers. The students will have access to state of the art supercomputers, and will be taught how to solve scientific problems on these machines. They will be introduced to various software and hardware issues related to high performance computing.

3 hours lecture, 3 semester hours

Prerequisites: Programming experience in C/Fortran, consent of instructor.

At the end of this course, the student should be able to demonstrate:

a. an understanding of the basic issues involved in using high performance computers.

b. a knowledge of how to write parallel programs on state of the art supercomputers.

c. a knowledge of how to solve scientific problems using parallel problem solving techniques.

A tentative course schedule for a 14 week semester is presented here:

Weeks 1 and 2 An overview of parallel computing, motivation for using parallel computing, suitable problems.

Presentations by prospective project mentors on their research.

Students submit a problem statement describing their project by the end of the second week.

Week 3 Using Unix workstations: Unix directory structures, e-mail, network news, ftp, remote logins, using Unix *make*. (This is necessary because a number of non-CS majors do not have these skills).

Weeks 4 and 5 Designing efficient parallel solutions to problems: the concept of an algorithm, design and analysis of sequential algorithms, design and analysis of parallel algorithms, performance analysis of a program, optimizing the performance of a parallel program.

Tour of the Parallel Processing Lab. Introduction to the setup. Initiating a login session to use the parallel machine. Introduction to the programming environment (Trollius and Logical Systems C).

Students turn in a preliminary report with the solution design for their project.

Weeks 6 and 7 Taxonomy of parallel computers. Vector Processors, SIMD computers, MIMD computers, suitability of machines for different problem types.

Student presentations of their problem-statement and solution design.

Week 8 The distributed memory, message passing paradigm versus the shared memory paradigm.

Writing programs in the Trollius environment.

Weeks 9 and 10 Software issues: parallelizing compilers, parallel debuggers, integrated program development environments, graphical user interfaces and their effects on the efficiency of program development.

Analysis of Trollius with respect to the above software issues.

Week 11 Scientific Visualization using tools like Mathematica, Matlab, XImage, XCollage, GVL.

Week 12 and 13 Hardware issues: interconnection networks, parallel computer architecture issues and their effect on program development.

Week 14 End semester in-class presentation of project results.

Submission of project report.

4 Conclusion

This paper describes a course that is being developed at the Colorado School of Mines. The goal of the new course is to instruct undergraduate science and engineering students in parallel computing. It is hoped that this knowledge will make them more valuable contributors to industry (one of the students from the Spring '93 class went on to do a summer internship with Mobil Oil, where she wrote programs on the CM-5). For those students who pursue a graduate education, this course will provide instruction in an extremely useful research tool (one of the student projects formed the basis of an article at an international conference [13]). In addition, it is hoped that this course will prove to be a model that can be used by other schools that are dedicated to enhancing undergraduate education in science and engineering.

References

[1] Al Brady. Mackay School of Mines, University of Nevada. Personal Communication.

[2] Greg D. Burns. Ohio Supercomputing Center. Personal Communication.

[3] R. M. Butler, R. E. Eggen, and S. R. Wallace. Introducing parallel processing at the undergraduate level. *SIGCSE Bulletin*, 20(1), February 1988.

[4] A. L. Fisher and T. Gross. Teaching the programming of parallel computers. *SIGCSE Bulletin*, 23(1), March 1991.

[5] L. Fosdick, E. Jessup, and C. Schauble. Course material for High Performance Scientific Computing. Material obtainable by anonymous ftp from cs.colorado.edu.

[6] A. Ghafarian. An experimental approach to a course on parallel and distributed algorithms. *SIGCSE Bulletin*, 23(1), March 1991.

[7] J. Hartman and D. Sanders. Teaching a course in parallel processing with limited resources. *SIGCSE Bulletin*, 23(1), March 1991.

[8] B. P. Hillam. Integrating an array processor into a 'hands-on' computer science curriculum. *SIGCSE Bulletin*, 22(2), June 1990.

[9] D. C. Hyde. A parallel processing course for undergraduates. *SIGCSE Bulletin*, 21(1), February 1989.

[10] D. J. John. Integration of parallel computation into introductory computer science. *SIGCSE Bulletin*, 24(1), March 1992.

[11] E. Luque, R. Suppi, and J. Sorribes. A quantitative approach for parallel computing teaching. *SIGCSE Bulletin*, 24(1), March 1992.

[12] M. J. Meredith. Introducing parallel computing into the undergraduate computer science curriculum: A Progress Report. *SIGCSE Bulletin*, 24(1), March 1992.

[13] Manavendra Misra and Terry Nichols. Computation of 2-d Wavelet Transforms on the Connection Machine-2. In Claude Girault, editor, *Applications in Parallel and Distributed Computing*, number A-44 in IFIP Transactions, pages 3–12. North Holland, 1994.

[14] C. Nevison. Colgate University. Personal Communication.

[15] C. Nevison. An undergraduate parallel processing laboratory. *SIGCSE Bulletin*, 20(1), February 1988.

[16] Nan C. Schaller and Andrew Kitchen. Rochester Institute of Technology. Personal Communication.

[17] Nan C. Schaller and Andrew T. Kitchen. Experiences in parallel computing. In *Proceedings of the Parallel Computing in Education Workshop, Miskolc, Hungary*, March 1993.

[18] G. S. Stiles. Utah State University. Personal Communication.

[19] D. Thiebaut. Smith College. Personal Communication.

A Evaluation Sheet

The following grading sheet lists the criteria based on which the students were assigned a grade for the class. These criteria were handed out to the students towards the end of the semester.

Scientific Supercomputing, Spring 1993
Grading Sheet

Technical Quality of Project:

90%

1. Motivation for choice of project–why did you think that this was an appropriate supercomputing project?
2. Motivation for choosing the particular target machine that you worked on. What was it about your project that made you think that this was the best machine to solve the problem on? Do you have an understanding of the special features and the functioning of this machine?
3. An understanding of the various supercomputing issues and tradeoffs. What kind of benefits were obtained by using the particular supercomputer that was used for the project? Was there an adequate explanation of why these benefits came about (e.g. if there is a significant speedup, why did it come about–if there wasn't, why not?). What kind of data mapping was used? What kind of special functions/functional units on the machine were used for this application? Why?
4. Amount of work appropriate for a semester project?
5. What was the quality of results achieved (with respect to supercomputing issues!)? How much (of the supercomputing issues covered in the class) was learned from the course?
6. Does the project report describe the relevant work done for the project adequately?

Communication of project results:

10%

1. Quality of in-class presentation.
2. Quality of project report.