Experiences in Parallel/Distributed Computing for Undergraduate Computer Science Majors

Jo Ann Parikh
Southern Connecticut State University

Abstract

The increasing prevalence of multiprocessor and distributed systems in modern society is making it imperative to introduce the underlying principles of parallel/distributed computing to students at the undergraduate level. In order to meet the needs of our students for training in this critical area, the Computer Science Department at Southern Connecticut State University (SCSU) is currently in the process of implementing a curricular and laboratory development project that integrates key concepts and practical experiences in parallel computing throughout the undergraduate curriculum. The goal of this project is to build a strong foundation in parallel computing which would optionally culminate in advanced, senior-level specialized courses in parallel computing and/or senior research projects.

This paper describes the laboratory facility we developed to support instruction in parallel and distributed computing and the parallel computing modules which were incorporated into three of our core undergraduate courses: data structures, operating systems, and programming languages. The laboratory facility enables us to provide our students with "hands-on" experiences in shared memory, distributed memory, and network parallelism. The modules and laboratory exercises give students the opportunity to experiment with a wide array of software and hardware environments and to gain a systematic exposure to the principles and techniques of parallel programming.

1. Introduction

The major objective of the curricular and laboratory development project in parallel/distributed computing at Southern Connecticut State University is to make every computer science major aware of the basic hardware models, fundamental concepts, and programming languages and tools relevant to parallel/distributed computing. Support for this project from the National Science Foundation has enabled the Computer Science Department to obtain dedicated parallel computer systems and proprietary parallel computing software. The computer network provides students with the ability to experiment with various parallel computing models both locally and remotely.

At the present time, parallel/distributed computing laboratories have been designed and assigned to students in three courses: data structures, operating systems, and programming languages. Typically, students obtain their first experience in parallel computing in the data structures course. This is followed by a more in-depth experience in the operating systems course in which students

explore various pitfalls of parallel computing. The programming languages assignment concentrates on parallel computing languages, constructs, and tools.

2. Parallel/Distributed Computing System

The parallel/distributed computing system consists of a heterogeneous network of UNIX-based workstations supported by a Sun SPARCserver 1000 running Solaris 2.5 and a Digital DECstation 5000 running Ultrix 4.2. Attached to the Sun SPARCserver 1000 is a dedicated parallel processing transputer system from Parsytec, Inc. The transputer system is configured basically as a 4x4 mesh with supplemental connections as shown in Figure 1(a) to support various topologies such as pipelines, rings, hypercubes and trees in addition to array or mesh topologies. The system can be configured in software as a 15-node binary tree using, for example, the node configuration shown in Figure 1(b). There are several excellent books on transputers suitable for classroom use which provide detailed descriptions of the hardware design of transputer systems and offer suggestions for software configurations for different applications. [1,2]

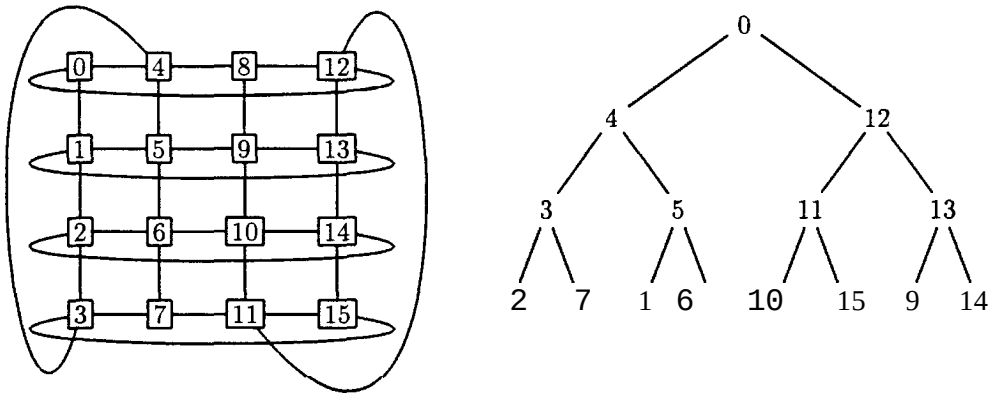

Figure 1: (a) Transputer Hardware Configuration
(b) An Embedded Binary Tree

The network is also used by the students to access remote parallel and distributed computing resources such as the IBM Scalable POWERparallel processor SP2 located at the Cornell Theory Center (CTC). The SP2 at CTC is a distributed memory machine with 512 nodes (where each node consists of a standard POWER2 architecture RS/6000 processor) connected by a high-performance switch. The SP2 was used by students in the programming languages course.

3. Programming Languages and Tools

Proprietary parallel computing software packages available on the local network are C-Linda from Scientific Computing Associates (which was installed on both the Sun server and the DEC server) and the INMOS C Toolset for the transputer system. Linda is used in the laboratory assignments to illustrate high-level parallel software for a virtual shared memory model. The INMOS software for the transputer system, in contrast to the Linda software, offers the potential to understand and experiment with the message-passing model at a level which is closer to the actual hardware.

Several software packages for parallel and distributed computing available through the Internet have also been installed on the local network. Packages used in the courses include PVM (Parallel Virtual Machine) from Oak Ridge National Laboratory, Parallaxis from the University of Stuttgart, and Tcl/Tk/Tcl-DP. PVM and Tcl-DP (Tk extensions for Distributed Programming) provided support for laboratory assignments in distributed memory parallelism and Parallaxis for assignments in data parallel programming.

4. Laboratory Assignments

This section contains a description of sample assignments which have been given to students in courses on data structures, operating systems, and programming languages. These projects were designed to help the students internalize important concepts in parallel and distributed computing and gain practical experience in the implementation of these concepts. The laboratories emphasize an exploratory, discovery-based learning approach.

4.1 Data Structures Assignment

The data structures assignment uses the Parsytec transputer system to introduce students to the concepts of speedup, efficiency, communication overhead, and algorithm scalability. In the first part of the assignment, students are supplied with a parallel mergesort code. They compute performance measurements for different problem sizes and numbers of processors. Students are often surprised by the counterintuitive result that speedup does not always increase linearly as more processors are added. In seeking to understand the results and write their conclusions, students begin to think about communication overhead and algorithm scalability.

In the second part of the assignment, students have to replace the call to the C library quicksort routine which is used to sort the numbers at each node with 1.) a student-coded recursive quicksort function and 2.) a student-coded nonrecursive quicksort function. Timings and performance measurements are then repeated using their own code. As they look through the code to determine where and how modifications are to be made, they gain familiarity with message-passing syntax and a deeper understanding of the message-passing model.

4.2 Operating Systems Assignments

Students in the operating systems course encounter parallel and distributed computing in three

different assignments. In the first assignment, they learn how to create multiple processes and threads and manipulate semaphores using standard C code and a standard C compiler. In the second assignment, they implement the producer-consumer algorithm on the transputer system using both a pipeline topology and a binary tree topology. In the third assignment, they explore page-based distributed shared memory using the scripting language Tcl [6] with the Tk graphics toolkit and Tcl extensions (Tcl-DP) for distributed programming.

The first assignment in the course provides an introduction to process synchronization and communication in the familiar environment of the C programming language. In the first part of the assignment, students create a child process using a "fork" system call which computes and outputs the partial sums of the numbers in an array while the parent computes and outputs the partial products. The students observe and are often surprised that the output may be interleaved. In the second part of this assignment, students implement the single producer, single consumer version of the producer-consumer problem using system calls to create a consumer thread and a producer thread and to manipulate semaphores. Pseudocode for the solution is in their operating systems textbook 8. This part of the assignment focuses on the topics of process synchronization and of mutual exclusion of shared variables.

The second assignment builds upon the producer-consumer problem of the first assignment using the transputer system. Students explore the concept of buffered communications using 1.) queues and semaphores in a message-passing system and 2.) a simple buffer process which runs on a separate transputer. They are supplied with code that implements a solution to the producer-consumer problem using unbuffered communication. This assignment was adapted from the producer-consumer laboratory in "Laboratories for Parallel Computing" [4] and code supplied to the students was implemented for our transputer system in the C language.

The third assignment explored issues in implementation of a page-based distributed shared memory system. Students were asked to read the material on distributed shared memory in the distributed operating systems textbook by Andrew Tanenbaum 9. We installed software for Tcl/Tk/Tcl-DP on our network and made available to the students code from Virginia Tech (http: //ei.cs.vt.edu/) which implemented in Tcl/Tk/Tcl-DP the central manager algorithm for distributed memory. The students were asked to analyze the algorithm, comment on possible problems such as deadlock, and modify the code by incorporate ing the least recently used (LRU) page replacement algorithm into the code in the context of a distributed shared memory system.

4.3 Programming Languages Assignment

In the programming languages course, students investigated different software environments for implementation of a parallel computation of the Mandelbrot set. Students were supplied with a sequential version of the algorithm and a program for graphical display of the output array. All the students in the class had accounts on the IBM SP2 at Cornell Theory Center (CTC).

The first part of the assignment was to solve the problem of parallel computation of the Mandelbrot set using PVMe (PVM enhanced for the SP2). Tutorials and helpful "step-by-step" exercises developed by the CTC staff were available for the students in addition to the user's guide [5]

and on-line documentation. Simple examples of PVM code for a "hello, world" program were demonstrated in class as well as different ways to access the SP2 computer from our local environment.

The second part of the assignment was to use Linda on the SP2 to solve the same problem. The concept of "tuple-space" was explained in class and basic Linda operations demonstrated for several simple examples. Materials available for the students were the Linda user's guide-', on-line documentation, and several example programs developed by staff from CTC and from Scientific Computing Associates .

5. Conclusions

The parallel and distributed computing laboratory assignments evoked a high degree of student response and student interest. In particular, some of the students in the data structures class prepared exceptionally outstanding reports with graphs and figures in addition to code explaining and analyzing the concepts of parallel computing. Although the data structures class should precede the operating systems and the programming languages course, students in the upper-level courses had not been introduced previously to parallel computing and, in many cases, were exposed to the concepts for the first time. In the operating systems class, the graphical demonstrations of paging provided by the Tcl assignment were extremely helpful in explaining the concepts of distributed shared memory but learning the syntax of Tcl and Tk in order to modify the program proved to be too time-consuming. In future, the Tcl assignment will be given primarily as a demonstration and analysis assignment with students asked to analyze the paging behavior of the distributed shared memory system for various different scenarios. The other operating systems assignments gave the students a very practical experience with the associated concepts that they were studying in their textbook. The programming languages assignment provided the students with the experience of actually working with distributed programs on a system in which timing is not affected by slow ethernet connections between workstations.

With the assistance of the National Science Foundation and the Cornell Theory Center, SCSU was able for the first time in 1995 to implement extensive curriculum modifications designed to offer out students a broad exposure to parallel and distributed computing in a wide variety of courses. In the future, students in upper-level classes will have the benefit of prior exposure to parallel and distributed computing architectures, paradigms, and issues which will facilitate a more in-depth approach in upper-level courses. The Computer Science Department plans to introduce a senior-level elective course in parallel and distributed computing within the next few years. Completion of our new program should prepare students for the challenges and opportunities posed by the rapid advances in networking technology and increasing prevalence of multiprocessor architectures.

Acknowledgments

The author gratefully acknowledges the support of the National Science Foundation Instrumentation and Laboratory Improvement Program, grant number DUE-9351364, for the SCSU Parallel Computing UNIX Laboratory. The author would also like to thank the Cornell Theory Center for educational accounts for the Programming Languages Course. The parallel processing laboratories using the SP2 were conducted using the resources of the Cornell Theory Center, which receives major funding from the National Science Foundation (NSF) and New York State, with additional support

from the Advanced Research Projects (ARPA), the National Center for Research Resources at the National Institutes of Health (NIH), IBM Corporation, and other members of the center's Corporate Partnership Program.

References

1. U. de Carlini and U. Villano, *Transputers and Parallel Architectures: Message-passing Distributed Systems,* Ellis Horwood Limited, 1991.

2. M. E. C. Hull, D. Crookes, and P. J. Sweeney, *Parallel Processing: The Transputer* and *Its Applications,* Addison-Wesley, 1994.

3. L. Jin and L. Yang, "Parallel Solution of Hough Transform and Convolution Problems on Transputers Using Multimodal Architectures", *Transputer Research and Applications 5, 1992.*

4. C. H. Nevison, D. C. Hyde, G. M. Schneider, and P. T. Tymann, *Laboratories for Parallel Computing,* Jones and Bartlett Publishers, 1994.

5. Oak Ridge National Laboratory, *PVM User's Guide and Reference Manual,* May, 1993.

6. J. K. Ousterhout, *Tcl and the Tk Toolkit,* Addison-Wesley, 1994.

7. Scientific Computing Associates, Inc., *C-Linda User's Guide & Reference* Manual, April, 1993.

8. A. Silberschatz and P. B. Galvin, *Operating Systems Concepts, 4th cd.,* Addison-Wesley, 1994.

9. A. S. Tanenbaum, *Distributed Operating Systems,* Prentice-Hall, 1995.

JO ANN PARIKH is a professor of computer science at Southern Connecticut State University. Her interests include parallel and distributed computing, artificial intelligence, and object-oriented programming. Her address is Southern Connecticut State University, Computer Science Department, MO121, 501 Crescent Street, New Haven, CT 06515 (E-mail: parikh@scsu.ctstateu.edu).