Submitted to: *IEEE Transactions on Education*
August 20th, 1998

A State-Wide Senior Parallel Programming Course

Barry Wilkinson, *Senior Member, IEEE*, and Michael Allen

Abstract — **In this paper, we describe an undergraduate parallel programming course based upon networked workstations. The course is offered on the NC-REN (North Carolina Research and Education Network), a private telecommunications network which interconnects universities in North Carolina and provides multi-way, face-to-face video and audio communications. Course materials are described and made available in a new textbook. Topics are divided into basic techniques and applications. In addition, extensive home page materials are described.**

I. Introduction

Using more than one computer or processor simultaneously to solve a problem has long been recognized as a way to obtain greater execution speed [1][2]. There are several ways that multiple computers or processors can be constructed for this purpose. Parallel programming is programming such multicomputers or multiprocessors for simultaneous operation, and parallel processing is the generic term that covers both the hardware and the software aspects of multicomputer/processor systems. Most universities and four-year colleges offer parallel processing courses. Major university centers usually offer a sequence of parallel processing courses starting at the senior undergraduate/first year graduate level [3]. Four-year colleges may offer a single introductory course in the undergraduate curriculum or introduce some of the concepts of parallelism into existing computer science and engineering courses, such as into computer architecture or operating system courses [4]. At the University of North Carolina at Charlotte (UNCC), parallel processing was first introduced as a graduate topics course in the early 1990s to complement existing computer architecture and programming courses. Subsequently, a senior undergraduate parallel programming course was offered which concentrated upon programming aspects of using multiple computers. We also introduced parallel programming into our Freshman courses. In a companion paper, we describe our experiences in bringing parallel processing concepts into the Freshman year, while here we will describe our work at the senior level.

There are a number of significant differences between the undergraduate course we describe and courses elsewhere. First, the course is offered on our state-wide televideo network and received simultaneously by several North Carolina universities. This in itself generated some interesting aspects regarding faculty cooperation and parallel computing equipment at each site. Second, the course is centered around the use of networked workstations rather than supercomputers or simple transputers as done previously [5][6], to take advantage of widely available and accepted message passing and shared memory software tools.

A major problem for undergraduate parallel programming instructors has been the lack of a suitable textbook. Almost all parallel programming course textbooks assume graduate-level readers. We solved this problem by writing a 430-page textbook for our course, the first such textbook concentrating upon undergraduate parallel programming [7]. We have also provided extensive web-based materials for both students and instructors.

II. Computing Platform

There are generally five choices of a computing platform for undergraduate parallel programming courses, namely:

- Multiprocessor simulator/emulator
- Transputer system
- Supercomputer, located at a supercomputer center and accessed remotely
- High performance multiprocessor system, purchased and maintained by the institution
- Workstation cluster (sometimes called NOWs, network of workstations, or COWs, cluster of workstations)

Several multiprocessor simulators/emulators exist. Sometimes, they model a real system and sometimes they represent an imaginary system. Lester provided a simulator with his book [8] which enables parallel PASCAL-like programs to be written and their performance evaluated. We have used Lester's simulator, but the base language PASCAL became unattractive when our programming language courses moved onto C and C++. We did re-write Lester's simulator for C-like programs, but in any event, a real system is much more desirable than a simulator.

Transputers are inexpensive processors designed specifically for multiprocessor systems. Undergraduate parallel processing courses based upon transputers have been described in [6] and [9]. We, like many others, did use transputers in introductory parallel programming courses in the early 1990s because of the very lost cost of transputers. Transputer boards are available for PCs and Sun workstations. We did put together several transputer configurations using PCs and SUN workstations as host systems. Our experiences with transputers were relatively positive for some programs, but less so for significant programming projects. We abandoned our transputers by 1993. Our main reason was that we prefer to teach a widely-used language and processor for the greatest benefit of students when they leave the institution. To use the special feature of transputers, such as instruction-level parallelism and automatic hardware process scheduling, requires the programming language occam to be used. This language is not applicable to any other platform. (It is possible to write programs in C but this would not take full advantage of the transputer's special features.) Our transputer performance was also lacking compared to the ever improving PCs.

Senior undergraduate/graduate parallel processing courses based upon supercomputers have been described in [5]. Using a supercomputer located at a supercomputer center is certainly a possibility for us as a supercomputer center is located nearby (North Carolina supercomputer center in Raleigh). However, there are several reasons for not doing so. The technicalities of supercomputers may be too much for an introductory undergraduate course and are a digression from our desire to teach parallel algorithms. Supercomputers, such as Cray vector computers, often require one to write programs in programming languages that support vector operations if the best performance is to be achieved. It requires one to study in detail optimizations applicable to the specific architecture. Optimizing compilers can do some of the work, but knowledge of the effects of data layout is often very important. The techniques are limited to the class of supercomputer and are not applicable to other types of computer systems.

Supercomputers are very expensive and access must be controlled carefully by the center. Researchers desiring access often have to write large research proposals to make a case for them to use the system. These researchers have to show that their problem will make efficient use of the system (say by measuring the MFLOPs achieved on sample programs). Supercomputer centers are not equipped to handle a large number of inexperienced undergraduate students. Their impact would be very significant on the ongoing research activities. However, supercomputer centers can handle a few small well organized graduate classes.

The next option is to purchase our own high performance multiprocessor system. Major universities may already have such systems for specific research projects. High performance multiprocessor systems may be based upon one of several architectures but the major disadvantage of this approach for most institutions is cost. For a smaller institutions, it is difficult to justify the equipment and maintenance costs for a single parallel processing course. Also, with the rapidly advancing technical progress towards faster systems, high performance multiprocessor systems quickly become obsolete. Later in the paper we will describe our efforts towards a unique locally shared memory multiprocessor cluster.

As mentioned earlier, we selected our final possibility, a workstation cluster. The advantages of workstation clusters for high performance computing is well documented [11]. Using a workstation cluster for teaching parallel programming is very attractive as workstations are readily available in all institutions and software tools for workstation clusters are now mature and freely available. Also, easily understood programming techniques can be used and portable programs written.

Initially, using workstations for parallel computing became interesting because networks of workstations existed for general purpose computing. Projects in the late 1980s to provide parallel programming software tools, most notably PVM, made the concept of using workstation clusters for parallel programming viable. Subsequently, the standard message passing library, MPI, was defined, establishing a workstation cluster as a mainstream approach to parallel computing. The literature on using networks of workstations for parallel computing is now growing as the importance of such computing platforms emerges. There have been a significant number of publications on using workstation clusters on various problems. With the advent of really powerful and inexpensive workstations and PCs, workstation clusters have assumed an even more important concept. Although workstation clusters will not have the performance of specially designed multiprocessor systems (notably because of the long latencies within the communication interface), the most important advantage that workstation clusters offer is ease of upgradability. This is now a very significant advantage. The performance of processors is increasing at about 50% per year. A workstation cluster solution to multiprocessor system design enables computers to be replaced and added to over time without having a major system redesign.

Because of the particular demands of parallel computing, we have chosen to create a dedicated workstation cluster for parallel programming rather than use general-purpose networked workstations. There are significant difficulties in relying upon a laboratory of workstations that are being used for other purposes. A particular early problem we had to face was not having remote login/rsh/rexec privileges on the systems because of administrative decisions. This prevents multiple general-purpose workstations being enrolled in PVM/MPI. A second problem we had to face was that the network is not dedicated to parallel programming and indeed the Pentiums could be switched between Windows NT and UNIX without the knowledge of remote users. We originally put together a cluster of eight existing SUN workstations just for the parallel programming classes, and are now planning a replacement cluster (see later).

III. Course Content

One of the first problems all parallel programming instructors have is to identify the course contents. The topics have not been standardized in the same fashion as, say, a data structures course where there are many textbooks. In parallel programming, there are very few truly undergraduate textbooks. We also wished to introduce parallel programming at lower undergraduate levels and were interested in a graded approach to the material, that is, to a set of material in which the first or introductory part is suitable for lower levels, including the Freshman year [12].

Using workstation clusters for parallel programming leads to message-passing programming. Our course uses PVM [13] and MPI [14] for message passing. However, we did not want to present all the algorithms specifically in PVM or MPI. We explain PVM/MPI in detail, but throughout we turn to pseudocode for explaining algorithms. This pseudocode is based upon C and uses a simplified notation for send/receive routines. Students can very easily convert the pseudocode to PVM or MPI, the two systems currently used at UNCC. We provide analysis of message-passing algorithms and discussion of latency issues.

We do also expose students to thread-based shared memory programming using Pthreads [15] although programs are run a single computer. The Pthreads library is chosen because it is a IEEE standard with many implementations and the students learn useful information.

We divided the material into two parts, Part I and Part II. Part I, “Basic Techniques,” describes basic parallel programming techniques using simple examples, briefly:

PART I Basic Techniques
- Parallel Computers

    - Types of Parallel Computers
    - Architectural Features
    - Potential for Increased Computational Speed

- Message-Passing Computing
    - Basics of Message-Passing Programming
    - Programming Options
    - Process Creation
    - Message-Passing Routines
    - Using Workstation Clusters
    - Software Tools, PVM, MPI
    - Evaluating Parallel Programs
    - Parallel Execution Time and Time Complexity
    - Debugging and Evaluating Parallel Programs

- Embarrassingly Parallel Computations
    - Embarrassingly Parallel Examples
        - Geometrical Transformations of Images
        - Mandelbrot Set
        - Monte Carlo Methods

- Partitioning and Divide-and-Conquer Strategies
    - Divide-and-Conquer Examples
        - Sorting Using Bucket Sort
        - Numerical Integration
        - *N*-Body Problem

- Pipelined Computations
    - Computing Platform for Pipelined Applications
    - Pipeline Program Examples
        - Adding Numbers
        - Sorting Numbers
        - Prime Number Generation
        - Solving a System of Linear Equations

- Synchronous Computations
    - Barrier and Implementations
    - Local Synchronization
    - Deadlock
    - Synchronized Computations
        - Data Parallel Computations
        - Synchronous Iteration
    - Synchronous Iteration Program Examples
        - Solving a System of Linear Equations by Iteration
        - Heat Distribution Problem
        - Cellular Automata

- Load Balancing and Termination Detection
    - Dynamic Load Balancing
    - Distributed Termination Detection Algorithms
    - Program Example – Shortest Path Problem

- Programming with Shared Memory
    - Specifying Parallelism
    - Sharing Data
    - Language Constructs for Parallelism
    - Dependency Analysis
    - Shared Data in Systems with Caches
    - Program Examples – UNIX, Pthreads, Java

Part II, “Algorithms and Applications,” describes algorithms for specific application areas, briefly:

PART II Algorithms and Applications
- Sorting Algorithms
    - Rank Sort
    - Bubble Sort and Odd-Even Transposition Sort
    - Two-Dimensional Sorting
    - Mergesort
    - Quicksort
    - Odd-Even Mergesort
    - Bitonic Mergesort

- Numerical Algorithms
    - Matrix Addition
    - Matrix and Matrix-Vector Multiplication
    - Relationship of Matrices to Linear Equations
    - Implementing Matrix Multiplication
        - Direct Implementation
        - Recursive Implementation
        - Mesh Implementation
    - Solving a System of Linear Equations Linear Equations
        - Gaussian Elimination – Parallel Implementation
    - Iterative Methods
        - Jacobi Iteration
        - Faster Convergence Methods

- Image Processing
    - Low-Level Image Processing
    - Smoothing, Sharpening, and Noise Reduction
        - Mean
        - Median
        - Weighted Masks
    - Edge Detection
        - Edge Detection Masks
    - The Hough Transform
    - Transformation into the Frequency Domain
        - Discrete Fourier Transform
        - Fast Fourier Transform

- Searching and Optimization
    - Branch-and-Bound Search
    - Genetic Algorithms
    - Hill Climbing
    - Example – Banking Application

Part I requires no specialized mathematical knowledge and could be used at lower levels in the curriculum. Part II does have some mathematical prerequisites (linear equations, partial different equations, matrices, etc.,) but only what would be expected of seniors. More details of the material can be found in [7]. A unique aspect is the concept of real-life problems. In addition to normal numerical problems, a large number of real-life problem are provided to apply the techniques given.

The material is decidedly practical in tone, which contrasts with most graduate level textbooks and courses. Theoretical aspects, such as PRAM and other models (BSP, LogP), are described but not used in any great extent.

The programming assignments begin with a simple familiarization assignment in which a working parallel program is provided that simply adds numbers together in parallel. Students set up their system environment, compile the program and obtain results. The students also have to modify the program to find the maximum value. At this stage only one workstation is used, but multiple workstations are used in subsequent assignments. This assignment only requires a few hours to do – the students simply read and follow the instructions on the home page. The concepts of creating the necessary makefile and other compiling matters are, of course, very familiar to most senior students.

The second assignment, which is still quite simple, illustrates an embarrassingly parallel program, the Mandelbrot computation. For this assignment, a working sequential Mandelbrot program is provided which includes X-window code for generating graphical output – the course does not assume any prior knowledge of graphics. The graphics code is in fact useful for many subsequent assignments. Students must parallelize the program, either by static process assignment or with dynamic load balancing as described in the lectures (the latter receives a higher grade).

The next assignments are of increasing difficulty and linked to the lecture materials. These assignments are varied on different occasions. One has been the astronomical *N*-body problem with graphical output (using graphics code taken from the Mandelbrot problem). Another assignment has been solving Laplace's equation to obtain the heat distribution in a room with a fireplace. Again, graphical output is required in the form of temperature contours. A follow-on assignment has involved solving the same problem by direct means (Gaussian elimination). It is useful to show different methods to solve the same problem and the speed implications of the different methods.

Load balancing is a key aspect of all the assignments, and timing information must be provided, usually by instrumenting the code with the time() system call. All assignments have "open-endedness" in that extra credit can be obtained by additional work.

IV. Teleclass Facility

North Carolina Research and Education Network

We have given our senior undergraduate parallel programming course on the NC-REN (North Carolina Research and Education Network) every year since 1996. NC-REN is a private telecommunications network interconnecting 19 university, medical center, research institution and graduate center sites in North Carolina. It became operational in 1985 and provides multi-way, face-to-face video and audio communications. Classroom "teleclass" facilities exist at each site, such as shown in Fig. 1. Each student is provided with a microphone, and several video cameras are used so that the instructor and students at each site can hear and see each other. All lectures were recorded so that students could view any lecture again or catch any they missed.

The course has been received at various times by North Carolina State University, University of North Carolina at Greensboro, University of North Carolina at Asheville, and University of North Carolina at Wilmington, in addition to the University of North Carolina at Charlotte. The success of this approach rests upon cooperation and support at each site. One faculty at each site has been responsible for making available the local parallel computing facility for their students.

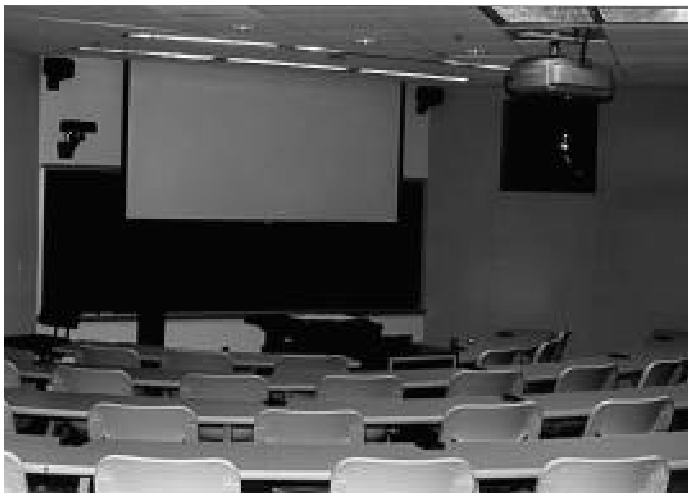


Fig. 1 Photo of one UNCC teleclass classroom facility.

All sites have networked workstations of various types that were suitable, but most did not ready access to a shared memory system. In that regard, Pthreads programs were simply run on a single processor system.

Guest Speakers

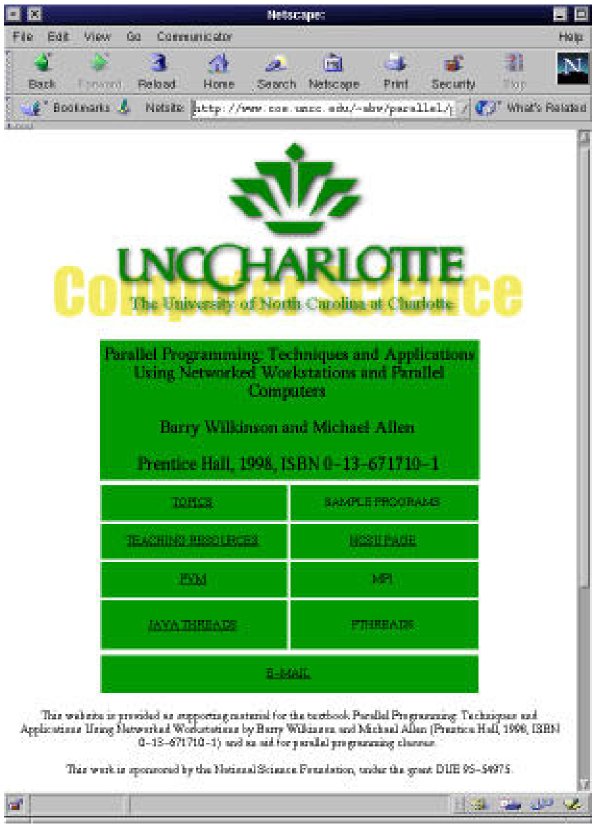
The televideo network offers an excellent way to reach many students. We have used the opportunity of this facility by inviting expert speakers to given presentations to the class and talk about some practical aspect of parallel programming. For example, in Fall 1996 Professor John Board of Duke University gave a presentation entitled "Networks of Workstations: The Plodding Workhorses of Parallel Computing" and also outlined modeling DNA. Professor Mladen Vouk of North Carolina State University gave a presentation on the current state of supercomputers and supercomputing conferences. Two graduate students made presentations of their parallel programming project dealing with parallel genetic algorithms, In Fall 1997, Professor Jan Prins of the University of North Carolina at Chapel Hill gave a presentation on Irregular Data Parallel Computations and Professor Harry Smith of the University of North Carolina at Wilmington gave a presentation on transputers. These presentations gave the material in the courses a greater significance.

V. Home Page Materials

With a distance learning environment as described here, it is essential to provide home page instructional materials that can be read or downloaded, such as sample programs mentioned, programs to be used in assignments etc. We have provided a home page (http://www.cs.uncc.edu/par_prog) as illustrated in Fig. 2 which includes:

- Online help for compiling PVM and MPI programs
- Chapter-by-chapter notes

- Special page for instructors which contains:
  Transparencies
  Sample programs/assignments
  On-line class questionnaire


Netscape:
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Netsite: http://www.cs.uncc.edu/~abw/parallel/
What's Related
UNCCHARLOTTE
Computer Science
The University of North Carolina at Charlotte

Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers

Barry Wilkinson and Michael Allen

Prentice Hall, 1998, ISBN 0-13-671710-1

TOPICS	SAMPLE PROGRAMS
TEACHING RESOURCES	NCSU PAGE
PVM	MPI
JAVA THREADS	PTHREADS
E-MAIL	

This website is provided as supporting material for the textbook Parallel Programming: Techniques and Applications Using Networked Workstations by Barry Wilkinson and Michael Allen (Prentice Hall, 1998, ISBN 0-13-671710-1) and as aid for parallel programming classes.

This work is sponsored by the National Science Foundation, under the grant DUE 95-54975.

Fig. 2 Home page

In a related activity at North Carolina State University, the sound of the lectures with images (slides) have been integrated on a home page there (http://renoir.csc.ncsu.edu/CSC495A).

VI. Current and Future Work

In this section, we will outline our on-going work connected with the parallel programming classes.

A. Improving Communication Latency

The fundament flaw in any workstation cluster is the weakness of the interconnect technology which is not increasing in performance at the same rate as the processors themselves. Fast 100 Mbit/sec Ethernet provides a simple and very inexpensive interconnect technology and is widely used for general-purpose workstation clusters. However, Ethernet suffers from inadequate communication bandwidth and more importantly, with typical software, excessive latency for parallel computing.

Improved bandwidth can be achieved by several well-known ways, for example using high speed switches and communication interfaces. An emerging possibility is the Gigabit Ethernet (IEEE 802.3z). These methods incur significant additional costs and there are cost/performance trade-offs.

Message latency is a vastly more important aspect than raw bandwidth. Latency is the major factor which can have a deleterious effect of performance of such programs. Excessive latency is often brought about by the layers of software imposed upon message-passing interfaces. A novel solution that we propose is using multiple physical interfaces. Given the low cost of commodity interfaces, it is now cost-effective to employ multiple interfaces; it is also possible to employ more than one connection between computers. This provides for hiding the message latency by overlapping message transmissions as illustrated in Fig. 3. However, there are significant software issues to resolve in this approach, since message-passing software is not designed for concurrently executed routines. The next section describes a multiprocessor system which has the potential for simultaneous operation of multiple interfaces within one system.

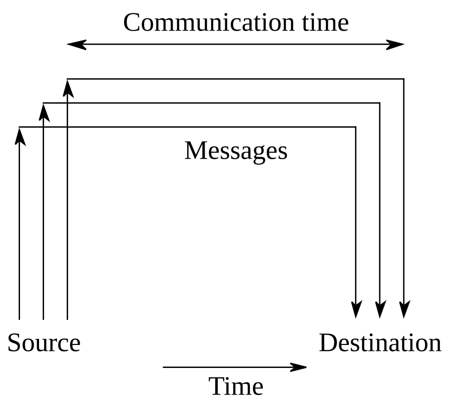


Fig. 3 Overlapping communication

B. Locally Shared Memory Cluster

Traditionally, two forms of multiprocessor system have existed, the shared memory system with a single address space, (whether the memory is physically distributed or not), and the multicomputer consisting of interconnected computers where each processor has its own address space. A workstation cluster is in the second category. Recent advances in system design have led to multiprocessor PCs. The Pentium II Xeon processor, for example, can be used for such systems in a shared memory configuration, with typically 2-8 processors. This has led the possibility of clustering multiprocessor systems. We call such clusters *locally shared memory systems*. Groups of processors share single but distinct address spaces. This form of cluster offers some very interesting programming possibilities which can be explored in parallel programming classes after message passing programming (using say MPI) and shared memory programming (say using Pthreads) have been covered.

A cluster of four quad-processor systems is shown in Fig. 4. With only four systems, it is quite feasible to provide full connectivity between the four 4-processor systems using three-port interfaces, as illustrated. The locally shared memory cluster offers the option of message-passing programming, shared memory programming, and a unique hybrid of using both techniques in a single program.

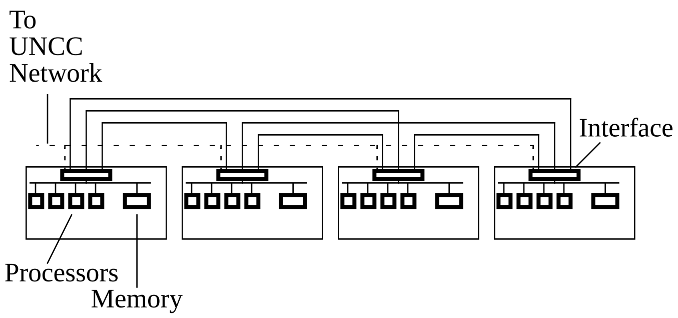


Fig. 4 Locally Shared Memory Cluster

VII. Conclusions

We have described a unique undergraduate parallel programming course which has been given on a televideo network to several North Carolina Universities. The course incorporates several new aspects partly because of its use of a televideo network. First, it is necessary for remote sites to have an adequate parallel computing platform. Networked workstations are ideal since every university has them. Second, very significant preparatory materials are necessary and we have produced both a textbook and web materials. The use of a televideo facility allowed experts from different sites to participate in the course and give presentations to undergraduates.

The material for our undergraduate parallel programming course is mostly based upon networked workstations and is practical in tone. Programming concepts introduced are centered around message passing. We also introduced thread-based programming into the class with at least one thread-based assignment. It is our view that thread-based programming will become more prevalent and should figure more prominently in undergraduate parallel programming courses. Though we use Pthreads, Java also has interesting possibilities for parallel programming.

Finally, we have proposed the use of multiple interconnects to ameliorate the effect of message latency, and the use of a cluster of shared memory computers – locally shared memory computers – a perfect computing platform for both message-passing, shared memory and a unique combination of both.

VIII. Acknowledgments

This work has been supported in part by the National Science Foundation, under grant DUE 9554975. It is a great pleasure to acknowledge Dr. M Mulder, program director at the National Science Foundation. We should like to thank the many students at the University of North Carolina at Charlotte, who helped us refine the material over the last few years. We owe a debt of gratitude to many other people including our guest speakers. Professor Vouk of North Carolina State University set up an impressive web page which included “real audio” of the lectures and “automatically-turning” slides.

References

[1] S. Gill, “Parallel Programming,” *The Computer Journal*, vol. 1, April, pp. 2–10, 1958.

[2] J. Holland, “A Universal Computer Capable of Executing an Arbitrary Number of Sub-programs Simultaneously,” *Proc. East Joint Computer Conference*, vol. 16, pp. 108–113, 1959.

[3] R. Miller, “The Status of Parallel Processing Education,” *Computer*, vol. 27, no. 8, Aug., pp. 40–43, 1994.

[4] C. H. Nevison, “Parallel Computing in the Undergraduate Curriculum,” *Computer*, vol. 28, no. 12, Dec., pp. 51–53, 1995

[5] J. A. Youssefi, and K. Zemoudeh, “A Course in Parallel Processing,” *IEEE Trans. Educ.*, vol. 40, no. 1, Feb., pp. 36–40, 1997.

[6] F. C. Berry, “An Undergraduate Parallel Processing Laboratory,” *IEEE Trans. Educ.*, vol. 38, no. 4, Nov., pp. 306–311, 1995.

[7] B. Wilkinson and M. Allen, *Parallel Programming: Techniques and Application Using Networked Workstations and Parallel Computers*. Upper Saddle River, NJ: Prentice Hall, 1998.

[8] B. Lester, *The Art of Parallel Programming*. Englewood Cliffs, NJ: Prentice Hall, 1993.

[9] T. Hintz, “Introducing Undergraduates to Parallel Processing,” *IEEE Trans. Educ.*, vol. 36, no. 1, Feb., pp. 210–213, 1993.

[10] J. Avila, “A Bread-First Approach to Parallel Processing for Undergraduates,” *Conf. Parallel Computing for Undergraduates*, Colgate University, June 22–24, 1994.

[11] T. E. Anderson, D. E. Culler, D. A. Patterson and the NOW team, “A Case for NOW (Network of Workstations),” *IEEE Micro*, vol. 15, no. 1, Feb., pp. 54–64, 1995.

[12] M. Allen, B. Wilkinson, and J. Alley, “Parallel Programming for the Millennium: Integration Throughout the Undergraduate Curriculum,” *Second Forum on Parallel Computing Curricula*, co-located with *Symposium on Parallel Algorithms and Architectures ‘97*, June 22nd, 1997.

[13] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, *PVM3 User’s Guide and Reference Manual*. Oak Ridge National Laboratory: Tennessee, 1994.

[14] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI Portable Parallel Programming with the Message-Passing Interface*. Cambridge, Massachusetts: The MIT Press, 1994.

[15] B. Nichols, D. Buttlar, and J. P. Farrell, *Pthreads Programming*. Sebastopol, California: O’Reilly & Associates, 1996.