Int. J. Engng Ed. Vol. 10, No. 3, pp. 291–298, 1994
Printed in Great Britain.

0742–0269/91 $3.00+0.00

Undergraduate Education and Research in High-performance Computing*

LYLE N. LONG
JESSE L. BARLOW
LOUIS F. CONSTABLE
KEVIN M. MOROONEY

Department of Aerospace Engineering, 233 Hammond Bldg., Penn State University, University Park, PA 16802, USA

This paper describes a project to teach high-performance computing software and algorithm development to undergraduate students. The project was funded by the National Science Foundation and Penn State University. The funds allowed us to purchase a massively parallel Connection Machine CM-200 computer. A new course called Parallel Processing was introduced and jointly taught by Aerospace Engineering and Computer Science. The course included topics such as interprocessor communication, programming languages, parallel algorithms, parallel efficiency and data structures. In addition to the course, undergraduates have also been writing senior theses in high-performance computing. Since the improvements in performance of vector-processing supercomputers is leveling off, future supercomputers will be massively parallel. Since these machines are quite different to classical serial computers, it is important to introduce massively parallel computing early in the students' careers. For most students, the projects described herein were their first exposure to high-performance computing. While we have numerous graduate students working in parallel processing, this paper will primarily address the undergraduate project.

INTRODUCTION

THE PROJECT described herein is a joint one between the Computer Science Department and the Aerospace Engineering Department of the Pennsylvania State University. The goal of this project is to introduce undergraduates to high-performance computing and parallel processing. The interdepartmental nature of the project means that a large and diverse student body has access to the material. In addition, it fosters interdisciplinary teaching and undergraduate research, between Computer Science and Engineering—for both students and faculty.

The Pennsylvania State University is located in central Pennsylvania and has approximately 32,000 undergraduate students and 6500 graduate students. In 1991 there were 8293, 1131 and 463 degrees conferred at the bachelor, master and doctoral levels, respectively. The Computer Science Department has roughly 200 undergraduate students. The areas of study for these students are mainly in software; in particular, compilers, algorithms, data structures, artificial intelligence, computer languages, operating systems and theoretical computer science. The Aerospace Engineering Department has roughly 150 students at the junior and senior levels. These students study the traditional areas of aerospace engineering: fluid dynamics, structures, propulsion, dynamics and control.

The teaching projects described herein address parallel processing and scalable algorithms for 'grand challenge' problems. The grand challenges require that the supercomputer industry aim for the teraflop computing power that D. Allan Bromley [1, see also 2], technical advisor to the President, asked for by 1996. Traditional supercomputing power, usually associated with large multi-CPU machines with vector processing capabilities, is fast approaching the physical limits imposed by the speed of light and the material properties of the semiconductors used in those machines. During the last several years, parallel processing has emerged as the technology that will provide teraflop computing power. The goal of 1 teraflop will be reached within a few years. There are a number of grand challenge problems, such as meteorology, turbulence, aircraft design, combustion, molecular dynamics, vision and cognition, protein folding, pharmaceutical design, neutron transport, database management, and the human genome, that will be forever changed by this technology.

Since Penn State is a large research university (ranked 11th nationally in terms of sponsored research), the undergraduate students are accustomed to having research topics introduced in their upper division courses. In addition, undergraduate research is becoming more and more common at

* Paper accepted 12 September 1993.

Penn State. Our undergraduates take courses in topics as advanced as turbulent flow, computational fluid dynamics, composite material processing, spacecraft control, artificial intelligence, computer architecture, compiler technology and matrix computations.

With the rapid progress made in the field of massively parallel processing and the impact it is having (and will continue to have) on almost all the above technologies, it is important to provide undergraduates with access to parallel computers. While the undergraduates at Penn State have good access to serial computers, access to parallel computers has been rare. Courses taught in both engineering and computer science have introduced parallel computing concepts, but in-depth knowledge and experience is not possible without access to the computers. This project tries to provide access to state-of-the-art parallel computers for undergraduate students.

The primary impact of parallel computing upon numerical analysis has been in matrix computations [3], the solution of partial differential equations [4, 5] and particle-based simulation methods [6]. These problems have a great deal of exploitable parallelism. It is important for instructional reasons to be able to demonstrate the ability to speed up large-scale computations on parallel machines. In addition, many of the language tools and constructs are not available on traditional computers.

As an example, in matrix computations, the simplest matrix operations, such as matrix–vector and matrix–matrix operations can be performed at near peak megaflop rates on most high-performance architecture. For this reason, the Linear Algebra Package (LAPACK) [3] was built around such operations. For shared-memory machines such as the Alliant FX/8, the Cray-X-MP and the Titan, many of these operations were incorporated into the Basic Linear Algebra Subroutines (BLAS) (levels 1–3). The manufacturers of these machines were asked to implement the BLAS in assembly language or in the fastest way that they could be implemented. The LAPACK modules for solving eigenvalue problems, systems of linear equations and least-squares problems were built on top of the BLAS. The result was that the LAPACK routines ran almost as fast as those written entirely in assembly language. It is important for students to understand this philosophy of software development. That is, that one should develop software that is as portable as possible while still being efficient. However, at the same time, it should be understood that new high-performance architectures make it more difficult to develop software that can be moved from machine to machine.

In addition, students should know how to program numerical algorithms effectively and how to use performance measurements. A classical example is the solution of a traditional system of linear equations. This problem is an important subproblem that arises in problems ranging from time-dependent partial differential equations (by ADI methods) to the solution of eigenvalue problems. If the resulting system does not require pivoting for Gaussian elimination, then it can be solved in $O(n)$ operations where n is the dimension of the system. Moreover, the algorithm is simple and easy to program on sequential computers. Gaussian elimination is, however, a serial algorithm that is not easily ported to parallel computers. We know that algorithms such as cyclic reduction or recursive doubling can solve the resulting system in $O(\log n)$ operations using $O(n)$ processors. These methods, however, are not stable for as large a class of matrices as Gaussian elimination. It is useful for students to see these techniques, to be able to experiment with them on a real parallel computer and to understand their limitations.

PARALLEL COMPUTERS

Figure 1 shows several of the more powerful existing and proposed computers. Their performance range is shown in terms of speed per processor and number of processors. The peak speeds shown have not been completely demonstrated for some of these computers, and, in most cases, a machine with the maximum number of allowable processors has not been assembled. For example, the 1024-processor CM-5s at Los Alamos National Laboratory and the National Security Agency (NSA) are the largest CM-5s that have been built. These machines have *sustained* four times the *peak* speed of a 16-processor Cray C-90. Massively parallel computers are the most effective path to teraflop computing.

One clear trend is that all the high-performance, massively parallel computers will take advantage of CMOS chips found in RISC workstations. There are vendors who will be using the Sparc, Intel I-860, DEC Alpha and IBM RISC chips. This is the most cost-effective approach to teraflop computing, since it takes advantage of mass-produced

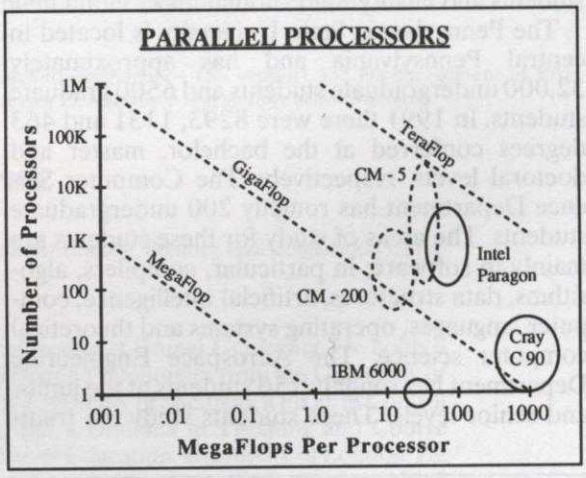

Fig. 1. Parallel processing computers.

chips. An important design point for these new parallel systems is that they be scalable. The challenge for scientists and engineers is to develop applications that will take advantage of these new teraflop machines as quickly as possible.

There are several massively parallel computers (i.e. more than 1000 processors) on the market. These computers fall into two categories: multiple-instruction, multiple-data (MIMD) or single-instruction, multiple-data (SIMD). The MIMD machines (e.g. Intel Paragon, Cray T3D, IBM SP2, TMC CM-5, and (nCUBE/2) have processors that can all be performing *different* operations simultaneously. The SIMD machines (e.g. TMC CM-200 and MasPar MP-2) have processors that all perform the *same* operation simultaneously. MasPar is essentially the only remaining manufacturer of general-purpose SIMD computers.

Cray, IBM, Intel, MasPar, Meiko and Thinking Machines have all announced new machines that show promise, but they are all quite different and a clear winner has not emerged. Massively parallel computers are still evolving rapidly. When they first appeared, the main types of parallel computers were either SIMD or MIMD designs. The processors on all the proposed teraflop machines are quite similar now and the hardware (SIMD vs. MIMD) debate is essentially over, since MIMD is the dominant architecture. There are significant differences in networks and memory access schemes however.

The issue today involves the most effective programming style. In particular, the question today seems to be whether or not the data-parallel approach (e.g. High Performance FORTRAN and FORTRAN 90) is superior to the message-passing approach (e.g. PVM [7]) for a particular application. Most likely, both approaches will succeed, since they both have their advantages and disadvantages. We have significant experience using Connection Machine computers and have been quite successful with the data-parallel approach. In addition, all of our CM-200 codes run very effectively on the CM-5. There are algorithms, however, that are difficult to program using the data-parallel approach, for example, particle methods or Monte Carlo methods.

There have also been advances in software technologies and network reliability which provides a useful, inexpensive context for developing parallel applications that will be prepared for the next generation supercomputers. Loosely coupled clusters of high-function workstations have been shown to be useful parallel computer engines. Software like Parallel Virtual Machine (Oak Ridge National Laboratory), Express FORTRAN (Parasoft Corporation), Control Process Software (Fermi Laboratory) and Linda (Scientific Computing Associates), all provide the mechanisms for using clusters of workstations as parallel machines. The power of these software solutions is that they all operate with existing communication links between the machines. The bandwidth of these links, however, can be quite low and the message latencies are quite large. A cluster of workstations in a department or across a college campus can participate in the solution of a single computational problem using Ethernet, Token Ring, etc., as the communication vehicle. Most of the above software packages also allow for development of applications in a heterogeneous vendor and architectural environment. Not only can we develop programs which will scale for use on new parallel supercomputers, but we can develop programs which can take advantage of several different architectures on the campus network. In the future people will rely on the Message Passing Interface (MPI).

PENN STATE COMPUTER FACILITIES

Students at Penn State have access to a wide range of computer facilities, as shown below:

- 3-processor IBM ES9000 (Center for Academic Computing);
- 48-processor IBM SP-2 (Center for Academic Computing);
- 2-processor Cray YMP/2E (Earth Systems Science Center);
- 2048-processor CM-200 (Aerospace and Computer Science);
- 16-processor nCUBE/2 (Electrical and Computer Engineering);
- Workstation clusters: IBM RS/6000 and Sun SparcStations;
- Workstations: IBM, Silicon Graphics and Sun;
- Microcomputers (Intel and Apple).

This is an environment smaller in scale but identical in offering to the environment provided at most supercomputer centers and national labs. The university has an IBM ES9000, a three-processor vector computer with 1 Gbyte of main memory and 190 Gbytes of disk space. We also just received an IBM SP-2, which is a tightly coupled array of RS/6000 computers with a high-speed switch and 6 Gbytes of main memory. The College of Engineering has a large number of workstations (IBM, SGI and Sun) and personal computers, which are available to undergraduates. The Department of Computer Science has a laboratory with approximately 30 Sun workstations. All of the above equipment is networked together on a fiber-optic backbone.

The main computers used for this project were the massively parallel Connection Machine (CM-200) computer and the workstations. The CM-200 has a Sun SparcStation 2 front-end. The students did not log onto the front-end directly, but used the workstation labs around campus (SGI, IBM and Sun). Several labs and the front-end computer ran the Distributed Queuing Systems (DQS) developed by Florida State University [7] and the Andrew File System (AFS) [8]. The students compiled and ran their programs using DQS. Since

the compilers only run on Sun workstations, the IBM computers would send the compile tasks to various Sun workstations. The students essentially used the SGI computers as X-terminals, since SGI did not support AFS. We did not activate time-sharing on the CM-200, but next time we will, since without it small jobs often sit in queues behind large jobs.

In addition to the mainframe and the workstation/computer laboratories, Penn State has also computerized several classrooms. These high-technology classrooms use a variety of computers and projection devices. The simplest systems use microcomputers with overhead projectors. The most sophisticated classroom has an IBM RS/6000 model 370 built into a podium with a ceiling-mounted, high-resolution Esprit projector. This classroom is in the College of Engineering and accommodates 90 students. It also has a sound system and a VCR that is connected to the projection system. The RS/6000 has 128 Mbytes of memory and 5 Gbytes of disk space, so advanced engineering packages such as NASTRAN, Patran, Mathematica, MATLAB and IDEAS can be run in the classroom. These classrooms are revolutionizing teaching at Penn State. They not only permit faculty to present their existing lectures more effectively, but also allow a teacher to discuss topics that are impossible to cover using traditional chalkboards and transparencies.

Penn State's Center for Academic Computing (CAC) has three full-time staff and two graduate students dedicated to the exploration of high-performance computing solutions for Penn State faculty, staff and graduate students. The staff provide expertise in all phases of high-performance computing: parallel processing, vector processing, general code optimization and application software support. Additionally, the group is an active member of the Cornell National Supercomputer Facility's Smart Node Program, the Pittsburgh Supercomputing Center, and National Center for Supercomputing Application's Academic Affiliates programs. The CAC staff were invaluable in setting up the computers for this project and in giving guest lectures.

NEW UNDERGRADUATE COURSE IN PARALLEL COMPUTING

Many of the issues discussed in the previous sections were considered in a new three-credit undergraduate course in parallel processing for scientific computing (Spring semester, 1993). The classroom was equipped with a Silicon Graphics Indigo workstation built into the podium and a high-resolution, ceiling-mounted Esprit projector. This computer is networked and available through Internet. The instructor can run programs locally on the SGI or connect to remote computers, such as the CM-200, across campus. Normally we use X-Windows and we can have several windows open at a time, each one can be running on a different computer.

Having a powerful workstation built into the podium opens up entirely new options for the instructor. Instead of using a blackboard or overhead projector, we can actually run live demonstrations, edit code, show color graphics, etc. At one point during class, we logged into our CM-200 on campus and a CM-5 at Thinking Machines Corporation (in Cambridge, Massachusetts) from the classroom computer. We ran the same code on each computer and could easily compare compile times, runs times and output. We have even run AVS on the remote CM-5 and displayed the color graphics results in the classroom, with all the results being computed 500 miles away.

An outline of the course is shown below:

I. **Computer Hardware and History** (four lectures)
   MIMD v. SIMD
   Data Parallel Programming
   Message passing
   Workstation Clusters
   Future Computers

II. **Operation of Computers** (six lectures)
   Unix, X-windows, vi, and emacs
   CM-200, and Sun Front-End
   Prism and Grafic V3
   Silicon Graphics Workstations
   Andrew File System (AFS)
   Distributed Queuing System (DQS)

III. **Languages** (three lectures)
   FORTRAN 90
   High-Performance FORTRAN (HPF)
   Parallel Virtual Machine (PVM)
   Message Passing Interface (MPI)

IV. **Parallel Performance Estimates** (four lectures)

V. **Partial Differential Equations** (four lectures)
   Finite Differences
   Laplace's Equation
   Boundary Conditions

VI. **Linear Algebra** (eight lectures)
   Full Matrices
   Tridiagonal and Block Tridiagonal Matrices
   Jacobi Method
   Conjugate Gradient Method
   BLAS/LAPACK

VII. **Conclusions** (one lecture)

An important component of the course was instruction in modern progamming languages that are designed specifically for parallel computers. FORTRAN has been the most-used language for scientific computing, in spite of its weaknesses compared to other procedure oriented programming languages (such as C). FORTRAN 90, however, does offer some modern programming constructs (such as recursion) and a number of

features that facilitate parallelism and vectorization.

The simplest such constructs come out of translating DO loops into parallel or vector code. For instance, we can transform the loop:

```
DO I=1,N
  IF (C(I) .NE. 0.0) A(I) = A(I+1) / C(I)
ENDDO
```

into the parallel construct:

```
WHERE (C .NE. 0.0)
  A = CSHIFT(A,1,1) / C
ENDWHERE
```

The latter structure allows all of the operations to be done at once. It is important for students to learn to use vectorizing and parallelizing compilers. It is also important to learn to find ways of exploiting the parallelism in the code on their own since such compilers do not spot all parallel constructs. For instance, some sequential dependencies can be unwound. A typical example is:

```
DO I=2,N
  A(I−1)=NEW(I)
  OLD(I) = A(I)
ENDDO
```

This does not seem to be parallelizable, but the two equivalent loops:

```
DO I=2,N                DO I = 2,N
  OLD(I)=A(I)             T(I) = A(I)
  A(I−1)=NEW(I)           A(I−1) = NEW(I)
ENDDO                     OLD(I) = T(I)
                        ENDDO
```

both have exploitable parallelism. Students should learn how to write loops in a manner where a sophisticated compiler can spot parallel and vector constructs. On the other hand, true linear recurrences (such as the one for Gaussian elimination on tridiagonal matrices) must be unwound using more sophisticated algorithmic techniques. Special compilers cannot be expected to do it. Instead the student must be taught how to use these algorithmic techniques.

Perhaps the most important contribution of FORTRAN 90 is its ability to treat arrays as simple data objects. This has always made sense both mathematically and for the development of algorithms. However, this change was strongly encouraged by the development of parallel computers. These array and vector facilities are introduced to the students along with the language FORTRAN 90 and the development of matrix algorithms.

Another important nonnumerical area is the measurement of algorithm performance. For instance, Schwartz [10] introduced the notion of a paracomputer. This computer is an infinite array of processing elements, each of which may access a common memory in parallel for any piece of data. Essentially, the normal causes of inefficiency, routing delays and memory conflicts are not present and the computer always has enough processing elements.

One of the most well-known methods for analyzing algorithm performance is Hockney's [11] $n_{1/2}$ method, where $n_{1/2}$ is the size of the problem that obtains 50% of the processor efficiency. Hockney and Jessope [12] show how this performance measurement can be used to yield important information for both parallel and vector algorithms.

We also tried to stress the trade-offs between communication, computation and branching. A formula developed by Long [13] illustrates how to match the best algorithm to a particular computer. The processor speed and the network bandwidth have a significant impact on which algorithms will be effective.

Programming assignments

Several programming assignments were given during the semester:

- Array addition and multiplication.
- Jacobi algorithm applied to the two-dimensional Laplace equation.
- Conjugate gradient applied to the two-dimensional Laplace equation.
- Level 2 BLAS routine.

Each of these were programmed in FORTRAN 90 on the Connection Machine. For each assignment they were told to do things such as:

- Run three different size problems and compare CPU times and efficiencies.
- Plot solutions.
- Compute megaflop rates.
- Compute ratio of communication to computation time.
- Plot convergence histories.

The students also had to work two written homework assignments. These were both in the area of linear algebra and BLAS algorithms.

The first assignment was designed primarily to acquaint them with editing, compiling and running jobs on the CM-200 from remote workstations. None of the students were familiar with AFS or DQS, so this first assignment was just to allow them to become familiar with these systems and the CM-200. They also learned how to time their programs and compute megaflop rates. It was also their first exposure to FORTRAN 90 and array constructs.

In the second assignment, they solved a two-dimensional incompressible fluid flow problem using the Jacobi method. This is an appropriate second assignment since it is quite easy to program; however, it is quite slow to converge. The students ran several grids of varying numbers of grid points, and they saw how the computer became more efficient as the virtual processing ratio increased.

For the third assignment they had to solve the same fluid flow problem as the second one, but using a much more sophisticated algorithm: the conjugate gradient method. The students were

given a conjugate gradient subroutine written in FORTRAN 90 and they had to write the main program and call this subroutine. They were able to see that this program ran quite fast, and converged very rapidly.

It would have been useful also to assign an implicit algorithm that involved tridiagonal matrices with a cyclic reduction scheme, but there was not enough time. We did discuss tridiagonal schemes and block tridiagonal schemes in detail in the lectures, and students had some non-programming assignments. Next time we will try to give assignments that include explicit, implicit and iterative algorithms.

In the fourth programming assignment, the students had to write a Level 2 BLAS routine to perform matrix–vector multiplies. They also had to compare the efficiency of their program to the MATMUL routine supplied with the computer. They had to discuss how the version written in a lower-level language could be programmed to run more efficiently on the CM-200.

These computer assignments required significant computer resources. A record was kept of the number of jobs submitted through the DQS system. The students compiled roughly 7000 programs and they ran approximately 3000 jobs on the CM-200 during the semester. In addition, the computer center, the faculty and the teaching assistant responded to roughly 200 e-mail messages, which was a very effective way of addressing some of the students' questions. We have also started using Mosaic for courses.

Students on the course

There were 33 students on the course. This was a senior-level course, but it was also open to graduate students. The distribution of students is shown in Table 1. The class consisted of 67% undergraduate students. Most of the students were either from the Aerospace Engineering or Computer Science Departments. There were four graduate students from Nuclear Engineering. Some of the other departments represented were Astronomy, Electrical Engineering, Chemical Engineering, Mechanical Engineering and Engineering Science.

It is often difficult having undergraduate and graduate students in the same course. It was especially difficult in this course since the range of backgrounds was very wide. There were some students who were not well prepared in FORTRAN 77, UNIX and partial differential equations. At the other extreme, there were graduate students who were quite familiar with numerical methods, programming and applied mathematics. The most difficult aspect was that some students were not well prepared in linear algebra. The next time this course is taught (Spring 1994), a much more complete list of prerequisites will be given, to narrow the range of student backgrounds. We are also coordinating our efforts with the Electrical Engineering Department, since they have begun offering a graduate course in parallel processing.

Books used in the course

There is no one single book that is appropriate for this course. We used several books, including:

- *Getting Started in CM-Fortran*, Thinking Machines Corporation, 1991.
- *Parallel Numerical Algorithms*, C. L. Freeman and C. Phillips, Prentice-Hall, 1993.
- *FORTRAN 90 Explained*, M. Metcalf and J. Reid, Oxford University Press, 1991.
- *Advanced Computer Architecture*, K. Hwang, McGraw-Hill, 1993.
- *Applied Numerical Analysis*, C. F. Gerald, Addison-Wesley, 1989.
- *CM-FORTRAN Reference Manuals*, Thinking Machines Corporation, 1992.

Copies of the first book were obtained directly from Thinking Machines Corporation, and the students were asked to buy the second book. The other books were put on reserve in the library. The students would have preferred to have a single textbook that we followed closely, but this was just not possible due to the newness of the material.

HONORS PROJECTS AND SENIOR THESIS

In addition to the new course, undergraduates have also been performing research in parallel processing. Students on the Penn State University Scholars Program are required to perform an undergraduate research project and write a senior thesis. Entrance into this program is by invitation only and it attracts the highest-quality undergraduate students. The presence of the CM-200 computer has generated honors projects and senior theses related to scientific computing and parallel processing. Undergraduate honors projects on a parallel architecture are excellent preparation for graduate research in scientific computing as well as

Table 1. Distribution of students in parallel processing course

Department	Undergraduate	Graduate	Undergraduate %	Graduate %
Aerospace Engineering	7	3	21	9
Computer Science	12	1	36	3
Nuclear Engineering	0	4	0	12
Other Engineering	3	3	9	9
Total	22	11	67	33

for jobs in industry. There is a great shortage of students trained in high-performance computing. The ability to do more interesting, practical projects at the undergraduate level will attract more good students in graduate study in this important area.

We will briefly describe four senior theses that have been written at Penn State in the area of high-performance computing. The topics of these were as follows:

- Parallel Jacobi and aerodynamics on the 3090-600s.
- Solving the Boltzman equation on the Connection Machine.
- Acoustic propagation solutions on the Connection Machine.
- Audio and video postprocessing on the Silicon Graphics.

The first project was in aerodynamics. The students (D. Gottfried and K. McGinniss) took a vortex lattice code written in FORTRAN 77 and modified it to use all six processesors of the IBM 3090 simultaneously. The resulting matrix was split across the processors and solved using the Jacobi algorithm.

The second project was a senior thesis of an Engineering Science student (M. Kamon). It involved solving the Boltzmann equation on the Connection Machine using FORTRAN 90. This is a very difficult problem, but the student was able to write a program to solve for the structure of a shock wave using the BGK form of the Boltzmann equation. This program has been under continual development since the student graduated and the latest version of the code was awarded the Gordon Bell prize in 1993 [14]. This code *sustained* 60 gigaflops on a 1024 processor CM-5, which is roughly 50% of the peak speed.

The third and fourth projects were related and were senior theses in Aerospace Engineering. Scott Reid wrote an acoustic propagation code for the Connection Machine for two-dimensional duct acoustics. This code used a finite-difference scheme and Runge–Kutta time marching. Chris Tatnall then wrote a postprocessing code for this code, which displayed the acoustic results using a color graphics program on the SGI workstation. The post-processing code also was able to play the predicted acoustic signatures through the SGI audio hardware.

CONCLUSIONS

This paper describes a recent project in teaching high-performance computing and parallel processing to undergraduate students. We feel this is an important area of study, and an introduction at the senior level is appropriate. This type of computing is quite different to traditional computer science or numerical methods, and should be introduced as early as possible in the students' career.

Acknowledgements—This work was supported by the National Science Foundation (grant no. CDA-90-50874) and Penn State University. The authors would like to thank Mr Dale Hudson, the teaching assistant for the course, for his valuable contributions. In addition, we gratefully acknowledge Thinking Machines Corporation for their assistance.

REFERENCES

1. Anon., *The Federal High-performance Computing Program*. Executive Office of the President, Office of Science and Technology, Washington, DC (1989).
2. Anon., *Strategic Computing: Fourth Annual Report*, DARPA, Washington, DC (1988).
3. J. W. Demmel, J. J. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling and D. Sorensen, Prospectus for the Development of a Linear Algebra Library for High-performance Computers, technical report no. ANL/MCS-TM-97, Mathematics and Computer Science Division, Argonne National Laboratory (1987).
4. L. N. Long, M. Khan and H. T. Sharp, A massively parallel Euler/Navier–Stokes method, *AIAA J.*, **29**, (4), 657–666 (1991).
5. Z. Weinberg and L. N. Long, An unstructured, adaptive, upwind scheme for the Navier Stokes equations, *Proc. Parallel CFD '93*, Paris (1993).
6. B. C. Wong and L. N. Long, Direct simulation Monte Carlo (DSMC) on the Connection Machine, *Comput. Sys. Engng*, **3** (1–4) (1992).
7. A. Beguelin, J. Dongarra, A. Geist, B. Manchek and V. Sunderam, PVM 3.0 User's Guide and Reference Manual, technical report TM-12187, ORNL (1993).
8. T. Green and R. Pennington, Distributed Queuing System, Version 2.0, Supercomputer Computations Research Institute and Pittsburgh Supercomputer Center (sources may be obtained via anonymous ftp from ftp.scri.fsu.edu).
9. AFS version 3.2, Transarc Corporation, 707 Grant Street, Pittsburgh, PA 15219, USA.
10. J. T. Schwartz, Ultracomputers, *ACM Trans. Prog. Lang. Syst.*, **2**, 484–521 (1980).
11. R. W. Hockney, Characterization of parallel computers and algorithms, *Comput. Phys. Commun.*, **26**, 285–291 (1982).
12. R. W. Hockney and C. R. Jesshope, *Parallel Computers 2*, Adam Hilger, Bristol (1988).
13. L. N. Long, Gas dynamics on the Connection Machine, Invited Paper, Parallel CFD '92 Conference, Rutgers University (1992).
14. L. N. Long and J. Myczkowski, Solving the Boltzmann Equation at 61 Gigaflops on a 1024-node CM-5, *Proc. of Supercomputing '93*, Portland, Oregon, Nov. (1993).

Dr Lyle N. Long is an Associate Professor of Aerospace Engineering. He teaches courses in numerical algorithms and fluid dynamics. Dr Long has developed several large computer programs for the Connection Machine for solving problems in computational fluid dynamics, electromagnetics, color graphics and linear algebra. The most notable achievement was the development of a parallel computer program to solve the non-linear, three-dimensional Boltzmann equation of fluid dynamics on a massively parallel Connection Machine. The code has been nominated for the 1993 Gordon Bell Award. He received BME, MS and DSc degrees from the University of Minnesota, Stanford University and George Washington University, respectively. He spent six years at Lockheed Aeronautical Systems Company developing a wide range of codes for vector and parallel computers to solve time-accurate fluid dynamics, electromagnetics, aerodynamics, viscous flows and rarefied gas dynamics.

Dr Jesse L. Barlow is a Professor of Computer Science and has performed research on the effect of computer architecture and computer arithmetic on algorithms in numerical linear algebra. His papers have included material on parallel computing environments ranging from systolic arrays to the Intel iPSC. He has taught graduate courses on matrix computation and the solution of differential equations and taught undergraduate courses in numerical methods, programming languages, data structures and algorithms, and computer architecture.

Lou Constable was a research programmer for the Numerically Intensive Computing Group in the Center for Academic Computing at the Pennsylvania State University. Prior to joining the Center for Academic Computing, he received his BS degree in computer science and mathematics from the Pennsylvania State University. He currently works at Cornell University in the Theory Center.

Kevin Morooney is the manager of the Numerically Intensive Computing Group in the Center for Academic Computing at the Pennsylvania State University. He has worked in the Numerically Intensive Computing group at Penn State for the last four years. Prior to working at Penn State, Kevin worked as a biomedical engineer at the Johns Hopkins University Teaching Hospital Francis Scott Key Medical Center in the Departments of Neurology and Pulmonary Medicine. He was awarded his BS in Engineering Science and Mechanics from the Virginia Polytechnic Institute and State University in 1985.