ED 425 723	IR 019 211
AUTHOR	Marcus, Robert L.; Robertson, Douglass
TITLE	Web Based Parallel Programming Workshop for Undergraduate Education.
PUB DATE	1998-00-00
NOTE	11p.; In: Association of Small Computer Users in Education: Proceedings of the ASCUE Summer Conference (31st, North Myrtle Beach, SC, June 7-11, 1998); see IR 019 201.
PUB TYPE	Reports - Descriptive (141) -- Speeches/Meeting Papers (150)
EDRS PRICE	MF01/PC01 Plus Postage.
DESCRIPTORS	Algorithms; Computer Assisted Instruction; *Computer Science Education; Computer System Design; Higher Education; Instructional Materials; *Material Development; *Programming; Programming Languages; Undergraduate Study; *Workshops; *World Wide Web
IDENTIFIERS	Central State University OH; Course Development; Department of Defense; FORTRAN Programming Language; *High Performance Computing; *Parallel Distributed Processing; Web Sites

ABSTRACT

Central State University (Ohio), under a contract with Nichols Research Corporation, has developed a World Wide web based workshop on high performance computing entitled "IBN SP2 Parallel Programming Workshop." The research is part of the DoD (Department of Defense) High Performance Computing Modernization Program. The research activities included developing techniques for converting classroom materials to Web presentations and algorithms in parallel programming techniques. Traditional classroom materials were prepared for Web presentations using the Accelerated Web Page System in the Scientific Visualization Center which used an optical character reader to scan printed material, and the Omnipage-Pro toolkit to produce HTML. The Internet Assistant software under Microsoft Office 97 was used for converting Microsoft Word documents to Web page files. The workshop material presents a series of sample programs on parallel programming using FORTRAN 90. It discusses topics on process synchronization, deadlock, data distribution, and load balancing. The workshop is designed for entrant level parallel programmers to make it suitable for undergraduate instructions and DoD applications. Basic algorithms on sorting, searching, statistical computations, numerical methods and linear systems are presented. The workshop is hosted on the Web server at Central State University and Wright Patterson Air Force Base, Dayton, Ohio. The workshop was successfully field tested on undergraduate students at Central State University. (Author/AEF)

Web Based Parallel Programming Workshop for Undergraduate Education

Mr. Robert L. Marcus, Computer Science
Mr. Douglass Robertson, Scientific Visualization Center
Central State University
Wilberforce, Ohio 45384

Abstract

Central State University has developed a web based workshop on high performance computing under a contract with Nichols Research Corporation (NRC) entitled "IBN SP2 Parallel Programming Workshop." The research is part of the DoD High Performance Computing Modernization Program (HPCMP). The research activities included developing techniques for converting classroom materials to web presentations, and algorithms in parallel programming techniques. Traditional classroom materials were prepared for web presentations using the Accelerated Web Page System in the Scientific Visualization Center which used an optical character reader to scan printed material and the Omnipage-Pro toolkit to produce HTML. The Internet Assistant software under MSOffice 97 was used for converting MS Word documents to web page files. The workshop material presents a series of sample programs on parallel programming using FORTRAN 90. It discusses topics on process synchronization, deadlock, data distribution, and load balancing. The workshop is designed for entrant level parallel programmers to make it suitable for undergraduate instructions and DoD applications. Basic algorithms on sorting, searching, statistical computations, numerical methods and linear systems are presented. The workshop is hosted on the web server at Central State University and the web server at Wright Patterson Air Force Base (WPAFB), Dayton, Ohio. The workshop was successfully field tested on undergraduate students at Central State University.

Introduction

Under the NRC contract Central State University was charged with developing web based education/training programs tailored for DoD high performance computing (HPC) users, other academic institutions and students. As such, Central State University developed the "IBM SP2 Parallel Programming Workshop." The programs were developed on a 128 node IBM SP2 system under a classroom grant from the ***Ohio Supercomputer Center*** (OSC). They were rehosted on the IBM SP2 system at the ASC/MSRC at WPAFB. The web version of the workshop is installed on the web server at CSU and WPAFB.

The workshop was designed for use in undergraduate education, or for entrant level HPC DoD programmers. Algorithms were selected from data structures, statistics, numerical methods, and linear systems. Consequently, the workshop is appropriate for (science and/or engineering) students at the sophomore level or above. Workshop pre-requisites are:

- FORTRAN 90 Programming
- An Introduction to UNIX Shell Scripts

The current versions of the programs in the workshop use task communication and synchronization constructs in the Message Passing Library (MPL). Conversions to MPI (an industry standard) is underway.

Accelerated Web Page Development System

The Center for Scientific Visualization developed the techno-scan system to reduce the amount of time needed to develop a web page. Some faculty in the department used Microsoft's Internet Assistant to develop web pages, but the quality of the HTML was not sufficient. The techno-system also reduced the amount of time to input graphics by using WYSIWYG interface. This rapid system relies heavily on OCR (Optical Character Recognition) technology and macros for its hard copy to HTML manipulation. Considerations are being given to using web tools such as FrontPage and Page Mil for developing a web site since most faculty now have suitable computing platforms for preparing text and graphical material for the web electronically.

Parallel Programming Design Issues

Three paradigms were emphasized:

apply Software Engineering principles using a modular design
use message passing only where necessary
give preference to group communication constructs

The first design paradigm is to apply traditional software engineering principles to develop programs with a modular design using functions and subroutines to separate communications from parallel computations. This facilitates optimizing the program to minimize time for communications, and maximize the time for parallel computations. Program designed to run in a distributed memory parallel environment, such as the IBM SP2, usually have three basic components that perform the following tasks:

- distribution of data using message passing
- execute local algorithms on nodes in the partition
- collect data from remote nodes using message passing

In the workshop programs, these components were named:

- distribute
- node_***application***
- collect

The term ***application*** is replaced by the name of the given application for the sample program (see the structure chart for each sample program). For example, the program ***mp_gaussm.f*** has a subroutine named node_***gauss*** which performs the Gaussian elimination algorithm on a partition (or group) of distributed nodes (or tasks). See the structure chart and data-flow diagram below.

The second design paradigm is to use one of the following data distribution methods, if the application design permits, as alternatives to message passing:

- loop parallelization, or
- parallel reads

These two methods will provide improved speedup of parallel processing over the use of message passing. Several workshop programs demonstrated that technique (the names of the programs ends with ***"lp.f")***. In other applications, collecting data may be inherent in the algorithm as demonstrated by the sorting example mp_sort.f. It uses a sort-merge algorithm. The merging process is designed so that the final merge of data occurs on task0, the master/control node, and there is no need for a distinct component to collect results.

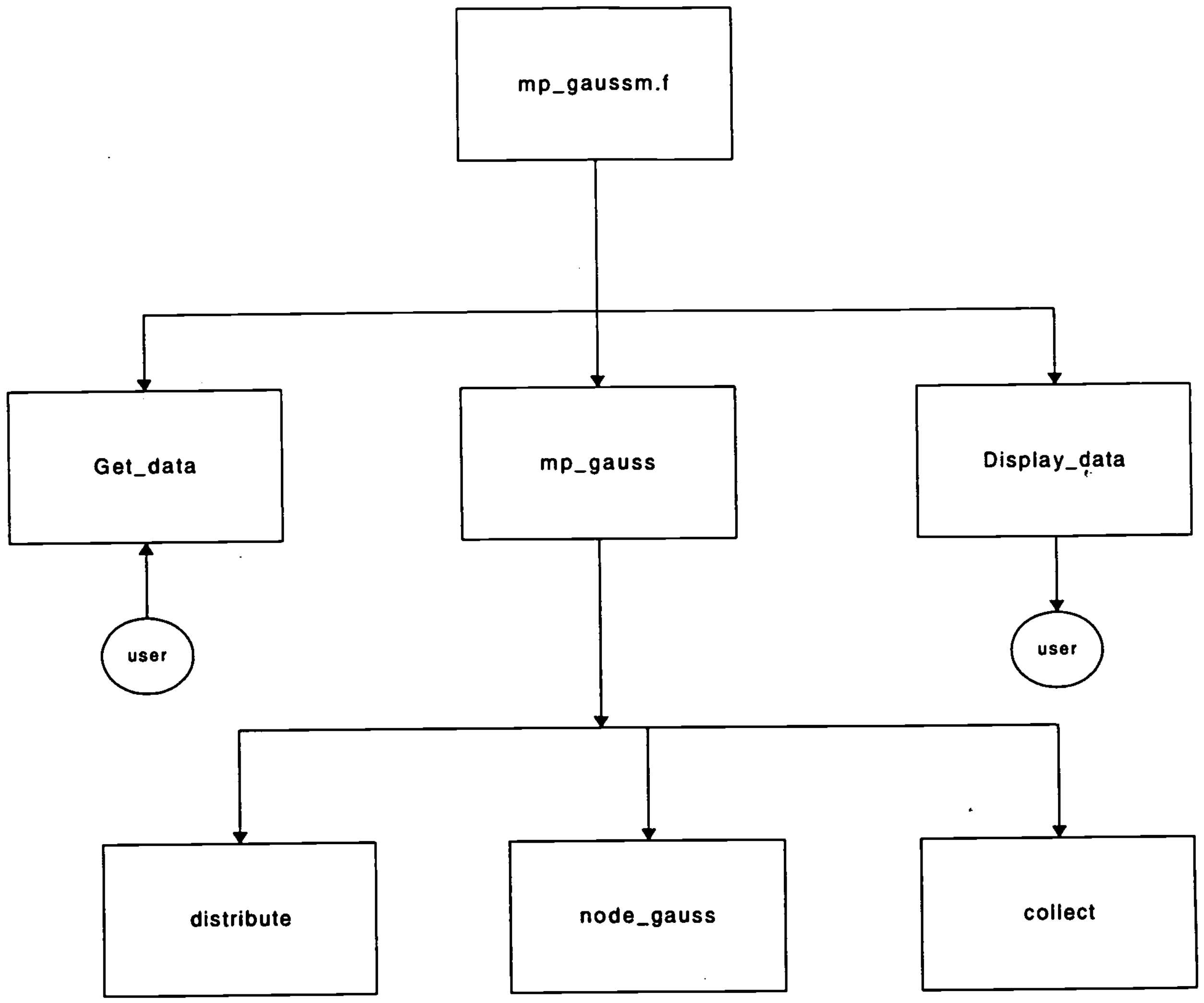
The third design paradigm is to use group communication constructs which simplifies program structure, improves its readability, enhances its maintainability. However, point-to-point communication constructs may be used to design tailored communication algorithms that are more efficient for a given application. Types of group message passing constructs are:

- point-to-group
- grout-to-point
- grout-to-group.

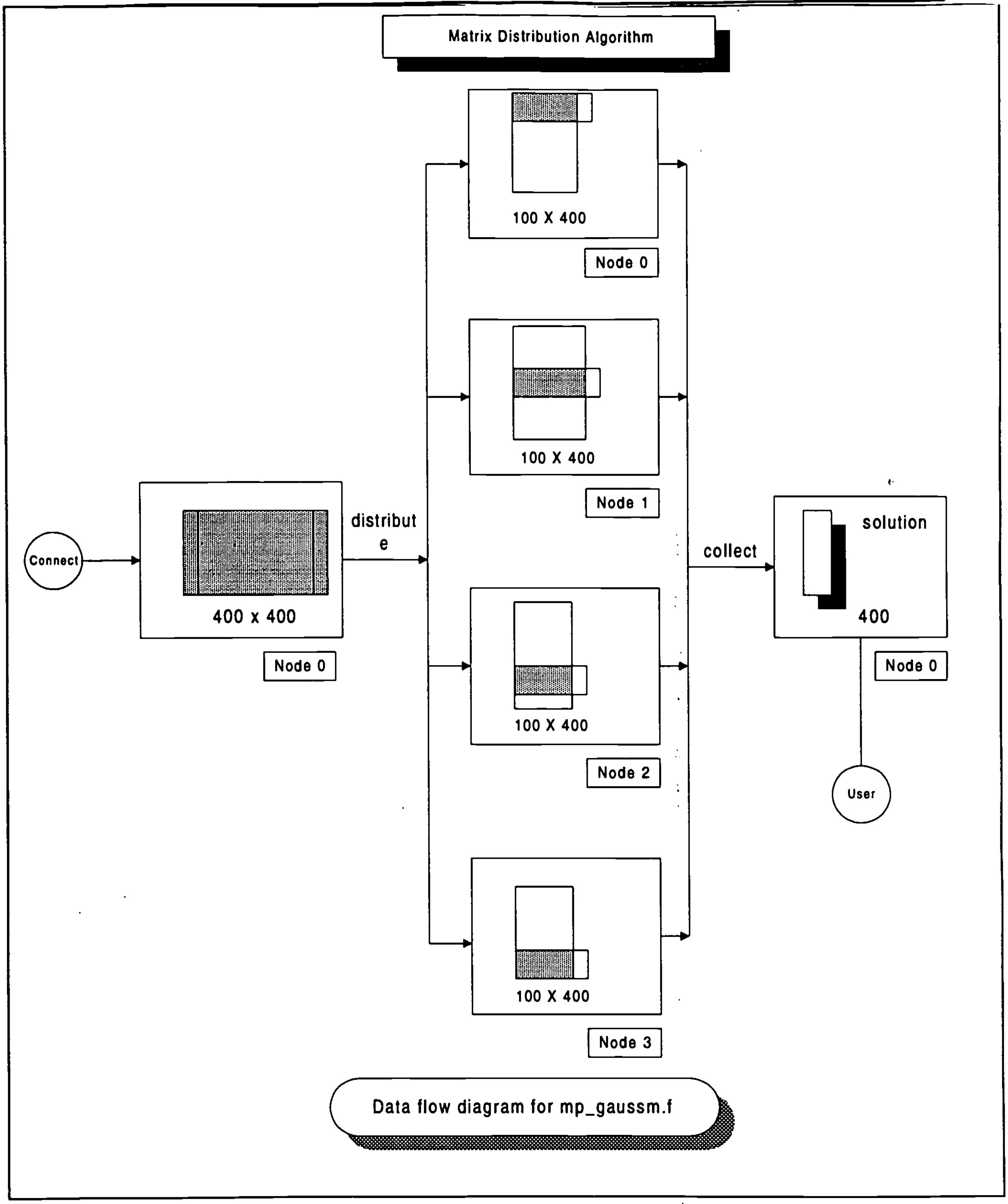
Figures and Diagrams:
Each program in the workshop included figures as the ones shown below for the program **mp_gaussm.f.**

Topic:	**Distributing a two dimensional matrix of data**
Program:	**mp_gaussm.f**
Purpose:	Solves an NxN linear system using a matrix distribution for the data and a Gauss-Jordan pivoting method.
Constructs:	mp_bcast --broadcasting mp_bsend -- blocking send mp_brecv -- blocking receive mp_bcast -- broadcast to group
Description:	This program distributes an NxN linear system of data to several tasks using a whole matrix distribution. The Gauss-Jordan pivoting method is used to pivot the system (a variation of matrix diagonalization) in parallel, leaving the solutions in column N+1 on each task. The solutions are passed to task0, put in standard order and then printed. The size of the linear system was 400x400.
Scalability:	YES, limited The maximum real time decreased for two tasks and increased when using three and four tasks.

Documentation Template: mp_gaussm.f



Structure Chart for mp_gaussm.f


Matrix Distribution Algorithm
Connect
400 x 400
Node 0
distribut e
100 X 400
Node 0
100 X 400
Node 1
100 X 400
Node 2
100 X 400
Node 3
collect
solution
400
Node 0
User
Data flow diagram for mp_gaussm.f

Twelve Basic MPL Constructs

All workshop programs were implemented using commands from the following list:

1. Parallel environment commands:
   MP_ENVIRON, Returns the number of tasks in the partition and the caller's task id.
   MP_TASK_QUERY, Returns information about system variables and constants.
2. Point-to-point non-blocking message passing:
   MP_RECV, Posts a receive buffer for a message and returns without waiting for the message arrival.
   MP_SEND, Identifies a message to be sent and returns without waiting for the send to complete.
   MP_STATUS, Returns status of a non-blocking send or receive.
   MP_WAIT, Waits until a non-blocking send or receive has completed.
3. Group message passing:
   MP_BCAST, Distributes a message throughout a group.
   MP_GATHER - Concatenates a distinct message from each task to create a single message on task dest.
   MP_SCATTER, Distributes distinct messages to each task in the group.
4. Point-to-point blocking message passing:
   MP_BRECV, Receives a message and waits until the requested data arrives.
   MP_BSEND, Sends a message and waits until the ouput buffer can be reused.
5. Define a group within a partition
   MP_GROUP, Explicitly defines a task group.

Workshop Chapters

The following is a listing of the workshop chapters.

1.0 Introduction
2.0 Parallel Programming Design
3.0 Program Execution
4.0 Topics in Parallel Programming
    Communications
    Synchronization
    Deadlock
    Cache Efficiencies
5.0 Point-to-Point Message Passing
5.1 Sample Program mp_max1.f
6.0 Point-to-Group and Group-to-Point Message Passing
6.1 Sample Program mp_max2.f

6.2 Sample Program mp_stats.f
    Statistical Computations: Maximum, total, average, average deviation, standard deviation, and average square
7.0 Loop Parallelization

Chapter Descriptions

The first four chapters presented important background information on parallel programming. The other chapters presented sample programs on the topics indicated. Chapter 4 presented some introductory examples to demonstrate how programs are executed in a distributed parallel environment.

Chapters 5 presented the program, mp_max1.f, which computed the maximum value of an array of 400,003 elements. Only point-to-point message passing was used. All communication constructs were blocking. The program was not scaleable (see table below). Chapter 6 presented a modified version of mp_max1.f, named mp_max2.f, which used point-to-group and group-to-point message passing. Blocking and non-blocking constructs were used. Program mp_max2.f was not scaleable either. Chapter 6 presented a second program, mp_stats.f, which computed several statistical parameters shown above. The strategy was to perform more parallel computations to offset the time due to communications (group communications). The array contained 100,003 elements. The program did not show scalability.

Chapter 7 presented two programs that used the loop parallelization technique: mp_statslp.f and mp_integral.f. Program mp_statslp.f was a modification of program mp_stats.f to use loop parallelization was to define data on each node instead of message passing to distribute the data. The array size was 400,003. The program showed scalability up to 5 nodes. The second program, mp_integral.f, is an example from numerical methods. It uses the trapezoidal rule to compute an integral which has as its value pi=3.1415926. The algorithm used 4,000,000 subintervals that were summed in parallel. The program showed scalability up to 5 nodes.

Chapter 8 presented a technique to replicate the ureka command (on the Cray T3D) which sends a signal from a source node to other nodes in a partition. This technique was implemented in the program mp_find.f. The technique used a non-blocking receive from "any source" node to receive a special value in a variable. A while search loop was entered to find a key value in an array. The receive variable (or buffer) is used in the boolean expression of the while statement so that when (or if) a value is received, the while loop is terminated. The task which finds the key value sends the special value to all other tasks in the partition.

Chapter 9 presented the use of the mp_group construct to define a sub-group of tasks within a given partition. The program mp_sort.f demonstrates a parallel sorting algorithm:

Data is distributed to all nodes in the sub-group
Sorting is done in parallel on each node
Merging is done in parallel

The process for merging is to repeated pair-wise merge data in the top half of the sub-group with nodes in the bottom half until all data is merged on the initial task (task 0). The program ensures that the sub-group is a power of 2. There were 10,000 elements in the array. The program was scaleable up to 8 nodes.

Chapter 10-13 presented techniques for distributing two-dimensional data. Each program solved an NxN linear system using the Gauss-Jordan method. In Chapter 10 program mp_gaussc.f distributed columns of the matrix of data to each node. The method required several inter-communications during each step of the pivoting process. A 20x20 linear system was solved, but it was not scaleable.

Chapter 11 presented a technique (using the broadcast construct) to distribute the entire matrix to each node. Each step of the pivoting process required only one communication step: broadcast the pivot row. A 400x400 linear system was solved, showing scalability up to 4 nodes.

Chapter 12 presented a loop parallelization technique to define rows of data on each node for the pivoting process. Each step of the pivoting process required only one communication step: broadcast the pivot row. A 400x400 linear system was solved, showing scalability up to 9 nodes.

Chapter 12 presented a parallel read technique for reading rows of data into the matrix for each node. The data was stored in a direct access binary file. Each step of the pivoting process required only one communication step: broadcast the pivot row. A 400x400 linear system was solved, showing scalability up to 6 nodes.

Scalability and Speedup

The workshop material emphasize speed-up as the primary goal of parallel processing. If possible, the algorithms were improved until the program demonstrated scalability (i.e. speed-up greater than one) for a reasonable size partition of nodes. Speed-up was computed from the following formula:

$$\text{Speed-up} = \frac{\text{Wall clock time with a single task}}{\text{Wall clock time with more than one task}}$$

The wall clock time is the maximum "real time" (obtained from using the ***timex*** UNIX command) for the tasks in the partition. We consider the algorithm to be scaleable over when the speed-up factor is greater than one.

The table below shows speed-up factors for the workshop programs.

Scalability and Speed-up Listing

Program	**Nodes**	**Maximum Real Time (sec)**	**Speed-up**
mp_max1.f	1	0.5	-
	2	0.88	0.57
	4	5.87	0.09
mp_max2.f	1	0.44	-
	2	0.85	0.52
	4	1.28	0.34
mp_stats.f	1	0.33	-
	2	0.55	0.60
	4	0.56	0.59
mp_statslp.f	1	2.38	-
	2	1.49	1.60
	4	0.79	3.01
	5	0.66	3.61
mp_integral.f	1	3.14	-
	2	2.04	1.54
	3	1.25	2.51
	4	1.07	2.93
	5	1.45	2.17
mp_find.f	1	0.45	-
	2	0.57	0.79
	4	0.68	0.66
mp_sort.f	1	72.06	-
	2	19.59	3.68
	4	4.35	16.56
	8	2.77	26.01
mp_gaussc.f	1	0.30	-
	2	0.38	0.79
	3	0.48	0.63
	4	0.49	0.62
mp_gaussm.f	1	55.36	-
	2	32.26	1.72
	3	21.45	2.58
	4	16.60	3.33
mp_gausslp.f	1	36.56	-
	4	15.27	2.39
	6	10.58	3.46
	9	8.42	4.34
mp_gausspr.f	1	46.89	-
	2	29.24	1.60
	4	16.99	2.76
	6	11.59	4.05

The following programs:

mp_max1.f
mp_max2.f
mp_stats.f
mp_find.f
mp_gaussc.f

were not scaleable because the time saved from performing parallel computations did not offset the additional time needed for communications. The other programs:

mp_statslp.f
mp_integral.f
mp_sort.f
mp_gaussm.f
mp_gausslp.f
mp_gausspr.f

were scaleable.

Future Research Directions

Convert the programs to MPI.
Write C versions of all programs.
Write similar programs for a shared memory system (Origin 2000).
Implement a small scale application program to generate volume data for visualization purposes.

Conclusions

The workshop was used successfully to teach part of a short introductory course on parallel computing for undergraduate students. Some of the workshop programs were written by students who took the course: CPS 460 – Advance Topics, during the winter of 1997. The workshop used algorithms that were familiar to science and engineering underclassmen. Even though only 12 basic MPL constructs were used in the examples, the workshop included man pages on all MPL constructs for additional reference. Central State University continue these activities and expand the workshop to a 4 credit hour course on An Introduction to Parallel Computing.

References

[1] Analytical, Numerical, and Computational Methods for Science and Engineering, Hostetter, Santina and Montalvo, Prentice Hall, 1991

[2] An Introduction to IBM SP2 Programming, David J. Ennis, The Ohio Supercomputer Center, 1996

[3] CTC Virtual Workshop on Parallel Computing and Progamming Languages, Cornell University, September, 1997