

Finite difference and finite element analyses using a cluster of workstations

K.P. Wang, J.C. Bruch, Jr.

*Department of Mechanical and Environmental Engineering,
University of California, Santa Barbara CA 93106, USA*

Abstract

Because of high computing speed, cost effectiveness, and scalability, parallel computation on clusters of workstations is becoming one of the major trends in the study of parallel computation. This paper presents studies of using a cluster of workstations for finite difference analysis and finite element analysis. A parallel algorithm proven to be simple to implement and efficient for both analyses is used to perform them on a cluster of workstations. A network of workstations is utilized as the hardware of a parallel system. Two popular parallel software packages, PVM (Parallel Virtual Machine) and P4, are used to handle the communications among the networked workstations. Also used for comparison purposes are the Paragon and Meiko CS-2 computers. Furthermore, an approach to develop a portable parallel code is given.

1 Introduction

For the past few years, with the advanced technology in the computer industry, workstations have been produced with high computing speed and at low cost. Because of their high computing speed, cost effectiveness, and scalability, parallel computation on clusters of workstations is becoming one of the major trends in the study of parallel computation. Studies of using a cluster of workstations for both finite difference analysis and finite element analysis are presented herein.

Previous work [1]-[4] has shown that the SOR (Successive Over-Relaxation) iteration method for the finite element and finite difference methods can be fully parallelized by reordering the discretized equations. Speedups close to linear (theoretical speedup) or better have been obtained using the iPSC/2 Hypercube parallel computer.

132 High-Performance Computing in Engineering

P4 [5] and PVM [6] are message passing libraries for a cluster of workstations and parallel computers. With P4 or PVM, a cluster of workstations can be used as if it were a single parallel computing resource. P4 was developed at Argonne National Laboratory and PVM was developed at Oak Ridge National Laboratory. The version of P4 used in this study is 1.4 while the version of PVM is 3.3.4.

A cluster of 7 SGI Indy workstations running the Irix 5.1 operation system was used for this study. Each workstation is equipped with a ethernet card. All workstations are networked with a central file server. The transmission is slow when compared to around 30 MB/sec for a parallel computer. However, it is common to see the same type of network configuration in many institutions.

The two parallel computers that will also be used are the Paragon and Meiko CS-2 computers. Both computers are MIMD distributed memory multi-computers. Processors on the Paragon or Meiko can only communicate by message passing. The programming model used is SPMD (Single Program Multiple Data). Every processor is loaded with the same code, but execute a different branch of the code or operate on a different set of data. This is not the only way a parallel program can be written but is a widely used model.

This study shows an approach to develop a portable parallel code as well as the feasibility of using a cluster of workstations to perform parallel computation. The approach for developing portable codes using a cluster of workstations will be presented first. The same codes are compiled and executed using P4 and PVM on a cluster of SGI workstations and on the Paragon and Meiko parallel computers. Speedups for all test cases will be shown in order to discuss the feasibility of using clusters of workstations for parallel computation.

2 Implementation

In solving many engineering problems, both the finite difference method and the finite element method will lead to a linear system:

$$Ku = f$$

where K is the coefficient matrix, f is the force vector and u is the solution vector. The K will be a banded diagonal matrix in general. Thus, it is possible to perform reordering of the equations in the linear system in order to decouple the system of equations. The process of reordering equations is equivalent to decomposing the computation domain into subdomains and interfaces. The parallel SOR iterative algorithm presented in [1]-[4] uses this idea to transform the sequential SOR to a fully parallel SOR algorithm in the sense that computations in all the subdomains are performed in parallel and the computations on all the interfaces are also performed in parallel.

High-Performance Computing in Engineering 133

The model problem to be solved is the free surface seepage problem presented in [1]-[4]. Because the solution of the reformulated problem of this model problem has a constraint of having to be greater than or equal to zero, no system of equations can be generated. Only a pointwise iterative scheme can be formulated as presented in [1]-[4].

The first step to implement the parallel algorithm on all the parallel systems is to convert the parallel codes developed from previous studies to one of the parallel systems. The second step is to make the code portable. Since there are differences in different message passing systems, the easiest way to make a parallel code portable is to use only a message passing library and translate all other message passing libraries to that message passing library. The translation mechanism will be unique for each system. However, once developed, all parallel codes developed later can use the same mechanism without modifying the parallel codes.

The original implementation of the SOR parallel algorithm was on the Intel iPSC/2 Hypercube parallel computer. The programming model was a host-node model. The input and output are controlled by a host program and the computation was handled by the node program. The message passing library in the iPSC/2 Hypercube is similar to the NX message passing library for the Paragon. The first step in this study is to convert parallel programs from the host-node programming model to a SPMD programming model used on the Paragon. This conversion can be achieved by assigning processor 0 as the host processor. An extra step is to convert the old iPSC/2 function calls to the NX functions.

To develop portable codes, it is intuitive to see that if all message passing libraries can be translated to the Paragon NX message passing library we can recompile the same parallel code with the translation mechanism on different systems without modifying the code. In this study only a few translation mechanisms need to be developed:

1. begin parallel program.
2. end parallel program.
3. send message.
4. receive message.
5. global collective operation of arrays.

Each mechanism does not mean an implementation for a function call. It may represent a group of function calls. Since different message passing libraries have different ways to begin or end the parallel processes and the NX library has no specific function calls to them, common function calls are needed for all systems to begin and end a parallel program. The first two mechanisms will be needed for all message passing libraries. Mechanisms 3-5 are required for P4,

134 High-Performance Computing in Engineering

PVM, and Meiko.

Upon completing the translation mechanism on a system, a parallel code can be compiled with the translation mechanism without changing any part of the code and run on that system. Speedup results and their evaluation are presented in the next section.

When testing the parallel codes using P4 or PVM on a cluster of workstations, it is important to make sure that there are no other users logged on the workstations to be used. Because if there are, the timing results will be affected by the computation load of the other users. Furthermore, it is also important to make sure there are no other users using computers on the same network even if they are not using the same workstations to be tested. If there are, the communication speed will be affected by these users. Therefore, the tests were performed during late night and quarter breaks.

3 Results and Discussion

Figures 1, 2, and 3 show the finite difference speedups for cases with (101,101), (141,141), and (201,201) mesh points. As shown in these three figures, speedups (sometimes better than the linear speedup because of the way boundary data is input) from the Paragon and Meiko parallel computers are better when increasing the number of mesh points. Similar trends can be observed for P4 and PVM. Their speedups, however, are only a little over 2 even when more than 2 processors are used. Speedups from P4 are better than those from PVM for all three cases. One explanation is that the communication speed on the

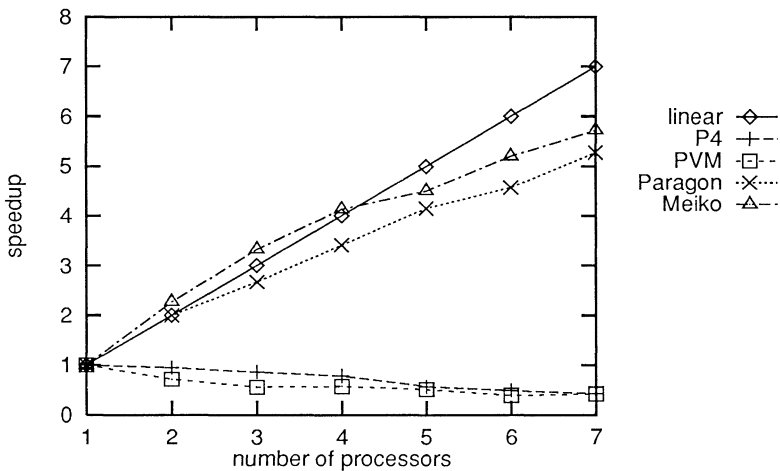


Figure 1: Finite difference speedup for (101, 101) mesh points.

High-Performance Computing in Engineering 135

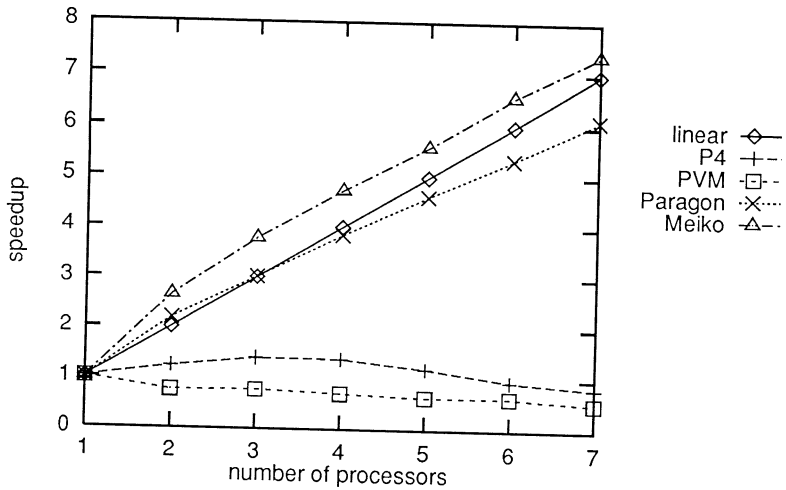


Figure 2: Finite difference speedup for (141, 141) mesh points.

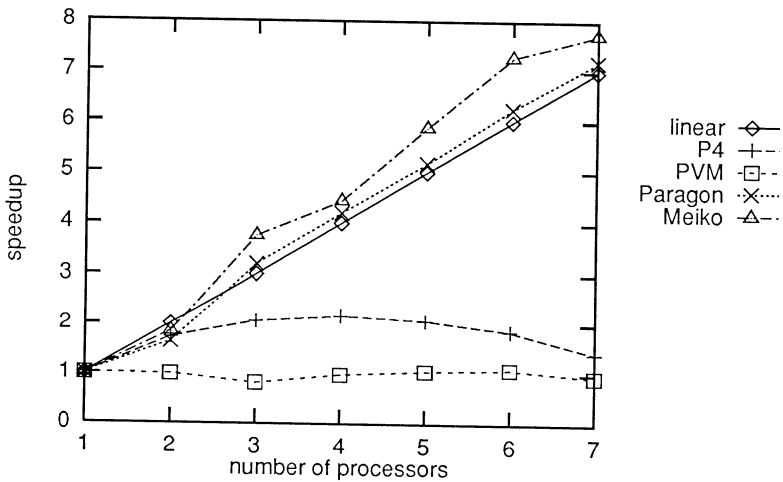


Figure 3: Finite difference speedup for (201, 201) mesh points.

cluster of workstations played an important factor in slowing down the parallel execution. The data transmission rate on the ethernet board is slow, and the communication management is not as efficient as the parallel computers. When the ratio of computation time versus the communication time is small, the speedup will be small even with increasing the number of processors.

These results, despite the speedup, show that a parallel program can be developed on a cluster of workstations first then moved to a parallel computer for

136 High-Performance Computing in Engineering

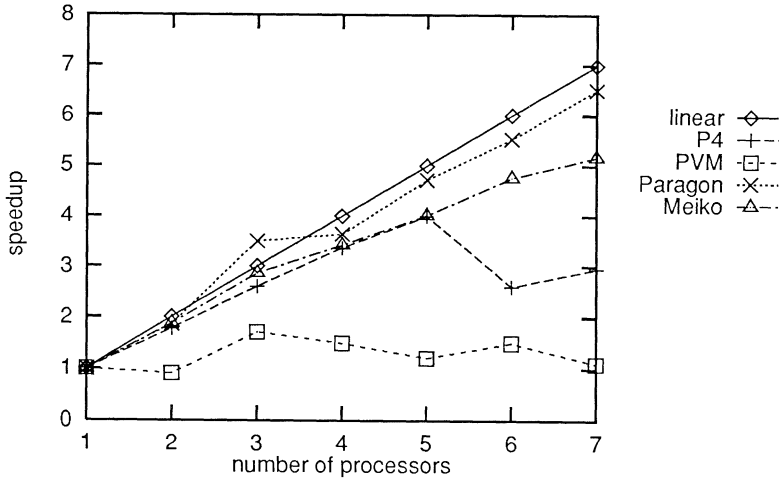


Figure 4: Finite element speedup for 4257 degrees of freedom.

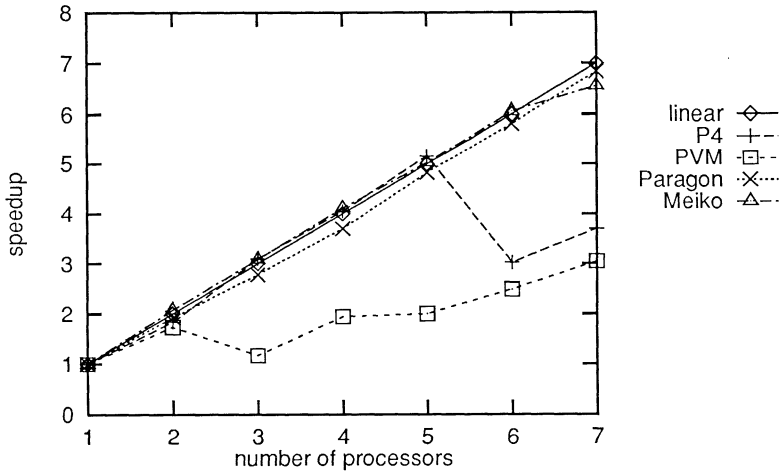


Figure 5: Finite element speedup for 8353 degrees of freedom.

the production mode. Since the code is portable with the developed mechanisms, no modification of the code is necessary. The goal of portability is achieved.

High-Performance Computing in Engineering 137

Figures 4 and 5 show finite element speedups for cases with 4257 and 8353 degrees of freedom, respectively. Speedups from the Paragon and Meiko parallel computers are close to the linear speedup for 4257 degrees of freedom and better than linear speedup for 8353 degrees of freedom. Speedups better than the finite difference analysis are obtained for P4 and PVM. This is due to the ratio of computation time versus the communication time being larger for the finite element analysis. When less than 5 workstations are used, speedups close to or better than linear speedup are obtained when using P4. When more than 5 workstations are used, the speedup decreased. This may be due to communication management of the network. This shows that it is still possible to use a cluster of workstations to perform parallel computations for computation intensive applications. Speedup for PVM is not as good as it for P4. This still shows the portability of the approach.

Note that, it is not the intention herein to compare the performance for the different systems in this study. Although the results from PVM are not as good as results from P4, it is not to say that P4 is a better system than PVM.

As discussed, the performances of using a cluster of workstations are limited by the speed of the network devices and the communication management. It is suggested that the performance on a cluster of workstations can be improved if fast network devices and better communication management are used.

4 Conclusion

In this study, it has been shown that a cluster of workstations can be used for developing parallel applications as well as performing parallel computation. Although the speedup on a cluster of workstations is not yet satisfactory, fast network devices and better communication management are suggested for further study of parallel computation using a cluster of workstations.

In addition, it has shown that it is possible to develop portable parallel codes across different parallel computing resources. By compiling with the translation mechanism on a parallel system, a portable parallel code which uses function calls from the translation mechanism can be executed without any modification.

Acknowledgements

The authors would like to thank the San Diego Supercomputer Center for providing time on its Paragon parallel computer and the Computer Science Department at the University of California at Santa Barbara for providing time on its Meiko CS-2 parallel computer which was obtained under a grant from the National Science Foundation, Award No. CDA92-16202.

138 High-Performance Computing in Engineering

References

1. Wang, K.P. & Bruch, J.C., Jr., An Efficient Fully Parallel Finite Difference SOR Algorithm for the Solution of a Free Boundary Seepage Problem, ed L. C. Wrobel & C.A. Brebbia, pp. 37-48, *2nd International Conference on Computational Modeling of Free and Moving Boundary Problems*, Milan, Italy, Computational Mechanics Publications, Southampton, U.K., 1993.
2. Wang, K.P. & Bruch, J.C., Jr., A Highly Efficient Iterative Parallel Computational Method for Finite Element Systems, *Eng. Comput.*, 1993, **10**, 195-204.
3. Wang, K.P. & Bruch, J.C., Jr., An Efficient Iterative Parallel Finite Element Computational Method, Chapter 12, *The Mathematics of Finite Elements and Applications*, ed. J. R. Whiteman, pp. 179-188, John Wiley, New York, 1994.
4. Wang, K.P. & Bruch, J.C., Jr., A SOR Iterative Algorithm for the Finite Difference and the Finite Element Methods that is Efficient and Parallelizable, *Advances in Engineering Software*, 1995, in press.
5. Butler, R. & Lush, E., User's Guide to the P4 Programming System, Technical Report TM-ANL/92/17, Argonne National Laboratory, 1992.
6. Geist *et al*, PVM 3 User's Guide and Reference Manual, Technical Report, ORNL/TM-12187, Oak Ridge National Laboratory, 1994.