PROGRAMMING TOOLS FOR DISTRIBUTED MEMORY PARALLEL SUPERCOMPUTERS

R.M. Chamberlain
Intel Corporation
Swindon, UK

ABSTRACT

The hardware for distributed memory parallel computers has made big strides in the last few years. Their great potential has been seen in a number of applications, but their more widespread use has been delayed by the less mature software for these machines. This paper describes a number of programming tools for distributed memory parallel computers and discusses the issues involved in designing such tools.

1. INTRODUCTION

The performance of traditional supercomputers has only increased very modestly over the last few years. In contrast, microprocessor performance has improved dramatically with the introduction of superscalar architectures (eg the Intel i860 (Kohn (1989)). When many of these powerful microprocessors are put together in a distributed memory architecture, they form the fastest and most cost-competitive supercomputers. For example Scott et al. (1990) achieve a sustained performance of over 1.6GFlops on only 64 Intel i860 processors for an electromagnetic application.

In a distributed memory parallel computer, there are a number (typically tens or hundreds) of processors. Each processor runs its own program using its own local memory. In order to co-operate on solving a problem, messages are sent between the processors containing data that the receiving processor requires to continue its calculations. On the iPSC/860, each processor is an Intel i860 and message passing is carried out by simple subroutine calls from standard languages like C and Fortran. This style of computation is often called MIMD (Multiple Instruction, Multiple Data). In many applications the same program is run on all processors, but on different data or scenarios. One popular way of parallelising a program is domain decomposition (eg for partial differential equations each processor works on a subdomain). Another common approach is to run the entire serial program on each processor with different initial data (eg Monte Carlo methods using random starting values).

However, serial programs do have to be modified in order to run on these machines. This perceived "difficulty to program" has frightened some users away from distributed memory machines. Certainly the challenges of debugging and optimising a parallel program can be daunting, unless sophisticated tools are available to help the task. An ideal solution would be to have a parallelising compiler which handled all these problems. However, such a compiler must analyse the whole program before deciding on data distribution. Although prototypes are being developed, it will take many years of research before general purpose parallelising compilers which make efficient use of the hardware are available.

One approach has been to develop special parallel languages. An early example is Occam, although its use is limited to the Inmos transputer. Linda (Carriero and Gelernter (1988)) and Strand (Foster and Taylor (1989)) are two languages where the same program can run without change on a serial machine, a distributed memory multiprocessor and a shared memory multiprocessor. Strand also allows the use of standard languages like Fortran as building blocks in the parallel application. While these languages have the great advantage of portability, they require the user to learn a new language - an obstacle that can be hard to overcome.

Another approach has been shared virtual memory (Li and Schafer (1989)). Here the operating system handles the movement of pages between processors and the updating of variables, so the distributed memory multiprocessor can be programmed like a shared memory multiprocessor. Automatic parallelising compilers, which split appropriate loops across processors, are available for shared memory multiprocessors. These techniques are suitable for a small number of processors in a time-sharing environment . However, for larger numbers of processors, they may not be sufficient to make efficient use of the hardware. In particular, the parallelism may well be in the application, (eg from a geometric decomposition) rather than in the algorithm in the serial program. Thus restructuring of the code or the choice of another algorithm may be necessary to make good use of the hardware.

There is a need to help programmers use standard programming languages (like C and Fortran) with extensions for message passing for distributed memory supercomputers. Particular areas for help are debugging tools, performance analysis to increase efficiency and tools to help port "dusty-deck" codes. Many other research groups are working on programming tools and include TOPSYS (Bemmerl (1990)), SUPERB (Kremer et al.

(1988)) and Hypertool (Wu et al. (1989)). In this paper we describe tools in this area for the Intel iPSC/860 and discuss the issues involved in the design of these tools.

2. DEBUGGERS

The initial consideration for the programmer is just to get the program running and giving the correct answers. Within a parallel context, there is more scope for error and an inexperienced user may have difficulty tracking down bugs. Besides the ordinary bugs in the serial programs (which can be on one or all nodes), there is scope for message passing errors (missing messages, wrongly directed messages, etc.).

Pan and Jackson (1988) describe DECON, the interactive concurrent debugger for the iPSC/2 and iPSC/860. One important feature of DECON is the context. By changing the context interactively the user can look at the whole application or any subset of it (down to a single program on one processor). Within a context, the programmer can stop programs, examine and change data structures. Thus debugging each program (or set of programs) can be done in essentially the same way as for one serial program.

As well as these features for debugging at the source-level, DECON has various facilities for identifying message-passing bugs. At any point where a program is stopped (which may be at different places on different processors), the user can examine the messages queued at a node and can list any messages the program may be trying to receive. This information, together with a display of the location in the program where the message passing problem is happening, allows the programmer to detect easily and quickly most, if not all, message related bugs.

DECON is run as one interactive program. A workstation can be used to run DECON in one window while other information (eg the source code for programs) can be displayed in other windows. Interface features are being investigated further to provide the most user-friendly interface.

3. PARALLELISATION TOOLS

The ideal tool for parallelising an existing serial code is an automatic parallelising compiler. Compilers that make efficient use of the hardware are still many years away so in the meantime the programmer has to restructure the program. Perhaps in the absence of the original author of the code, the programmer requires sophisticated tools to aid the analysis and restructuring of the programs.

Two such tools are FORGE and CAST (formerly MIMDizer) developed by Pacific Sierra Research Corporation (Hill (1990)). FORGE, available now for several systems, parses the serial FORTRAN program and builds a database. The database contains descriptions of all variables and records of all dependencies. FORGE can also insert timing calls into the program so the database can include information on the percentage of time spent in any part of the program. Thus the user can see the parts of the program to concentrate on (and more importantly, the sections that do not require parallelisation). CAST, the tool for restructuring the code, is built on FORGE. From the database, the user can extract all instances of arrays that the user may want to distribute across processors. CAST is interactive, so the programmer decides on the strategy for parallelisation (eg splitting DO loops, breaking up into independent program blocks, etc.). CAST then handles the clerical tasks of this restructuring, avoiding possible errors that may be made by programmers.

FORGE and CAST are almost entirely menu- and mouse-driven. As well as being completely user-friendly, this enables the user to move easily and rapidly around the database. For the future, CAST will be developed further to add more facilities and "intelligence" to the restructurer.

4. PERFORMANCE ANALYSIS

Using the interactive parallel debugger, FORGE and CAST, the programmer can get his application running on the iPSC/860. During this process, the user may have used FORGE to get a breakdown of the timing for the serial program. Before the program is used in production or for long runs, the programmer may want to tune the program for optimal performance. In order to do this timing data for the running of the parallel program is required.

Several issues arise. Should this monitoring be done in hardware or software? Bemmerl, Lindhof and Treml (1990) have developed a hardware monitor, which is currently being tested. Hardware monitoring is less flexible, but a software monitor adds to the overhead of the program and may change its behaviour. When many processors are being used, large quantities of performance data may be gathered and require output for analysis. Another issue is synchronising events in a parallel environment. Typically each processor has its own independent clock, so time-stamping events (eg the sending of a message) may be necessary to understand the behaviour of the program.

Intel is providing a software monitor and analysis tool, PAT. PAT produces CPU usage histograms, event monitoring of multiple processes and internode communication tracing. PAT's graphical interface uses time-lines, graphs and bar-charts to give programmers greater insight into their application.

5. LINEAR EQUATION SOLVERS

Techniques for parallelising standard numerical algorithms are already well-established in a number of areas. Therefore it is pointless for the user to write his own code for algorithms where parallelisation procedures are well-known. Libraries of such routines are most productively written and optimised by experts. Solving large sets of linear equations is a critical part of many numerical applications and this is where Intel has started to provide such library packages.

Three solvers are included in the ProSolver package. The Skyline Equation Solver is a direct, in-core or out-of-core solver for sparse matrices. The elements of the matrix are stored optimally. The solution of the equations is based on an inner-product formulation of the factorisation to achieve performance of up to 20 MFlops per processor. The solver also includes a data capture module that relieves the programmer of the burden of storing and decomposing the coefficient matrix and right-hand sides. Suitable applications for the Skyline Equation Solver include the implicit solution of partial differential equations (eg in structural mechanics, circuit simulation and hydro codes).

The second solver, the Iterative Equation Solver, is a fully sparse solver that stores and processes only the non-zero elements of the matrix. Both 32-bit and 64-bit precision are handled. It shares a common interface to the data capture routines in the Skyline Equation Solver. This enables programmers to switch easily between direct and iterative sparse solvers. Iterative solvers are often appropriate for applications such as fluid mechanics.

The third solver, The Dense Equation Solver, is useful when dense matrices are generated, such as in integral equation methods. It can process matrices well beyond the memory capacity of any known computing system. The solver is optimised both for the parallel calculations and the I/O management to achieve sustained performance of more than 1.6GFlops on 64 Intel i860 processors. For long-running applications, checkpointing is available so restarting can easily be done.

6. NETWORK FACILITIES

During the development cycle the programmer normally desires to use a familiar environment (eg SUN workstation, VAX VMS, etc.) including tools, graphics, windows, etc. Certainly in production, the user requires the familiar interface and does not care what hardware the code is running on.

In this field, it is sensible to keep to industry-standard connections. TCP/IP Ethernet is currently the most widely used and this connection is available to Unix systems. Non-Unix systems such as DEC VAX/VMS and IBM MVS computers can be reached through gateways. An emerging standard is FDDI, the progress of which is being followed. The Network File System (NFS) is provided to allow access to files across the network and to the parallel processing nodes.

Remote Host facilities are available which enable users to develop and run codes on the iPSC/860 without logging in to the machine. Applications running on the workstation can communicate with the iPSC/860 processing nodes so the graphical user interface continues to run in the familiar environment X Window System client services are provided on the iPSC/860 nodes to communicate with workstations over the network.

As a network server, batch-style processing is available through the Network Queuing System (NQS). Jobs are submitted in the form of Unix shell scripts or command files and the resources of the parallel machine can be partitioned (so a part of the machine could be dedicated to large tasks). Priorities can also be assigned to jobs. The provision of network facilities is governed by the existing industry standards. Thus, as standards change the services will be updated.

7. CONCLUSIONS

The potential of distributed memory parallel supercomputers has already been demonstrated in a number of applications. The emergence of tools to aid programming these machines is making that potential more readily attainable. These tools are being refined and improved with the goal of making distributed memory parallel supercomputers no more difficult to program than serial machines. Then supercomputer users, whose job is to do science, can use the full power of these machine.

REFERENCES

Bemmerl, T., 1990: The TOPSYS Architecture, CONPAR 90 - VAPP IV, Springer Verlag, 732-743.

Bemmerl, T., Lindhof, R. and Treml, T., 1990: The distributed monitor system of TOPSYS, CONPAR 90 - VAPP IV, Springer Verlag, 756-765.

Carriero, N. and Gelernter, D., 1990: Applications experience with Linda, Proceeding of the ACM/SIGPLAN PPEALS.

Foster, I. and Taylor, S. 1989: Strand: New concepts in parallel programming, Prentice Hall, 1989.

Hill, R.H., 1990: MIMDizer A New Tool for Parallelisation, Supercomputing Review, April 1990, 26-28.

Kohn, L., 1989: Architecture of the Intel i860 64-bit Microprocessor, Spring Compcon.

Kremer, U., Bast, H.-J., Gerndt, M., and Zima, H.P., 1988: Advanced tools and techniques for automatic parallelisation, Parallel Computing, 7, 387-393.

Li, K. and Schafer, R., 1989: A hypercube shared virtual memory system, IEEE Parallel Processing Conference, St Charles, pt. I, 125-132.

Pan, W.M. and Jackson, V., 1988: A concurrent debugger for iPSC/2 programmers, Proc. Third Conf. on Hypercube Concurrent Computers and Applications, ACM Press, 823-831.

Scott, D.S., Castro-Leon, E. and Kushner, E., 1990: Solving very large dense systems of linear equations on the iPSC/860, Proc. 5th Conference on Distributed Memory Computers (DMCC5), Charleston, NC.

Wu, M.-Y. and Gajski, D.D., 1989: Hypertool: A programming aid for multicomputers, Proc. 1989 Int. Conf. on Parallel Processing, Penn State Univ. Press, II, 15-18.