

CONF-970342--

**Software Tools for Developing Parallel Applications
Part 1: Code Development and Debugging***

Jeffrey Brown

Los Alamos National Laboratory
Los Alamos, New Mexico

Al Geist

Computer Science and Mathematics Division
Oak Ridge National Laboratory
P.O. Box 2008, Bldg. 6012
Oak Ridge, TN 37831-6367

Cherri Pancake

Department of Computer Science
Oregon State University
Corvallis, Oregon

Diane Rover

Department of Electrical Engineering
Michigan State University
East Lansing, Michigan

"This submitted manuscript has been authored by a contractor of the U.S. Government under Contract No. DE-AC05-96OR22464. Accordingly, the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes."

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

MASTER

*This work was supported in part by the National Science Foundation, Science and Technology Center Cooperative Agreement No. CCR-8809615 and the Mathematical, Information, and Computer Sciences subprogram of the Office of Energy Research, U.S. Department of Energy, under contract No. DE-AC05-96OR22464 with Lockheed Martin Energy Research Corporation.

DTIC QUALITY INSPECTED 1

19980528 036

RECEIVED

MAY 04 1998

OSTI

36

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

Software Tools for Developing Parallel Applications

Part 1: Code Development and Debugging

Jeffrey Brown* Al Geist† Cherri Pancake‡ Diane Rover§

Abstract

Developing an application for parallel computers can be a lengthy and frustrating process — making it a perfect candidate for software tool support. Yet application programmers are often the last to hear about new tools emerging from R&D efforts. This paper provides an overview of two focuses of tool support: code development and debugging. Each is discussed in terms of the programmer needs addressed, the extent to which representative current tools meet those needs, and what new levels of tool support are important if parallel computing is to become more widespread.

1 Introduction

The demands of scientific computing have driven hardware vendors to provide ever more powerful and complex computing platforms. One consequence of this market demand is that vendors have less and less time to mature the system software and tools for new platforms. The situation has given rise to a paradox that affects many application programmers. Early releases of software tools, developed quickly and sometimes concurrently with the hardware on which they will run, are notoriously fragile. They may be incapable of accommodating programs of any real complexity, despite the fact that the machines themselves are intended to support ever larger and more complex programs. Early users must bear the brunt of product immaturity; they often spend as much time debugging the tool as they do applying it. By the time a tool reaches maturity, however, it is two or three years into its life cycle. The platform for which it was designed may already be waning, and at least some of the tool capabilities will have been superseded by newer technological developments. Thus, although late users enjoy improved stability and usability, it is at the cost of lagging one or two generations in tool technology.

Are the advantages of new tool innovations worth the pain involved in being an early user? This paper compares the kinds of tool support available in relatively mature tool products with those of newer experimental approaches. In some cases, parallel programmers who “do without tools” or who wait for them to mature may be expending more time than they would if they served as early users, providing feedback to tool developers and shaping the direction of parallel tool support.

The discussion is organized in four sections, reflecting the general life cycle of parallel applications. Developing the parallel program code — whether starting from scratch or

*Los Alamos National Laboratory, Los Alamos, NM.

†Oak Ridge National Laboratory, Oak Ridge, TN.

‡Department of Computer Science, Oregon State University, Corvallis, OR.

§Department of Electrical Engineering, Michigan State University, East Lansing, MI. Assisted by Nayda Santiago, Abdul Waheed, and Kenneth Wright.

by modifying an existing serial code — is the first area where tool support can be applied to reduce programmer effort. Section 2 provides a quick overview of user requirements for tools, describes representative examples of current tool offerings, and some promising new directions. The next section provides a similar description of how tools can support the programmer's efforts to establish whether or not the program is performing as he or she intended. This is followed by a discussion of runtime support, where tools can be used interactively to control how program results are delivered. Section 5 discusses tool support for improving the application's performance, typically of concern once acceptable levels of correctness and appropriateness have been achieved. It should be noted that these steps are often interrelated and occasionally even simultaneous; the division here is largely a matter of convenience.

One problem with current tools is that virtually all of them are constrained to just certain computers, certain programming languages, and certain programming situations. In the parallel tools research community, however, increasing attention is being devoted to more generalized approaches. Where appropriate, emerging tools that span multiple architectures and programming paradigms are singled out.

2 Development Tools for Parallel Applications

The process of developing the code for a parallel application involves a number of repetitive, error-prone tasks that could be facilitated greatly through tool support. Although this characteristic applies regardless of which programming model is employed, we find that in practice, the nature of tool requirements varies substantially according to the paradigm [9].

Message passing is the best understood model for developing parallel applications. It offers two distinct advantages: the logic is straightforward to understand (although quite tedious to implement), and message passing is available on virtually all parallel computers, including networks of workstations. The dominant examples are PVM and MPI. PVM (Parallel Virtual Machine) [4] is popular for distributed computing, as it can function in heterogeneous environments (i.e., those mixing machines of multiple sizes and architectures). The PVM library allows a collection of computers connected to a network to be used as though it were a single message-passing parallel computer, providing users with a uniform parallel programming interface. Since PVM is developed and maintained as shareware, performance is not optimal on all platforms, but it provides a relatively uncomplicated, easy to learn model for developing parallel applications. In some ways, MPI (Message Passing Interface) [8] is the antithesis of PVM. It furnishes a large number of message operations so that different hardware vendors can tune the library to their particular architectures, and was really designed for use by compiler writers. An application programmer, then, must either choose the more generic operations and possibly sacrifice performance, or tune the application to a particular platform by selecting specialized operations. MPI is supported by virtually all parallel computing vendors, but since each vendor implements it differently, an MPI application cannot execute in a heterogeneous environment (or even across different systems from the same vendor).

The data-parallel programming paradigm, typified by HPF [11], moves programming to a higher level of abstraction. The programmer inserts compiler directives describing how data is to be laid out and accessed, and uses Fortran 90 array operations to describe where computations can proceed in parallel. The compiler assumes responsibility for distributing the data across multiple processors and communicating it back and forth among them using messages. HPF is now available for most parallel machines, but it does not support

heterogeneous computing.

Parallel object-oriented languages raise the level of abstraction another notch, introducing parallelism through special class libraries. Perhaps the best known is pC++, a programming system distributed by Indiana University, University of Oregon, and University of Colorado [7]. It has been ported to many parallel systems, and it has evolved into the HPC++ environment, which supports network-based (i.e., coupling geographically distributed computing and data resources) as well as heterogeneous applications.

The newest programming systems provide comprehensive problem-solving environments for specific application domains. One example is MultiMATLAB [12], developed at the Cornell Theory Center, which makes it possible to execute MATLAB problems across multiple, heterogeneous computers, using MPI to control their activities. MultiMATLAB also provides access to the functions in MATLAB's toolboxes (e.g., the image processing toolbox).

Clearly, the types of tool support needed to develop a MATLAB or other domain-specific program are very different from those experienced by a programmer whose "parallel language" consists of calls to subroutines controlling primitive message operations. Since message passing and data parallelism are the models employed by the overwhelming majority of today's parallel programmers, however, the remainder of this section discusses the needs of these two audiences.

2.1 Basic Requirements for Code Development Support

To develop a successful parallel application, the programmer must understand the interactions and interdependencies among the application's component parts. Much of the real effort is directed at identifying how data is used during computational steps. On serial platforms, the failure to provide good data locality results in poor performance, but the desired results are still produced. On a parallel system, improper data locations result in errors that can be very hard to find.

Requirements for tool support, therefore, focus on program analysis, a time-consuming and tedious procedure which is relatively easy to automate but is extremely error-prone when performed manually. A simple example is checking for the consistency of subroutine and function calls, in terms of the number and type of arguments, particularly for calls to parallel libraries. More comprehensive program analysis techniques yield information on how individual program variables are used: which variables are used globally (i.e., by other routines); which are aliased or re-named in other parts of the program; which arrays are re-dimensioned, and when; the order in which array elements are accessed, and whether access is limited to certain sub-portions of the array; whether variables are read-only or also modified, and are which points in the program; etc. Such analysis must be able to cross subroutine boundaries, so that it is possible to analyze patterns of use throughout the entire application.

Other requirements arise because parallel applications involve potentially huge collections of files and directories. Tools are needed to analyze the entire collection and produce data such as maps of make dependencies (i.e., the order in which files need to be compiled or linked when one is changed) or "differencing trails" that track which portions of which files were modified and at what points in time.

Automatic code restructuring is another area where tools can be of significant help in code development. In the simplest case, a restructurer could simply carry out comprehensive or repetitive tasks, such as variable renaming, subroutine inlining, or

reorganization of data structures. Restructurers capable of analyzing the flow of data and control through the program, and using that information to rearrange code or insert parallelizing code, would not only reduce the time needed for application development, but also reduce the possibility for introducing errors.

2.2 Examples of Current Code Development Tools

Although the techniques for analyzing source code are well understood, and in fact form the basis for current compilers, there are few tools that report analysis information back to the user in any useful way. Serial program tools like `lint` or dependency analysis tools such as ParaScope's PED are available for some programming languages, but they are limited in terms of the size or complexity of program that they can handle.

The most comprehensive tool currently available is FORGE [1], developed as part of a programming environment for converting Fortran77 programs to HPF. The program analysis tools, which can handle large and reasonably complex program structures, can be used to study the flow of data through the application, even if the target language is message passing rather than HPF.

No tool currently available on parallel computers has addressed the problems of organizing and managing the construction of applications with extensive file structures.

2.3 Looking Ahead

Code development activities — fixing the original serial program (if there was one) in preparation for parallelism, and actually coding the parallel portions — accounts for some 40% of all programmer effort in developing a parallel application, according to user surveys [10]. It is surprising, therefore, that so few tools are available to support these activities. Nor have any of the traditional parallel hardware or software vendors announced efforts likely to yield new tools in the near future. Further, while automatic parallelization is still the focus of some research efforts, the kinds of program analysis that would be of direct benefit to parallel application programmers are no longer being pursued in the research setting.

3 Debugging Large-Scale Applications

In order to obtain the fidelity necessary to effectively model physical phenomena, modern large scale scientific programs are employing fine grained three dimensional grids using terabytes of memory distributed over thousands of processors. Verifying program correctness and tracking down bugs over such a large compute space requires debugging tools that aggregate data and processes in order to reduce the complexity presented to the code developer. Traditional serial debugging techniques — such as examining the value of a variable or stopping the process — require parallel extensions that scale to meet the needs of current tera-scale code development.

3.1 Basic Requirements for Debugger Support

Integrated two and three dimensional data visualization needs to become a routine debugger feature for scientific computing in order to gain a high level feel for the correctness of the data. The debugger needs to be able to effectively handle very large programs with millions of symbols while providing reasonable response time to the user. A parallel debugger should handle both threads and message passing, possibly mixed in the same program. The identical processes in a single process multiple data (SPMD) program need to be treated

as a group through the availability of process sets. If a preprocessor is used at compile time, the debugger should present the user with the original source code. Popular modern languages for scientific computing such as C++ and FORTRAN 90, possibly mixed, need to be fully supported. Fast conditional breakpoints and watchpoints need support and integration with process sets. Finally, the debugger needs to span the entire compute space of interest, even if it is heterogeneous.

3.2 Examples of Current Debugging Tools

Here, the capabilities of three typical commercial offerings (TotalView, dbx, and PRISM) and two experimental debuggers (P2D2 and ldb) are summarized, from the viewpoint of application programmers involved in DOE's Accelerated Strategic Computing Initiative (ASCI) program.

TotalView [3] is a commercial parallel debugger marketed by Dolphin. (Although there is a derivative of TotalView provided by Cray Research for their MPP and PVP product lines, these comments apply to the Dolphin product.) TotalView is a capable multi-platform debugger with many of the features needed. Data visualization, while not yet available, is currently under development, as is support for the other platforms (IBM, Intel, and SGI) required for the ASCI program. TotalView needs better support for data and process aggregation and lacks a command line interface. Missing features will likely be developed as part of the ASCI program as TotalView has been selected as the ASCI multi-platform debugger of choice as part of a common development environment.

Dbx is the de-facto standard serial debugger. Debugging message-passing based parallel programs often becomes a task of managing separate dbx debuggers, each in its own window and controlling a single process. Dbx provides a familiar command line interface and reasonable support for serial C programs, but lacks parallel support and is not especially Fortran friendly.

The PRISM debugger was developed to debug data parallel programs on the TMC Connection Machine. PRISM remains the only debugger to provide the data and process aggregation features needed for large scale scientific code development. PRISM is part of the Global Works product recently purchased by Sun Microsystems. PRISM still comes closest to the ideal parallel debugger, but the fact that it does not run on recent generations of parallel computers is a serious problem for computing centers.

The NASA Ames portable parallel distributed debugger [5, 6] is a small research and development project to provide a multi-platform debugger to support heterogeneous debugging. The P2D2 specification defines a common protocol between a portable debugger client and debugger servers that reside on distributed compute platforms. A goal of the P2D2 project is to encourage hardware vendors to provide optimized debugger servers that would provide objects to the client over the common protocol. To date this has not happened. The reference P2D2 implementation is based on gdb node debuggers to provide debugger server functionality. Thus, P2D2 is limited to capabilities provided by gdb. Since gdb does not support FORTRAN 90, P2D2 doesn't either. P2D2 supports MPI message passing and thread support is in development. The notion of a common protocol could accelerate the development of common debugger features, but has not as yet caught on. P2D2 provides many innovative features but requires more resources to become a full functioning debugger.

The Los Alamos Debugger, ldb [2], was designed and developed by the computing division at Los Alamos National Laboratory to support the development of large scale

FORTRAN programs on Cray PVP platforms. Ldb features a code patching mechanism to provide fast conditional breakpoints and watchpoints. The ldb command line interface is based on the ddt debugger which was tied to the CTSS operating system. Ldb is very Fortran friendly but lacks C and parallel support, and lacks a graphical user interface. Ldb provides macro support and integrated performance analysis. Like P2D2, ldb could evolve into a full-functioning parallel debugger with more resources, but current LANL strategy favors collaborative R&D with commercial software providers. Specific useful ldb features (e.g. code patching) may find their way into commercial products such as TotalView through ASCI collaborations.

3.3 Looking Ahead

There has been considerable effort put into the research and development of parallel debuggers, with mixed results. To date, there is no single debugger that provides all the features named previously as requirements.

The need for solid parallel debuggers is growing. The currently popular message-passing programming models introduce new levels of complexity and non-determinism that make the tasks of verifying program correctness and debugging much more difficult. Higher quality and more fully functioning parallel debugging tools are required to fully exploit the power of this new generation of compute platforms.

In particular, all the needed functionality must be provided in a single debugger that runs on multiple parallel computing platforms. There are two paths to accomplishing this: collaborative R&D, and the proliferation of debugger standards. Collaborations such as the ASCI program will pool resources and focus requirements resulting in a full functioning debugger. The other path means that the user and tools communities must come together and define a set of standards by which debuggers are measured. Just as users can count on a standard set of functionality in a compiler, future debuggers adhering to a standard could provide a common set of capabilities that users can count on. The emerging High Performance Debugging Forum (sponsored by the Parallel Tools Consortium) is an attempt to pull the debugging community together to pursue common debugging standards and facilitate the development of debugging technology.

References

- [1] Applied Parallel Research, *FORGE Explorer User's Guide*, APR, Placerville CA, 1995.
- [2] J. Brown, R. Klamman, J. Peterson, *LDB Reference Manual*, Proceedings of Supercomputer Debugging Workshop, October 1992, http://www-c8.lanl.gov/dist_comp2/LDB/ldb.html.
- [3] Dolphin Toolworks, <http://www.dolphinics.com/toolworks/>.
- [4] G. A. Geist, et al., *PVM: Parallel Virtual Machine — A Users Guide and Tutorial for Network Parallel Computing*, MIT Press, 1994, <http://www.epm.ornl.gov/pvm>.
- [5] R. Hood, et al., *A Portable Debugger for Parallel and Distributed Systems*, Proceedings of Supercomputing '94, Nov. 1994.
- [6] R. Hood, et al., *Accommodating Heterogeneity in a Debugger for Distributed Programs*, Proceedings of HICSS '95, Jan. 1995, <http://www.nas.nasa.gov/NAS/Tools/Projects/P2D2>.
- [7] A. Malony, B. Mohr, P. Beckman, and D. Gannon, *Program Analysis and Tuning Tools for a Parallel Object Oriented Language: An Experiment with the TAU System*, in Debugging and Performance Tuning for Parallel Computing Systems, ed. M. Simmons, A. Hayes, J. Brown, and D. Reed, IEEE CS Press, 1996, <http://www.cs.uoregon.edu/paracomp/tau>.
- [8] Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard*, International Journal of Supercomputing Applications, 8 (4), 1994, <http://www.mcs.anl.gov/mpi>.

- [9] C. Pancake, *Multithreaded Languages for Scientific and Technical Computing*, Proceedings of the IEEE, 81 (2), pp. 208-304, 1993.
- [10] C. Pancake and C. Cook, *What Users Need in Parallel Tool Support: Survey Results and Analysis*, Proceedings Scalable High Performance Computing Conference, May 1994.
- [11] H. Richardson, *High Performance Fortran: History, Overview and Current Developments*, Thinking Machines Corporation, 1996 <http://www.crpc.rice.edu/HPFF/publications.html>.
- [12] A. Trefethen, V. Menon, C. Chang, G. Czajkowski, C. Myers, and L. Trefethen, *MultiMATLAB: MATLAB on Multiple Processors*, Technical Report TC96TR239, Cornell Theory Center, May 1996.

M98004821



Report Number (14) ORNL/CP--97321
CONF-970342--

Publ. Date (11) 199703
Sponsor Code (18) ^{DOE} ER; NSF; , XF
UC Category (19) UC-405; UC-000, DOE/ER

DOE