

CONF-970342--

**Software Tools for Developing Parallel Applications
Part 2: Interactive Control and Performance Tuning***

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

19980528 039

RECEIVED
MAY 04 1998
OSTI

"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

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 2: Interactive Control and Performance Tuning

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

Abstract

This paper continues the discussion of parallel tool support with an overview of the current state of tools for runtime control and performance tuning. 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 Runtime Tools for Scientific Applications

Our focus in this section is on runtime tools that are designed to be used by scientists and whose purpose is to improve or enhance the way computational science is now done. Runtime tools provide users access to information about and within running applications. Popular examples of such tools include computational steering and interactive visualization.

Interactive visualization has become more and more important over the last few years because the sheer volume of data generated by computational experiments requires some method for the scientist to interactively explore the results. By moving this capability into the runtime environment, the scientist is able to watch the evolution of the computation and determine much earlier if the expected physics is being properly simulated.

Computational steering, which provides feedback during execution, has the potential to revolutionize computational experiments by allowing scientists to interactively steer a simulation in time and/or space. Computational steering helps a scientist or engineer to concentrate more on the science and less on the computer. Through the use of interaction, the computer becomes a more useful tool to the engineer, allowing experimentation and real time exploration of a design space. This provides a much more flexible framework for science than the typical simulation cycle — manually setting input parameters, computing results, storing data off to disk, visualizing the results via a separate visualization package, then starting again at the beginning.

Computational steering and interactive visualization tools are further complicated when the application is parallelized and the data and computation are distributed across many processors. Because these parallel runtime tools have the potential of being used on every run, rather than just to initially debug or tune an application, it is important that the tools be designed with scientists in mind.

*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.

1.1 Basic Requirements for Runtime Support

Scientists want to have computational steering and interactive visualization in their applications. They see the advantages these capabilities bring to their ability to do science better. At the same time they want these capabilities to be easy to incorporate and easy to customize to the views and steering parameters unique to their application. It is important that the visualization tool be able to dynamically attach and detach from a long running application. The scientist also wants to make sure the incorporation of computational steering and interactive visualization have little or no impact on the performance of the application when the visualization tool is detached. Further, he or she wants the impact minimized (or user controllable) when the application is being steered.

1.2 Examples of Current Runtime Tools

There is no widely-used runtime visualization tool for parallel programs. Applications with this capability have often had to reinvent much of the synchronization and data collection algorithms required to create time coherent views from distributed asynchronously created data.

1.3 Looking Ahead

We present three new experimental projects to develop runtime packages that provide interactive visualization and computational steering capabilities to general parallel applications. Each of these packages allows client "viewers" to dynamically attach and detach from the running application, and all handle the details of collecting and sending distributed data fields to and receiving steering parameters from a viewer.

Oak Ridge National Laboratory is developing a system called CUMULVS [1], which allows scientists to easily incorporate fault tolerance, interactive visualization and computational steering into distributed applications. Built upon the PVM [2] virtual machine, CUMULVS is portable to all the architectures that PVM runs on. The CUMULVS system supports collaborative computational experiments through multiple dynamically attached viewers. These viewers are independent and can be commercial packages such as AVS or customized viewers for a specific application. All data collection and control in CUMULVS is done asynchronously to minimize intrusion. An additional runtime feature available through CUMULVS is automatic fault detection and checkpoint/restart in a heterogeneous environment. CUMULVS can even reconfigure a checkpoint file for a different number of processors if a replacement processor can not be found.

MIT is developing pV3 [4], which is a visualization software system for three dimensional, distributed, unsteady, unstructured, CFD calculations. While pV3 is primarily used in CFD applications, it is general enough to be used in a wide range of applications. Built upon PVM, pV3 can run on clusters of workstations as well as supercomputers. The unique feature of pV3 is that the visualization calculations are co-processed on the parallel computer. This significantly reduces the network bandwidth required for animations at the cost of sharing computational power and memory between the visualization process and the application process. By performing the visualization calculations in parallel, pV3 can rapidly display isosurfaces, streamlines, and other features from very large datasets. Since the visualization processes need to take advantage of customized hardware, there is a number of restrictions on what hardware platforms pV3 is supported on. A single client "viewer" is allowed per application. The viewer is based on OpenGL and is supported on DEC, HP, IBM, SGI, and SUN workstations.

Georgia Tech is developing Falcon [3], which is a set of tools and libraries supporting runtime monitoring and steering of parallel and distributed applications. Falcon consists of a sensor specification language and compiler for generating application sensors, local agents for collection of event data, and steering middleware that allows multiple dynamically attached clients to interact. Falcon uses the OpenInventor graphical display framework for its collaboration infrastructure. Each of the clients is independent and each collaborator can customize his views to match his interest. The Falcon software has been tested on SUN, SGI, and IBM workstations, as well as the SP-2 and KSR.

2 Tools for Analyzing Parallel Performance

Performance analysis tools provide users with information about system and application behavior during execution. The information comes in diverse forms depending on what is being monitored and how it is being monitored. One of the most challenging aspects of performance analysis is presenting the information at a level of abstraction that is meaningful to the scientist. Moreover, tools should provide the scientist with some insights on tuning the application. The role of performance analysis tools is complicated by emerging computing and problem-solving environments. In this section, we consider the implementation of an application in several environments and highlight performance analysis tool options, particularly in support of high-level information.

As usual, the rapidly changing landscape of high-end computing systems and paradigms complicates the process. Even before programmers are faced with navigating through information provided by a particular analysis or visualization tool, a number of steps must be taken to collect, store, process, and present the information. The tools available at each of these steps impact the ability of programmers to effectively tune an application.

2.1 Basic Requirements for Performance Analysis Support

As stated by Hollingsworth et al. [6], performance analysis tools exist to provide insight to programmers to help them understand why their programs do not run fast enough. For these tools to be effective, they need to collect data about the application, the operating system, and the hardware and to synthesize it in a way to let programmers concentrate on getting their work done. Users of high-level parallel programming languages need performance information that is accurate and relevant to the programming model and the source code.

2.2 Examples of Current Performance Analysis Tools

Most commercial parallel systems provide a platform-specific set of tools and utilities for profiling and tracing program execution, typically including some source-level information. Examples include: CrayTools (including MPP Apprentice and ATEExpert); Silicon Graphics SpeedShop and Co-Pilot; Hewlett Packard CXpa; and IBM UTE and VT. The environments described in the following subsections represent a spectrum from what is presently available to the user to what is on the horizon, in terms of support for performance information associated with three different programming paradigms.

2.2.1 In Message-passing Environments PVM (Parallel Virtual Machine) [2] is a widely used environment for distributed computing. The PVM package allows a collection of computers connected to a network to be used as a single message-passing parallel computer. It is portable across many different computers and provides users with a common parallel programming interface.

XPVM is a graphical user interface for PVM that displays both real-time and post-mortem animations of message traffic and machine utilization by PVM applications [2]. While an application is running, XPVM displays a space-time diagram of the parallel tasks showing when they are computing, communicating, or idle and animating messages between tasks. XPVM stores events in a trace file that can be replayed, stopped, and stepped to analyze the behavior of a completed execution. The trace file written by XPVM is in SDDF format as defined in Pablo [9]. The format also can be converted to PICL format for use with ParaGraph [5]. Pablo and ParaGraph provide a large number of visualizations, so the number of displays available in XPVM is small, with particular relevance to the PVM environment. The performance of the PVM package itself continues to be enhanced by its developers.

Pablo and ParaGraph are trace-based visualization tools for performance tuning of parallel applications. An alternative tool for measuring the performance of large-scale parallel programs in the PVM environment is Paradyn [8]. Paradyn supports networks of workstations running PVM, works well with several hardware platforms (e.g., Sun SparcStation, HP PA-RISC, and IBM RS/6000 and SP-2), operating systems, and programming models, and measures programs running on heterogeneous combinations. Paradyn dynamically instruments an application (inserting and removing instrumentation automatically) as it searches for performance bottlenecks. Paradyn's Performance Consultant uses the W^3 search model to assist the user in locating program performance problems on the basis of three questions: Why is the application performing poorly? Where is the performance problem? When does the problem occur? Paradyn allows high-level language programmers to view performance in terms of high-level objects (such as arrays and loops for data-parallel Fortran) and maps high-level information to low-level objects (such as nodes and messages). Paradyn also provides a set of standard visualizations (time histograms, bar graphs, and tables) and an interface to use displays from other sources (e.g., ParaGraph, Pablo, or AVS).

2.2.2 In Object-Oriented Environments TAU (Tuning and Analysis Utilities) is a visual programming and performance analysis environment for pC++ [7]. The TAU graphical interface represents objects of the pC++ programming paradigm: collections, classes, methods, and functions. These language-level objects appear in all TAU utilities. TAU uses the Sage++ toolkit as an interface to the pC++ compiler for instrumentation and accessing properties of program objects. TAU provides profiling and tracing support. The TAU tools act in concert, and each tool implements some well-defined tasks; a tool can request a feature of another tool. Global features can be executed in all currently open TAU tools. The program and performance analysis environment includes tools for accessing static information about the program and for querying and analyzing dynamic data obtained from program execution; it also includes interfaces to stand-alone performance analysis tools from other sources (e.g., Pablo, Nupshot).

2.2.3 In a MultiMATLAB Environment MultiMATLAB is one of several projects to develop a high-performance computing version of MATLAB. The analysis tools for these environments are not yet mature. Problem-solving environments pose the greatest gap between low-level performance information and programming abstractions. Current options for analyzing the performance of MultiMATLAB applications are limited. One option is to use the MPI profiling interface to provide an MPI trace library for MultiMATLAB. This, coupled with visualization tools such as Pablo, Nupshot, or ParaGraph, lets the user view low-level behavior. The IBM SP-2 version of MultiMATLAB can take advantage of IBM's unified trace environment (UTE) [10] or its visualization tool (VT). These approaches

require re-linking or re-compiling for trace generation. Michigan State University is collaborating with the Cornell Theory Center to support high-level performance analysis in the MultiMATLAB environment.

2.3 Looking Ahead

The advantages of high-level programming environments will be fully realized only if programmers are able to understand and tune the performance of their applications based on high-level performance information. In part, this will require greater integration of compilation and performance analysis processes. Additionally, enhancements for developing application-specific instrumentation and visualizations will assist programmers, as will search-based tools that support performance diagnosis strategies. Tools also must support automated analyses, for example, of multiple executions of a program or of program scalability (without requiring users to develop extensive program and/or system models). The closer tools come to mapping low-level performance data to abstract program information, the closer they will be to prescribing solutions for performance problems.

3 Conclusions

Of the four areas of tool support discussed in this paper, only debugging and performance analysis include enough current offerings that an application programmer is likely to find tools regardless of what parallel machine he or she uses. Even in these cases, the tools vary widely from one machine to another, and most address only a subset of the real user requirements. Interactive runtime tools are an emerging area of interest that is likely to yield significant innovation over the next few years. The fourth area, support for code development, appears to be receiving less and less attention, in spite of the fact that it accounts for a significant proportion of user efforts.

A promising aspect of the current situation is that the tools community is beginning to respond to user demands for greater consistency and interoperability across machine boundaries. The increasing use of standard file formats (such as SDDF for recording performance data) and new efforts to standardize tool interfaces (chiefly the Parallel Tools Consortium) reflect this long-overdue change.

Finally, the growth of the World Wide Web has made it easier to share information on tool development and user experiences with parallel tools. These new resources include:

- <http://www.ptools.org>: various tool projects sponsored by the Parallel Tools Consortium
- http://nhse.cs.utk.edu/sw_catalog: catalog of shareware, including a number of parallel tools, sponsored by the National HPCC Software Exchange
- <http://www.nhse.org/ptlib>: results of evaluations of parallel tools, sponsored by the National HPCC Software Exchange
- <http://www.nero.net/~pancake/SSTguidelines>: guidelines for writing system software and tools requirements in procurements, results of a task force sponsored by the National Coordinating Office for HPCC

Given the time- and effort-intensive nature of parallel application development, improvements in the quality and the availability of parallel tools are important investments for the future of parallel computing.

References

- [1] G. A. Geist, J. A. Kohl, P. M. Papadopoulos, *CUMULVS: Providing Fault-Tolerance, Visualization, and Steering of Parallel Applications*, Proceedings of Workshop on Environments and Tools for Parallel Scientific Computing, August 1996 (to appear in International Journal of Supercomputer Applications), <http://www.epm.ornl.gov/cs/cumulvs.html>.
- [2] 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>.
- [3] W. Gu, et al., *Falcon: On-line Monitoring and Steering of Large Scale Parallel Programs*, Georgia Institute of Technology Report GIT-CC-94-21, 1994, <http://www.cc.gatech.edu/tech.reports>.
- [4] B. Haimes, *pV3: A Distributed System for Large-Scale Unsteady CFD Visualization*, AIAA Paper 94-0321, Reno NV, Jan. 1994, <http://raphael.mit.edu/pv3/pv3.html>.
- [5] M. Heath and J. Etheridge, *Visualizing the Performance of Parallel Programs*, IEEE Software, 8(5), September 1991, pp. 29-40.
- [6] J. Hollingsworth, B. Miller, and J. Lumpp, *Techniques for Performance Measurement of Parallel Programs*. in *Parallel Computers: Theory and Practice*, ed. T. Casavant, P. Tvrdik, and F. Plasil, IEEE CS Press, 1996.
- [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] B. Miller, et al., *The Paradyn Parallel Performance Measurement Tool*, IEEE Computer, 28(11), pp. 37-46, November 1995, <http://www.cs.wisc.edu/paradyn>.
- [9] D. Reed, et al., *Scalable Performance Environments for Parallel Systems*, Proceedings of the Sixth Distributed Memory Computing Conference, IEEE CS Press, April 1991, <http://www-pablo.cs.uiuc.edu>.
- [10] C. Wu, H. Franke, and Y. Liu, *UTE: A Unified Trace Environment for IBM SP Systems*, <http://www.tc.cornell.edu/UserDoc/Software/Ptools/ute/>.
- [11] 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.

M98004836



Report Number (14) ORNL/CP--97244
CONF-970342--

Publ. Date (11) 199703

Sponsor Code (18) ^{DOE} /ER; NSF, XF

UC Category (19) UC-405; UC-000, DOE/ER

DOE