Visualization and Debugging in a Heterogeneous Environment

Adam Beguelin, Carnegie Mellon University and Pittsburgh Supercomputing Center
Jack Dongarra, University of Tennessee and Oak Ridge National Laboratory
Al Geist, Oak Ridge National Laboratory
Vaidy Sunderam, Emory University

A monitoring tool and a graphical interface working on top of the PVM software can help programmers make better use of heterogeneous networks of computers.

The emergence of a wide variety of commercially available parallel computers has created a software dilemma. Will it be possible to design general-purpose software that is both efficient and portable across these new parallel computers? Moreover, will it be possible to provide programming environments sophisticated enough for explicit parallel programming to exploit the performance of these new machines? For many computational problems, the design, implementation, and understanding of efficient parallel algorithms can be a formidable challenge. Additional issues of synchronization and multiple-task coordination make efficient parallel programs more difficult to write and understand than efficient sequential programs. Parallel programs are often less portable than serial codes because their structure may depend critically on the hardware's specific architectural features (such as how it handles data sharing, memory access, synchronization, and process creation).

The computing requirements of many current and future applications, ranging from scientific computational problems in the material and physical sciences to simulation, engineering design, and circuit analysis, are best served by concurrent processing. Multiprocessors can frequently address the computational requirements of these high-performance applications, but other aspects of concurrent computing are not adequately addressed when conventional parallel processors are used.

For instance, software aspects, including program development methods, scalable programs, profiling tools, and support systems, require significant development. While hardware and architectural advances in parallelism have been rapid, the software infrastructure has not kept pace, resulting in unsystematic and ad hoc approaches to the implementation of concurrent applications. In recent years, several research groups have focused on various aspects of this shortcoming, producing significant developments in programming paradigms, data partitioning, algorithms, languages, and scheduling.

Heterogeneous networks of computers ranging from workstations to supercomputers are becoming commonplace in high-performance computing. Until recently, each computing resource on the network remained a separate unit, but now hundreds of institutions worldwide are using the Parallel Virtual Machine[1] soft-

0018-9162/93/0600-0088$03.00 © 1993 IEEE

PVM: Heterogeneous distributed computing

PVM (Parallel Virtual Machine) is a software package being developed by Oak Ridge National Laboratory, the University of Tennessee, and Emory University. It enables a heterogeneous collection of Unix computers linked by a network to function as a single large parallel computer. Thus, large computational problems can be solved by the aggregate power and memory of many computers.

PVM supplies the functions to start tasks and lets the computers communicate and synchronize with each other. It survives the failure of one or more connected computers and supplies functions for users to make their applications fault tolerant. Users can write applications in Fortran or C and parallelize them by calling simple PVM message-passing routines such as pvm_send() and pvm_recv(). By sending and receiving messages, application subtasks can cooperate to solve a problem in parallel.

PVM lets subtasks exploit the type of computer best suited for finding their solution. Thus some subtasks may run on a vector supercomputer and others on a parallel computer or powerful workstation. PVM applications can be run transparently across a wide variety of architectures; PVM automatically handles all message conversion required if linked computers use different data representations. Participating computers can be distributed anywhere in the world and linked by a variety of networks.

The PVM source code and user's guide are available by electronic mail. The software is easy to install. The source has been tested on Sun, DEC, IBM, HP, Silicon Graphics Iris, Data General, and Next workstations, as well as parallel computers by Sequent, Alliant, Intel, Thinking Machines, BBN, Cray, Convex, IBM, and KSR. In addition, Cray Research, Convex, IBM, Silicon Graphics, and DEC supply and support PVM software optimized for their systems.

PVM is an enabling technology. Hundreds of sites around the world already use PVM as a cost-effective way to solve important scientific, industrial, and medical problems. PVM users include petroleum, aerospace, chemical, pharmaceutical, computer, medical, automotive, and environmental cleanup companies. Department of Energy and NASA laboratories use PVM for research, and numerous universities around the US use it for both research and teaching.

The software described in this article is freely distributed to researchers and educators, allowing them to harness their distributed computation power into comprehensive virtual machines. PVM and Hence are available by sending electronic mail to netlib@ornl.gov containing the line "send index from pvm3" or "send index from hence." Instructions on how to receive the various parts of the PVM and Hence systems will be sent by return mail.

Xab is also available from netlib. The index from pvm explains how to obtain this software. PVM problems or questions can be sent to pvm@msr.epm.ornl.gov for a quick and friendly reply.

ware package to develop truly heterogeneous programs utilizing multiple computer systems to solve applications (see sidebar). We designed PVM with heterogeneity and portability as primary goals. It lets machines with different architectures and floating-point representations work together on a single computational task.

In the development of heterogeneous concurrent applications for heterogeneous target environments, coarse-grained subtask partitioning and processor allocation are critical. Additionally, program module construction, specification of interdependencies and synchronization, and management of multiple objects for different architectures are tedious, error-prone activities. To address these issues and to provide at least partial solutions, we developed Xab and Hence, two packages that work on top of PVM to aid in the use, programming, and analysis of parallel computers.

Xab (X Window Analysis and Debugging) is a tool for runtime monitoring of PVM programs. Using Xab, programmers can easily instrument and monitor PVM programs by simply relinking to the Xab libraries. Xab is itself a PVM program, so it is very portable. However, making it peacefully coincide with the programs it monitors is problematic.

Hence (Heterogeneous Network Computing Environment) is an environment for the development of high-level programming techniques for the type of concurrent virtual machines provided by PVM. Its goal is to simplify the task (and thus reduce the chance of error) of programming a heterogeneous network of computers, while still providing the programmer with access to the high performance available from such configurations. There are several systems with goals similar to those of Hence. The Code system[2] and Paralex[3] both allow graph-based high-level specifications of parallel programs. Code includes tools that can map the specification into several different parallel languages or libraries such as Ada or C with shared-memory extensions. Paralex directly maps its specifications into C with calls to the Isis library.[4]

PVM

With PVM, users can exploit the aggregate power of distributed workstations and supercomputers to solve the computational Grand Challenges. Users view PVM as a loosely coupled distributed-memory computer programmed in C or Fortran with message-passing extensions. The hardware that constitutes a user's personal PVM may be any Unix-based network-accessible machine on which the user has a valid login.

We have tested the software with combinations of the following machines: Sun3, Sparcstation, MicroVAX, DECstation, IBM RS/6000, HP-9000, Silicon Graphics Iris, Next, Sequent Symmetry, Alliant FX, IBM 3090, Intel iPSC/860, Thinking Machines CM-2 and CM-5, KSR-1, Convex, Cray Y-MP, Fujitsu VP-2000, DEC Alpha, Intel Paragon, and Cray C90. In addition, users can port PVM to new architectures by simply modifying a generic "makefile" supplied with the source and recompiling.

Using PVM, users can configure their own parallel virtual computers, which can overlap with other users' virtual computers. Configuring a personal parallel virtual computer involves simply listing the names of the machines in a file that is read when PVM is started. Several different physical networks can coexist inside a virtual machine. For example, a local Ethernet, a Hippi (High-Performance Parallel Interface), and a

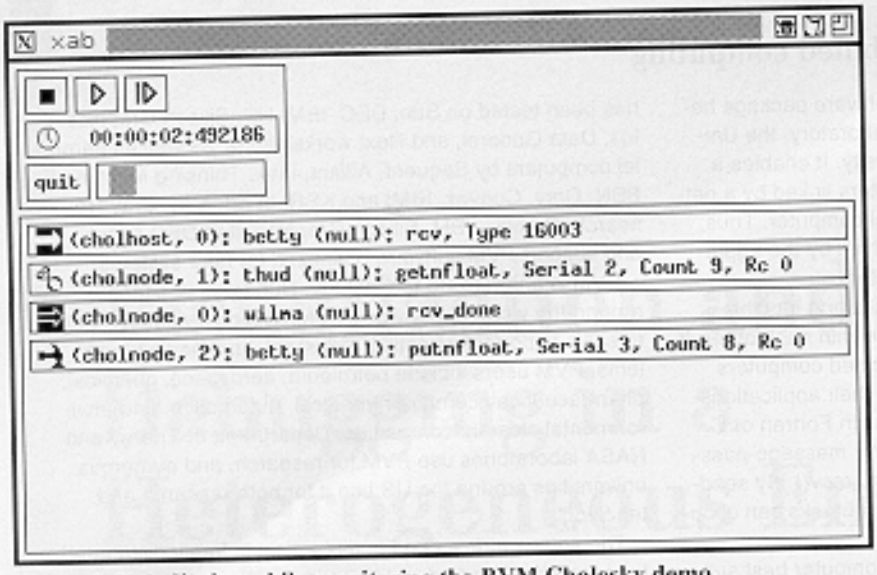

Figure 1. Xab display while monitoring the PVM Cholesky demo.

fiber-optic network can all be part of a user's virtual machine. Each user can have only one virtual machine active at a time; however, since PVM is multitasking, several applications can run simultaneously on a parallel virtual machine.

The PVM package is small (approximately 1 Mbyte) and easy to install. It needs to be installed only once on each machine to be accessible to all users. Moreover, installation does not require special privileges on any machines, so any user can do it.

Application programs that use PVM are composed of subtasks at a moderately coarse level of granularity. The subtasks can be generic serial codes or specific to a particular machine. In PVM, the user may access computational resources at three different levels:

- the *transparent* mode, in which subtasks are automatically located at the most appropriate sites,
- the *architecture-dependent* mode, in which the user can indicate specific architectures on which particular subtasks are to execute, and
- the *machine-specific* mode, in which the user can specify a particular machine.

Such flexibility lets different subtasks of a heterogeneous application exploit particular strengths of individual machines on the network.

The PVM programming interface requires that programmers explicitly type all message data. PVM performs machine-independent data conversions when required, thus letting machines with different integer and floating-point representations pass data. Applications access PVM resources via a library of standard interface routines. These routines allow the initiation and termination of processes across the network, as well as communication and synchronization among processes.

Application programs under PVM can possess arbitrary control and dependency structures. In other words, at any point in the execution of a concurrent application, the existing processes can have arbitrary relationships with each other and, further, any process can communicate or synchronize with any other.

While PVM is a very popular system for programming heterogeneous networks of computers, it is not the only system of this type. The p4 system[5] from Argonne National Laboratory, Express[6] from Parasoft, and Linda[7] from Scientific Computing Associates provide functionality similar to that of PVM.

Monitoring, debugging, and performance tuning

Tools should help programmers write and debug applications and tune their performance. With small-scale changes based on analysis of execution profiles, communication patterns, and load imbalances, programmers can improve concurrent application performance by an order of magnitude. Previous research in visualization focused on homogeneous parallel processing for both shared- and distributed-memory machines.[8] Our work focuses on visualization and debugging for networks of heterogeneous computers. Xab and Hence provide tools to help users with the complex task of understanding a program's behavior for both correctness and performance.

Xab. While PVM provides a solid programming base, it does not give users many options for analyzing or debugging PVM programs. To help in the development of PVM programs, Xab, a runtime monitoring tool, gives users direct feedback about the PVM functions their programs are performing. Xab has three parts: an Xab library to which the user links applications, a PVM process called *abmon* that quietly receives tracing messages from the library routines, and a display process called *xab* that is a graphical X Window display of the trace events.

Real-time monitoring is particularly apropos in a heterogeneous multiprogramming environment where differences in computation and communication speeds result from both heterogeneity and external CPU and network loads. Monitoring gives the user insight into program behavior in such an environment.

Xab monitors a PVM program by instrumenting calls to the PVM library. The instrumented calls generate events displayed during program execution.

A Fortran program normally accesses the PVM user routines via the libfpvm library that comes with PVM. Fortran programs use Xab by simply linking to libxpvm in place of libfpvm. With C, the procedure is slightly more complicated. The programmer must add the include file xab.h to source files that call PVM routines and then recompile the modified source files. This include file contains macros that replace the normal PVM routines with calls to the Xab library. Both Fortran and C programs must be linked with the Xab library, called *libab*.

Event messages. The Xab libraries call the normal PVM functions for the user, but they also send PVM messages to a special monitoring process called *abmon*.

Xab event messages generally contain an event type, a time stamp (in microseconds), and event-specific information. The event type indicates which PVM call is being invoked. In some cases, a PVM call may generate two events. For instance, the PVM barrier function generates an event before and after the barrier call. This lets the user see when barriers are initiated and