# Xab: A Tool for Monitoring PVM Programs

Adam Louis Beguelin
adamb@cs.cmu.edu
School of Computer Science
Pittsburgh Supercomputer Center
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213-3890
Phone: 412-268-5295
Fax: 412-681-5739

**Abstract**

*Xab (X-window Analysis and deBugging) is a tool for run time monitoring of PVM (Parallel Virtual Machine) programs. PVM supports the programming of a network of heterogeneous computers as a single parallel computer. Using Xab, PVM programs can easily be instrumented and monitored. Xab uses PVM to monitor PVM programs. This makes Xab very portable but it leads to interesting issues of how to make Xab peacefully coincide with the programs it monitors.*

*Xab consists of three main components, a user library, a monitoring program, and an X windows front end. The user library provides instrumented versions of the PVM calls. The monitoring program runs as a PVM process and gathers monitor events in the form of PVM messages. The Xab front end displays information graphically about PVM processes and messages.*

*This paper discusses the design, implementation, and use of the Xab tool. Related work is briefly presented and contrasted with the approach taken with Xab. How Xab works and how it is used are discussed in detail. Finally, the current status of Xab is presented along with future directions of where the research may go from here.*

## 1 Introduction

The PVM message passing system [2] for heterogeneous networks of computers has become quite popular. PVM is being used by scientists and educators in the US and abroad. Several of the US supercomputer centers provide PVM as a programming infrastructure for scientists who wish to spread their computation over machines available at the centers. Educators at several universities use PVM as a teaching tool in parallel programming courses. While PVM provides a solid programming base, it does not provide the user with many tools for analyzing or debugging PVM programs. Xab aids the user in the development of PVM programs. Xab is a run time monitoring tool for PVM programs. Xab gives the user direct feedback as to what PVM functions his or her program is performing. In its simplest form, this feedback is displayed in a window as shown in Figure 1.

The approach of real time monitoring is particularly apropos in a heterogeneous multiprogramming environment. Differences in computation and communication speeds here are due both to heterogeneity and external CPU and network loads. Monitoring can help give the user insight into how a program is behaving in such an environment.

Xab is a continuing research project. This paper discusses several related research projects, the current version of Xab, and the future development of the Xab tool.

## 2 Related Work

There are many other research projects that provide event display tools for parallel and distributed computing. SHMAP [4] displays shared memory access patterns for parallel Fortran programs. The HeNCE system [1] provides trace feedback specifically tailored to its programming paradigm. ParaGraph [6]

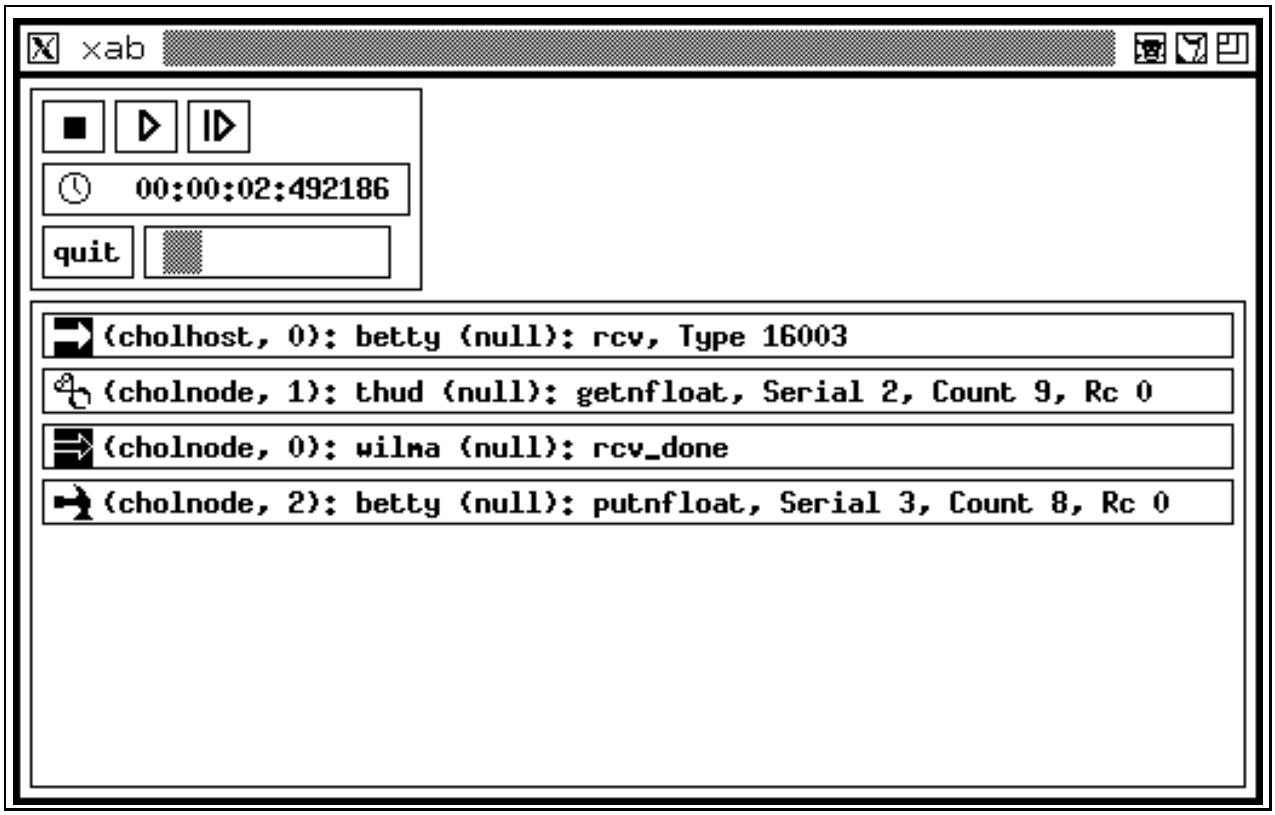


Figure 1: Monitoring the PVM Cholesky demo.

and BEE [3] are two projects that are similar to Xab.

ParaGraph is an X based tool for the display of events generated by parallel programs. ParaGraph provides a rich set of views for displaying events. PICL (portable instrumented communication library) [5] trace files are used as input to ParaGraph. These trace files are typically generated by a message passing parallel program written using PICL. However, the trace file may be generated by other tools. For instance, Xab provides a translator from Xab trace files to PICL format, allowing ParaGraph to be used with the output of Xab. Currently ParaGraph does not support the real time display of events. In [6], Heath and Etheridge discuss some of the issues involved in adding real time event display to ParaGraph.

BEE [3] also supports the display of events generated by parallel programs. Like Xab, BEE can display events as the program is executing. BEE events can be user defined and are generated by user instrumentation of a program. BEE monitoring is dynamic and multiple event interpreters are supported. BEE also supports the monitoring of heterogeneous programs and heterogeneous collections of machines. Unlike Xab, BEE does not currently support the automatic instrumentation of programs. However, the facilities provided by BEE are more general than those of Xab. Xab is specifically focused on monitoring PVM programs.

## 3 How Xab Works

Xab monitors a PVM program by instrumenting calls to the PVM library. The instrumented calls generate events that can be displayed during program execution. The PVM calls are instrumented by replacing them with calls to the equivalent Xab library routines. This replacement procedure is performed differently in Fortran and C.

### 3.1 Instrumentation

A Fortran program normally accesses the PVM user routines via the f2c library that comes with PVM.

With Xab, the f2x library replaces the f2c library. This allows Fortran programs to use Xab simply by linking to libf2x in place of libf2c. 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 these routines. 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.

### 3.2 Event messages

The Xab routines perform the normal PVM functions for the user but they also send PVM messages to a special monitoring process, called `abmon`. Covertly sending PVM messages is tricky business. Care must be taken not to disrupt any PVM messages the user's program may be manipulating. The current version of PVM (2.4.1 at the time of this writing) supports a single active message buffer. In other words, a PVM message is built by constructing a buffer and then sending it off to one or more locations. A typical sequence of commands is:

```
initsend();
putnint(&i, 1);
putnfloat(&f, 1);
snd("compnode", 1, 63);
snd("comppart", 4, 63);
```

Here the message buffer is initialized via `initsend()`. An integer and a float are placed into the buffer. Finally this buffer is sent to two different PVM processes. (Processes in PVM are addressed by a name and an integer process identifier.) When the program is instrumented, each of these lines of code generates a message to the Xab monitoring process. The Xab message is itself a PVM message. Precautions must be taken so the Xab message does not destroy the user's message. If, in the example above, the Xab instrumented `putnfloat()` function was to simply call `initsend()` to build a message and send it to the monitor, it would destroy the user's partially built message buffer. Therefore Xab messages must be hidden. To hide its messages, Xab takes the liberty of altering internal PVM data structures in a very limited way. Before Xab sends one of its event messages, it stores PVM's internal pointers to the user's message and reinitializes these pointers so the Xab message can be built. After the Xab message is sent, the original internal PVM pointers are restored, thus reinstating the user's message buffer. Since Xab only sends covert messages and never receives them, PVM's internal data structures for received messages do not need to be altered.

At first it may seem strange to use PVM to monitor PVM programs. There are several reasons why this makes sense. PVM is a very portable package and by using PVM as the infrastructure for Xab, Xab can work wherever PVM works. Using another medium such as sockets and TCP/IP would involve reinventing some of PVM's functionality. The least attractive part of using PVM for Xab is the altering of data structures internal to PVM. Future versions of PVM will provide multiple message buffers. Once this facility is available, the Xab implementation should no longer need to intrusively alter internal PVM data structures. Another alternative is to send messages back to the monitor only when doing so is guaranteed not to destroy a user's message buffer. In PVM there is only one distinct time when Xab could initialize the user's message buffer without damaging any user data. At `initsend()` time the user is destroying the current message buffer so Xab could do so as well. To further complicate matters, it is possible that a user program builds a buffer before enrolling in the virtual machine. If this is the case, then Xab cannot send messages back to the monitor process until the user program finally calls `initsend()`, after enrolling in the virtual machine. Another consequence of only sending events at `initsend()` time is that any events after the last `initsend()` in a program would never make it back to the monitor process. Finally, periodically sending back messages to the monitor is problematic. Section 3.4 discusses these tradeoffs in more detail.

The content of the Xab event messages generally include an event type, a time stamp, 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 allows the user to see when barriers are initiated as well as completed. The time stamp in the event message is the time of day on the machine where the PVM call is being executed. It is possible that the clocks on various machines involved in a computation will not be synchronized. Xab does not rely on the clocks being synchronized; events are simply displayed as they arrive. However, future versions of Xab may make use of the time stamps. For instance, it may be informative to know how long processes wait at a particular barrier in a program. Xab could use the time stamps from barrier events to dis-

play this information. The event specific information in an Xab message varies for different PVM routines. For the event generated at the start of a barrier, it is the name of the barrier and the number of processes that must reach the barrier before continuing. Other event messages contain similar event specific information.

Besides the event messages, Xab also inserts one additional piece of information into user messages. Each message sent by the user program is given a serial number. The message serial numbers are not unique until combined with the process identifier. For instance, the first message sent from the PVM process `<comp,0>` will be serial number 0, as will be the first message sent from processes `<comp,4>`. This serial number is prepended to the user's message buffer at `initsend()` time and stripped from the buffer at receive time. The addition of this single integer to user messages facilitates analysis of messages.

### 3.3 The monitoring processes

The `abmon` process receives event messages from the instrumented PVM calls and formats them into human readable form. The `abmon` program must be running before the user's program starts since it needs to receive event messages from the instrumented calls. The formatted event messages can either be written to a file or sent to the Xab display program. Just as an astronomer on Earth observes events that have traveled various distances, the `abmon` process observes events relative to its position in the virtual machine. When `abmon` formats events it also adds its own perspective within the virtual machine by placing local time stamps into the event record. The additional time stamps may be used to discern `abmon`'s perspective by indicating how long it takes events to propagate from a user process to the monitor process.

The display process will take events as formatted by `abmon` and display them in a window as shown in Figure 1. Xab supports two modes of event playback, continuous play or single step. When the play button is pressed the events will be displayed in real time. The slider controls the speed of the playback in continuous playback mode. Playback may be stopped at any time by pressing the stop button. The single step button will show only the next event in the monitor's event queue.

The following command line executes Xab, displaying the events in real time and saving them in a file for later review:

```
% abmon | tee evfile | xab
```

The `abmon` program reads event messages and writes them to standard out. The unix command `tee` copies the events to the file `evfile` and also passes them to `xab` via the pipe. The `xab` program actually opens a window and displays the events.

### 3.4 Timeliness versus message traffic

The method described previously for sending Xab monitor messages is utilized for every user call to the PVM library. This approach generates what may be considered an inordinately large number of messages. There is a tradeoff between the number of messages and the timeliness of the event display. If events are buffered and sent to the monitor after every $n$ events then the event display becomes more asynchronous as $n$ grows. In fact, when $n$ reaches the number of events in the program, the real time monitor becomes a postmortem processor. As the display lags behind the program state, some problems in program behavior cannot be detected. The example in Section 3.5 illustrates this point. Another factor one must consider is the amount of memory required to store events before sending them to the monitor. With Xab, events are immediately dispatched to the monitor. As a result, Xab adds little, in terms of memory requirements, to the PVM processes being monitored.

### 3.5 An Example

One of the demos that comes with PVM is a distributed Cholesky decomposition program. The window in Figure 1 is the Xab display in progress for this program. The host process, `(cholhost, 0)` is blocked on a receive. Process `(cholnode, 0)` has just received a message. The node process `(cholnode, 1)` is extracting data from a message buffer while `(cholnode, 2)` is placing eight floats into a message buffer.

One of the advantages of the real time display of Xab is its ability to show events immediately as they happen. For instance, Figure 2 shows the same Cholesky demo, but this time an error has been deliberately introduced. The host is waiting on a message of type 16040 while all the cholnode processes are sending messages of type 16004, thus the program has blocked indefinately. This example illustrates the advantage of real time monitoring. Postmortem monitoring would not work in this situation since the program would not complete and therefore, would never flush the events for display.

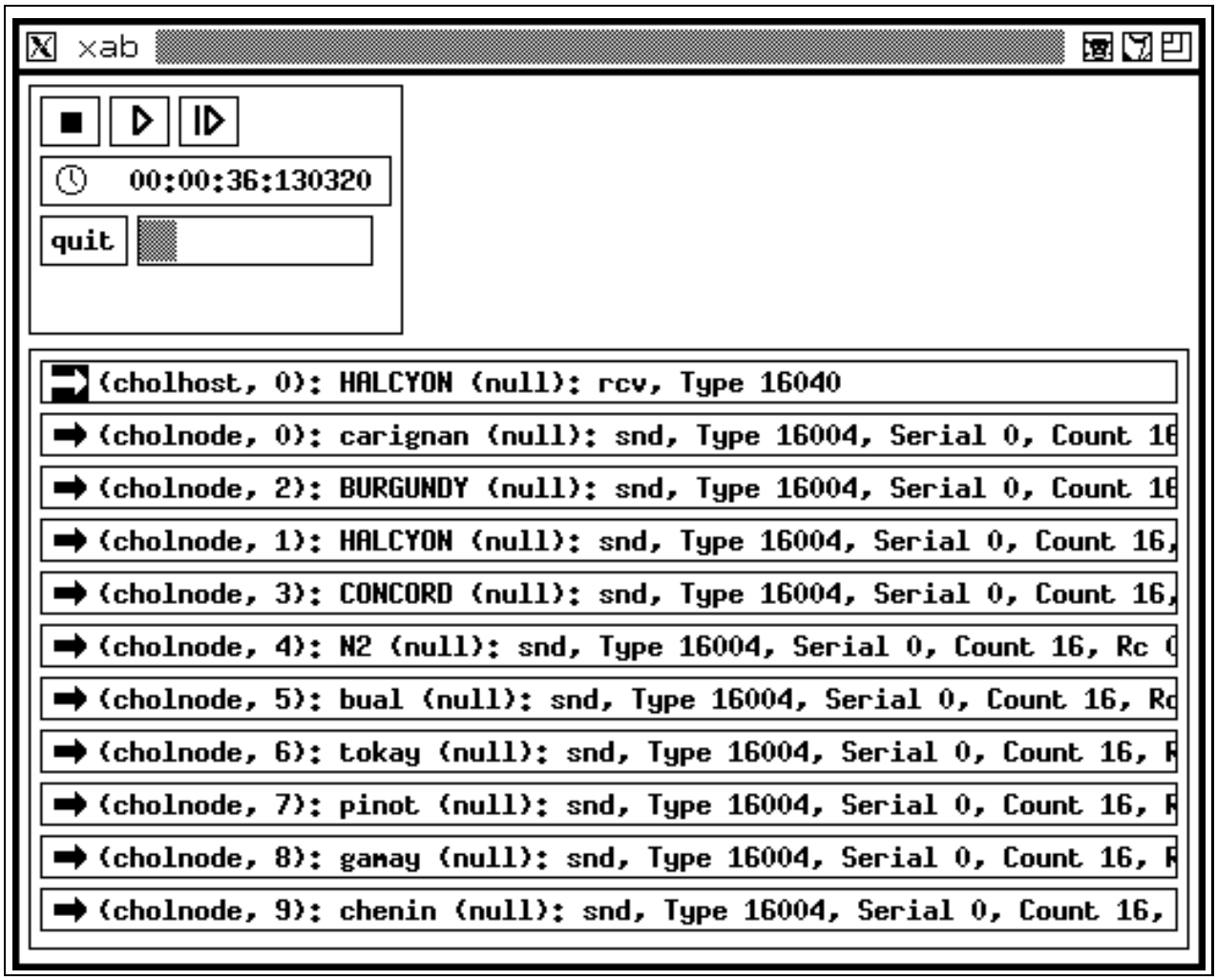


Figure 2: Indefinately blocked Cholesky demo.

## 4 Status and Future work

Xab is still in its early stages of development, and there are many issues that can be explored. Currently, the Xab display is very restricted. In the future Xab should support more interesting displays, ala ParaGraph. More analysis could also be done on the events that Xab collects. For instance, it is important that PVM programs extract data from a message buffer in the same order in which it it is inserted. Xab could alert the programmer when data insertion and extraction operations do not match. The bandwidth realized by a program could be calculated and displayed. If a standard CPU load metric could be devised, it could be piggy backed on the Xab events and machine load could be integrated into the Xab display. Since PVM works with multiprogrammed systems, program behavior may change drastically from run to run. Xab could help analyze multiple program executions by collecting events over several runs and supporting aggregate views of these events. For instance, run times for particular processes could be displayed per architecture. This kind of view would be useful when tuning an application for a heterogeneous environment. Scalability also becomes an issue as the number of processes and processors grow. Scalability needs to be addressed in terms of visually displaying large amounts of information and in terms of collecting this information. Parallel gather algorithms could possibly be used for collecting events. Finally, the actual amount of overhead involved in Xab monitoring needs to be carefully assessed.

### 4.1 Availability

Currently the Xab software has entered the testing phase. There are approximately 15 sites in the US and Europe exercising the code. By the time this paper appears, the code should be in relatively stable condition. The Xab system can be obtained by requesting it from the author.

## 5 Acknowledgements

Al Geist and Jack Dongarra provided insight early in the development that helped set the direction of the Xab research. Discussions with Bob Manchek and Roldan Pozo on tediously subtle issues of message passing systems proved insightful. Peter Stephan's proof reading improved the grammatical and semantic content of this article. Finally, I would like to thank the beta testers for their enthusiasm and patience.

## References

[1] A. Beguelin, J. J. Dongarra, G. A. Geist, R. Manchek, and V. S. Sunderam. Graphical development tools for network-based concurrent supercomputing. In *Proceedings of Supercomputing 91*, pages 435–444, Albuquerque, 1991.

[2] A. Beguelin, J. J. Dongarra, G. A. Geist, R. Manchek, and V. S. Sunderam. A users' guide to PVM parallel virtual machine. Technical Report ORNL/TM-11826, Oak Ridge National Laboratory, July 1991.

[3] Bernd Bruegge. A portable platform for distributed event environments. *ACM SIGPLAN Notices*, 26(12):184–193, December 1991. Proceedings of the ACM/ONR Workshop on Parallel and Distributed Debugging.

[4] J. Dongarra, O. Brewer, J. Kohl, and S. Fineberg. A tool to aid in the design, implementation and understanding of matrix algorithms for parallel processors. *Journal of Parallel and Distributed Computing*, 9(6):185–202, June 1990.

[5] G. A. Geist, M. T. Heath, B. W. Peyton, and P. H. Worley. A machine-independent communication library. In J. Gustafson, editor, *The Proceedings of the Fourth Conference on Hypercubes, Concurren t Computers, and Applications*, pages 565–568, P.O. Box 428, Los Altos, CA, 1990. Golden Gate Enterprises.

[6] M. Heath and J. Etheridge. Visualizing the performance of parallel programs. *IEEE Software*, 8(5):29–39, September 1991.