ORNL/TM-12981

Computer Science and Mathematics Division

Mathematical Sciences Section

XPVM 1.0 USER'S GUIDE

James Arthur Kohl, kohl@msr.epm.ornl.gov
G. A. Geist, geist@msr.epm.ornl.gov

Oak Ridge National Laboratory
P.O. Box 2008, Bldg 6012, MS 6367
Oak Ridge, TN 37831-6367

Date Published: November 1996

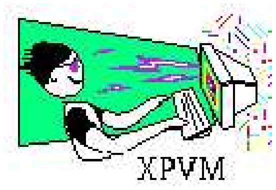
Research supported by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy.

Prepared by the
Oak Ridge National Laboratory
Oak Ridge, Tennessee 37831
managed by
Martin Marietta Energy Systems, Inc.
for the
U.S. DEPARTMENT OF ENERGY
under Contract No. DE-AC05-84OR21400

Contents

1 Introduction 1
2 Getting Started 3
  2.1 Obtaining XPVM / PVM 3
  2.2 Installing XPVM 3
    2.2.1 Installing the XPVM Source Distribution 3
  2.3 Using XPVM 6
    2.3.1 XPVM Command Line Options 7
    2.3.2 XPVM On-Line Help Information 8
3 Console Commands 11
  3.1 File Commands 11
    3.1.1 Quitting XPVM 11
    3.1.2 Halting PVM & XPVM 11
  3.2 Adding / Deleting Hosts 12
  3.3 Manipulating Tasks 13
    3.3.1 Task Spawning 14
    3.3.2 On-The-Fly Task Control 17
    3.3.3 Task Input & Output 19
    3.3.4 Non-XPVM Spawning 19
    3.3.5 Killing Tasks 21
    3.3.6 Signaling Tasks 22
    3.3.7 System Tasks 23
  3.4 XPVM Options 23
    3.4.1 User Action Modes 24
    3.4.2 Task Sorting 25
  3.5 Resetting PVM & XPVM 25
4 XPVM Monitor Functions 26
  4.1 Trace Files & Trace Control in XPVM 26
  4.2 Network View 30
  4.3 Space-Time View 31
    4.3.1 Space-Time Queries 32
    4.3.2 Space-Time Zooming 33
    4.3.3 Space-Time Task Height 33
  4.4 Utilization View 34
    4.4.1 Utilization Zooming 35
  4.5 Message Queue View 35
5 XPVM Debugging Features 37
  5.1 Call Trace View 37

5.2 Event History View 38
5.3 Task Output View 39
6 Future Work 40
7 Correspondence 41
8 References 41



XPVM 1.0 USER'S GUIDE

James Arthur Kohl, kohl@msr.epm.ornl.gov
G. A. Geist, geist@msr.epm.ornl.gov

Abstract

XPVM is a graphical console and monitor for PVM. XPVM provides a graphical interface to the PVM console commands, along with animated views to monitor the execution of PVM programs. These views provide information about the interactions among tasks in a parallel PVM program to assist in debugging and performance tuning. This report serves as a User's Guide for XPVM 1.0, the first released version of the interface, including all patch levels 1.*.*.

1. Introduction

Often when developing a parallel program it is difficult to determine the program's behavior due to the many concurrently executing threads of control. However, it is useful for the programmer to evaluate whether the program is running as expected or performing its work efficiently. One useful approach to this problem is the use of *program visualization* [3] to visually depict the program's execution using computer graphics.

XPVM is a graphical console and monitor for PVM [6]. Using the X Window System, it provides a visual interface to the functions of the PVM console, a real time performance monitor, a debugger, and a replay analysis tool.

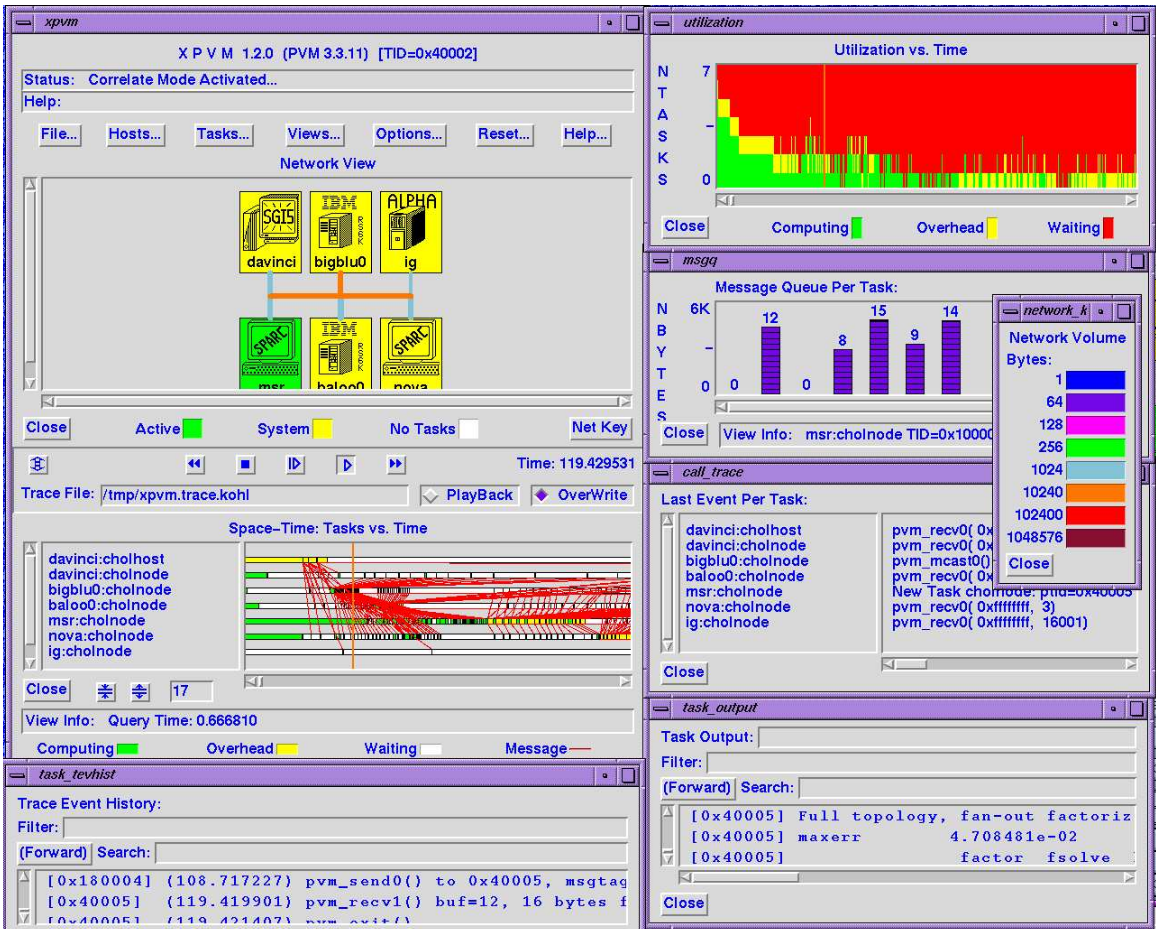
XPVM provides "point-and-click" access to the PVM console commands. A pull-down menu allows users to add or delete hosts to configure the virtual machine. Tasks can be spawned using a dialog box that prompts for spawn options, including which PVM routines to trace for XPVM. There are also buttons and menu choices for resetting, quitting and halting PVM and XPVM.

XPVM serves as a real time performance monitor for the virtual machine and PVM tasks. The monitor consists of views such as "Utilization" and "Space-Time" that scroll and zoom in unison and are time correlated, allowing the user to more easily compare various information about a particular occurrence in the program execution. There is also a "Message Queue" view that assists with analyzing message-passing efficiency, as well as a "Network" view to observe the message bandwidth and volume among hosts on the network.

XPVM provides some simple debugging features that allow access to textual details from remote tasks' executions. The text includes details of each invocation of a PVM library routine, including the calling parameters and return codes or results. Task output is also automatically collected and displayed by XPVM. (See Figure 1 for a snapshot of the XPVM interface.)

All of the above views and functionality in XPVM are driven by information captured via the PVM tracing facility. No annotation of the user program is necessary to use XPVM, as the PVM system libraries generate tracing information on request. All PVM distributions version 3.3.0 or later include "built-in" instrumentation for tracing user applications. No special patches or modifications to PVM are necessary.

The tracing information from PVM is delivered to XPVM using standard PVM message operations. Any tasks spawned from XPVM automatically send back trace events that describe their PVM activity. XPVM decodes these trace messages and saves the data into a trace file which can be read in to drive the views. XPVM can be used either "real time," to display the current state of a program as it changes, or



X P V M 1.2.0 (PVM 3.3.11) [TID=0x40002]

Status: Correlate Mode Activated...

Help:

Time: 119.429531

Trace File: /tmp/xpvm.trace.kohl

davinci:cholhost
davinci:cholnode
bigblu0:cholnode
baloo0:cholnode
msr:cholnode
nova:cholnode
ig:cholnode

View Info: Query Time: 0.666810

View Info: msr:cholnode TID=0x10000

Last Event Per Task:

davinci:cholhost	pvm_recv0(0x
davinci:cholnode	pvm_recv0(0x
bigblu0:cholnode	pvm_mcast0()
baloo0:cholnode	pvm_recv0(0x
msr:cholnode	New Task cholnode: ptid=0x40005
nova:cholnode	pvm_recv0(0xffffffff, 3)
ig:cholnode	pvm_recv0(0xffffffff, 16001)

Task Output:

Filter:

Search:

```
[0x40005] Full topology, fan-out factoriz
[0x40005] maxerr         4.708481e-02
[0x40005]                factor   fsolve
```

Trace Event History:

Filter:

Search:

```
[0x180004] (108.717227) pvm_send0() to 0x40005, msgtag
[0x40005]  (119.419901) pvm_recv1() buf=12, 16 bytes f
[0x40005]  (119.421407) pvm_exit()
```

Figure 1: XPVM Interface

"post-mortem" by replaying the saved trace files after the program has completed.

The remainder of this report describes the use and implementation of XPVM 1.0 in more detail. Section 2 provides a starting point to describe the general usage of XPVM, including how to obtain and install the software. Section 3 describes the main control menus and access to the PVM console commands via XPVM. Section 4 describes the monitor functions that XPVM provides in terms of the views provided to monitor the state of PVM programs. Section 5 describes the debugging features supported by XPVM. Section 6 proposes areas for future work.

2. Getting Started

2.1. Obtaining XPVM / PVM

XPVM is written in C using the TCL [4] and TK [5] systems, allowing extensibility to include a variety of views. XPVM is available via Netlib in the "pvm3/xpvm" subdirectory via "http://www.netlib.org/pvm3/xpvm/index.html," or by anonymous ftp to "netlib2.cs.utk.edu." The latest version of XPVM and other information may be obtained from the web at "http://www.netlib.org/utk/icl/xpvm/xpvm.html."

The latest version of PVM can be obtained via Netlib in the "pvm3" directory via "http://www.netlib.org/pvm3/index.html," or by anonymous ftp to "netlib2.cs.utk.edu." There is also a web page for PVM at "http://www.epm.ornl.gov/pvm."

The latest version of TCL / TK can be obtained via anonymous ftp to "ftp.smli.com" in the "/pub/tcl" subdirectory. Be sure to run the "make test" option after installing both TCL and TK on your system.

2.2. Installing XPVM

In general, the source distribution of XPVM can be compiled on any Unix platform which supports X Windows and the TCL and TK systems. Each distribution of XPVM includes a "README" file that describes exactly how to install that version of the XPVM software. You should always read the "README" file before proceeding to install XPVM. The following subsection overviews the general installation process for source distributions of XPVM.

2.2.1. Installing the XPVM Source Distribution

In order to build XPVM from the source distribution you will need:

1. PVM 3.3.0 or later.

2. TCL 7.3 or later.
3. TK 3.6 or later.

PVM 3.3.0 and later releases of PVM are instrumented to support tracing. This alleviates the need to annotate user applications to obtain tracing information, and allows flexible control over tracing functions at run-time.

XPVM is programmed using the TCL and TK systems to expedite the development of window environment operations. As a result, XPVM requires the TCL and TK libraries for compilation. TCL and TK can be obtained via anonymous ftp to "ftp.smli.com" in the "/pub/tcl" directory. These packages are well documented and contain their own installation instructions. Additional information regarding TCL and TK can be obtained from the web page at:

http://www.sunlabs.com:80/research/tcl/

Be sure to test your TCL and TK installations by running "make test" for **each** before proceeding with XPVM's installation.

To install XPVM from a source distribution, execute the following steps:

1. First unpack the distribution file using "uudecode" to produce a compressed tar file. Obtain the file "xpvm.src.tar.z.uu" from Netlib and execute the following command:

   ```
   % uudecode xpvm.src.tar.z.uu
   ```

   This should produce an "xpvm.src.tar.Z" file. Next extract the actual distribution contents using the command:

   ```
   % zcat xpvm.src.tar.Z | tar xvf -
   ```

   This will create an "xpvm/" directory containing the necessary files for building and running XPVM, including the "xpvm/src" and "xpvm/tracer" subdirectories.

2. Next, customize the Makefile for generating the XPVM executable. XPVM uses the "aimk" tool included in the regular PVM distribution, so the main Makefile for XPVM is found in "xpvm/src/Makefile.aimk." The definitions in this file for "TCLLIBDIR," "TKLIBDIR," "TCLINCL," and "TKINCL" must be adjusted to point to the locations of the TCL and TK libraries and include files on your system.

   Typical settings for these variables are:

```
TCLLIBDIR = -L/usr/local/tcl
TKLIBDIR = -L/usr/local/tk
TCLINCL = -I/usr/local/tcl
TKINCL = -I/usr/local/tk
```

In general, TCLLIBDIR and TKLIBDIR should point to the directories containing the TCL and TK library files "libtcl*.a" and "libtk*.a," respectively. TCLINCL and TKINCL should point to the main source directories for TCL and TK that include the "tcl.h" and "tk.h" header files, respectively.

You will also need to verify the location of the X11 libraries and include files on your system, and appropriately set the XLIBDIR and XINCL variables.

In a typical system, with the X11 header files installed in "/usr/include" and the X11 libraries in "/usr/lib," you can just set:

```
XLIBDIR = -L/usr/lib
XINCL = -I/usr/include/X11
```

- or equivalently -

```
XLIBDIR =
XINCL =
```

You may also need to add linking of the dynamic loading library for your system, to support the new TCL / TK dynamic loading capability. This amounts to adding a "-ldl" or "-ldld" to the SYSLIBS define in Makefile.aimk. The specific library depends on the dynamic loader on your system. Try a "man ld" to determine the proper flag.

3. The next step is to set up the XPVM environment, so that XPVM can determine where it has been installed. The user must set certain shell environment variables. "XPVM_ROOT" must be set to point to the absolute path where you are installing XPVM. For example, if "joe_user" unpacked XPVM using the above commands in his home directory, "/home/joe_user," he should set "XPVM_ROOT" as follows:

   ```
   setenv XPVM_ROOT /home/joe_user/xpvm
   ```

   This should be done in the user's ".cshrc" file (or equivalent), and can be accomplished using the "xpvm/src/cshrc.stub" file in the distribution. This file also includes commands to add the XPVM executable to the user's path.

The user will also need to set the "TCL_LIBRARY" and "TK_LIBRARY" environment variables to point to the directories containing the "*.tcl" initialization script files for TCL & TK. A typical location for these directories is:

```
setenv TCL_LIBRARY /usr/local/tcl/library
setenv TK_LIBRARY /usr/local/tk/library
```

4. Once the above steps have been applied, go to the top-level "xpvm/" directory ($XPVM_ROOT), and type "make."

   This will create an executable "xpvm" file for the architecture you are currently on. The executable file will be placed in a subdirectory of "xpvm/src" with the same name as the PVM architecture identifier:

   ```
   $XPVM_ROOT/src/$PVM_ARCH/xpvm
   e.g. /home/joe_user/xpvm/src/SUN4/xpvm
   ```

   This file should already be in the user's execution path if the cshrc.stub file was used.

This should complete the installation of XPVM from the source distribution. If any problems are encountered please contact your system's administrator or send email to the author, at "kohl@msr.epm.ornl.gov."

2.3. Using XPVM

To analyze a program using XPVM, a user need only compile their program using the PVM library version 3.3 or later. Any task spawned from XPVM (see Section 3.3.1 on Page 14) will return trace event information. These trace events can be used for analysis in real time or for post-mortem playback from saved trace files.

When XPVM is invoked it will automatically start up PVM on your machine if it is not already running there. If PVM is already running, XPVM simply attaches to the existing virtual machine as a new task. Users can specify a default list of hosts to XPVM, exactly as can be done with the regular PVM console, using a hostfile. The default location for this hostfile is in the user's home directory in a file called ".xpvm_hosts" (see Section 3.2 on Page 12).

2.3.1. XPVM Command Line Options

There are several command line options supported for XPVM that allow customized selection of default host files, trace files, and the local machine name (see below). The command line syntax and options currently supported in XPVM are as follows:

usage: xpvm [hostfile] [-PO] [-T trace] [-N name] [-M nbytes] [-HSetv]

where:

hostfile = Specifies an alternate XPVM hostfile, other than "$HOME/.xpvm_hosts," for starting the virtual machine. Note that this hostfile, or the default hostfile, are only used to actually create the desired virtual machine if PVM is *not* already running. If a virtual machine is already present, any hostfile is ignored. XPVM will, however, keep track of the host options specified in the hostfile, for subsequent host adding (see Section 3.2 on Page 12).

-P = Start XPVM in "PlayBack" mode, where existing trace files are replayed for post-mortem analysis. In this mode, XPVM will not try to reset the default trace file (or the manually specified trace file, see "-T trace" below).

-O = Start XPVM in "OverWrite" mode, to monitor a new application in real time by creating a new trace file. In this mode, XPVM will try to reset (delete) the current default trace file (or the manually specified trace file, see "-T trace" below). If the file exists, XPVM will prompt the user to overwrite it.

-T trace = Use "trace" as the default trace file. If this option is omitted the default trace file will be "/tmp/xpvm.trace.*user*," where *user* is the user's login name. The trace file is used to store real time monitoring information, and saves this information for replay. Note that XPVM always prompts the user before overwriting any trace file. See Section 4.1 on Page 26 for more details.

-N name = Use "name" as local hostname. This option is typically used to select from among different network interfaces on a single machine. An alternate name, other than what is returned by the "hostname" command, can be specified.

-M nbytes = Set the default message size for the "Message Queue" view and "Network" view message volume rendering. In PVM 3.3, the size of a message in bytes is not passed in the trace events captured for the message send events. For this reason, the message size is not known until the message receive event is processed. This prohibits the display of queued-but-not-received messages for the "Message Queue" view, and prevents display of in-transit message volume in the "Network" view. By setting a default message size, these view features can simulate actual message queues and network volume by assuming a fixed size for each message.

-H = Print this help information (and quit XPVM).

-S = Dump the PVM 3.3. SDDF trace event descriptors (and quit XPVM). These textual descriptors define the contents of each event type in the trace file (see Section 4.1 on Page 26).

-e = Dump raw PVM event text during trace play. This option allows the user to observe the raw textual event information coming from a PVM application. This option is typically only useful in rare cases where the portrayal of events in the views is not sufficient to show intermediate or pending information, such as incomplete send-receive pairs. (The "Trace Event History" view now provides this information via the graphical interface.)

-t = Text Mode, no animation in views. This option allows more efficient capturing of trace files without the overhead of animating them at run-time. The trace events are still processed and stored in the trace file but none of the views are updated. (This is a more severe measure than the "Fast-Forward" trace play mode.)

-v = Verbose operation. This flag turns on the display of internal XPVM status and information.

To see the options supported by your current version of XPVM, execute XPVM with the "-HELP" ("-H") option. (Note that the "-help" ("-h") option is reserved in all TCL applications, and displays general help for the TCL portion of XPVM's functionality only.)

2.3.2. XPVM On-Line Help Information

XPVM provides some status information at the top of the main window, in the message line that shows "Welcome to XPVM" at startup. Often, when XPVM pauses to compute internal view data, a status message here will indicate what operation is being performed, and when it is finished. Each individual view may also contain its own informational messages (see Section 4 starting on Page 26).

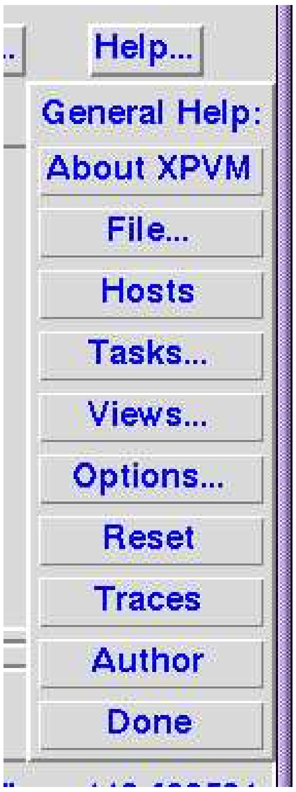


Figure 2: Help Menu

Below the status message line is an automatic help message display that show one-line information based on the location of the mouse pointer in the interface. For example, if the mouse pointer is moved over a particular button, label or view canvas, the help message will display a one-line description of what will happen if the given button or canvas are clicked on, or to what the corresponding label refers.

In addition to the status and automatic help information, there is also a full text on-line help system in XPVM. This system is menu-driven, and is brought up from the "Help..." button on the main panel at the top of the XPVM window, next to the other console command buttons (see Section 3 on Page 11). The "Help..." menu is shown in Figure 2.

There are two levels of menus for the on-line help information. Selecting an item from a help menu either brings up a scrollable text window containing a brief overview of the desired help information, or else switches to a sub-menu for more detailed help categories.

At the top help level, there are the following choices:

About XPVM This provides some general information about XPVM, including a quick reference to the remainder of the on-line help categories.

File... This provides information about the main XPVM controls, such as are typically found under the main "File" menu of a user interface. This category describes the functionality of the "Quit" command and the "Halt" command (see Section 3.1 on Page 11).

Hosts Hosts help describes how to add or delete hosts from the virtual machine using the console command interface. The use of hostfiles is also described. (See Section 3.2 on Page 12.)

Tasks... Selecting this help category switches to a more detailed sub-menu that provides information regarding different aspects of tasks in XPVM. Included here is information regarding spawning, killing and signaling tasks. The subcategories for Tasks help are "Spawn," "Kill," "Signal" and "Sys Tasks." These items are described in detail in Section 3.3 on Page 13.

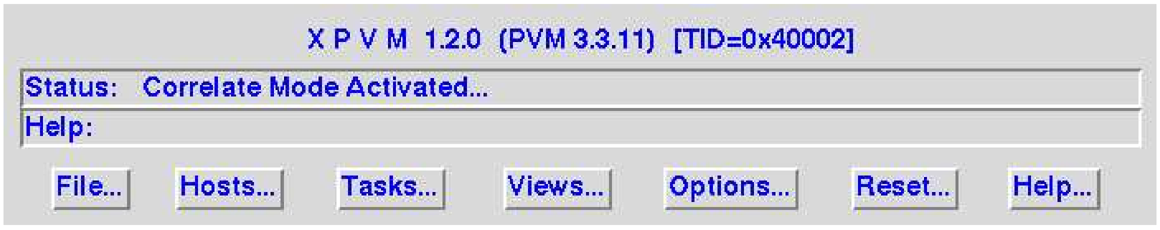
Views Selecting this help category switches to a more detailed sub-menu that provides information regarding different aspects of views in XPVM. The subcategories for Views are "Network," "Space-Time," "Utilization," "Message Queue," "Call Trace," "Task Output" and "Event History." These views are described in detail in Section 4 on Page 26.

Options... This help category describes some of the different XPVM run-time options which can be set. The sub-categories for Options include "Action Modes" which describe how to query views for details or correlate views to specific times. The "Task Sorting" sub-category describes different ways that tasks can be arranged in the views, including custom user placement. (See Section 3.4 on Page 23.)

Reset This category describes the different XPVM Reset functions provides, for resetting PVM, the views and the trace file. (See Section 3.5 on Page 25.)

Traces This help describes how the tracing features of XPVM work, and how the user can control the monitoring of PVM tasks and trace file playback. (See Section 4.1 on Page 26.)

Author This help category serves as a "catch-all," and provides information on how to reach the author of XPVM to answer questions that are beyond the scope of the provided help information. (See Section 7.)


X P V M 1.2.0 (PVM 3.3.11) [TID=0x40002]

Status: Correlate Mode Activated...

Help:

Figure 3: Console Interface

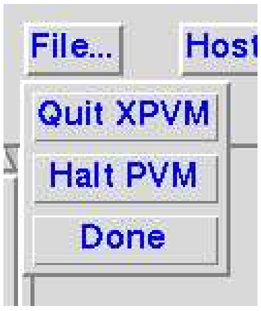


Figure 4: File Commands

3. Console Commands

XPVM provides "point-and-click" access to the PVM console commands. The interface for these console commands is shown in Figure 3. The following subsections describe these functions in more detail.

3.1. File Commands

At present, there are two supported main XPVM file commands, "Quit" and "Halt," as shown in Figure 4.

3.1.1. Quitting XPVM

This function is identical to the PVM console "quit" command. When the "Quit" button in XPVM is pressed, XPVM will exit without further prompting, and will leave PVM running. The current virtual machine will remain intact and the PVM daemons will still be running.

3.1.2. Halting PVM & XPVM

This function is identical to the PVM console "halt" command. When the "Halt" button in XPVM is pressed, PVM will be shut down without any further prompting.

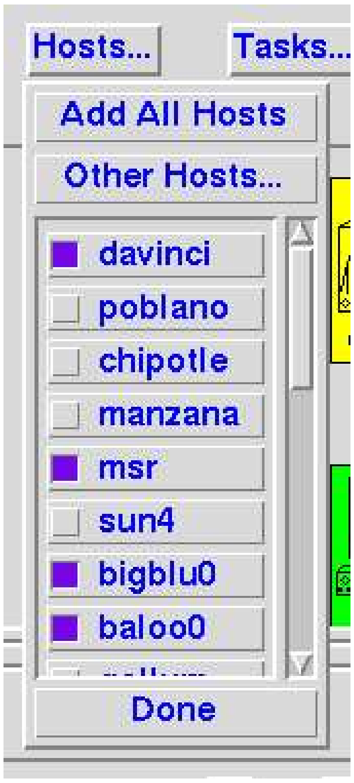


Figure 5: Hosts Menu

All tasks will be killed, the virtual machine will be disconnected and halted (not the actual machines, just the PVM daemons), and XPVM and all PVM consoles will exit.

3.2. Adding / Deleting Hosts

Hosts can be added to and deleted from the PVM virtual machine using a pull-down menu in XPVM, as shown in Figure 5. This menu is raised by clicking on the "Hosts..." button in the main XPVM window. The menu shows a list of hosts, with a colored status box to the left of each host name to indicate whether the host is currently in the virtual machine. Clicking on a particular host toggles its membership in the virtual machine, either adding or deleting that host from the current configuration depending on whether the host is already present or not. A purple status box means the host is in the virtual machine (or in the process of being added), and a clear status box means the host is not in the virtual machine.

The default list of hosts is user configurable by modifying the "$HOME/.xpvm_hosts" file (or by specifying a hostfile manually with the "-T trace" command line option, see Section 2.3.1 on

Page 7 for more details). If a virtual machine is not already running when XPVM is invoked, a user-customized virtual machine configuration will automatically be started by XPVM using the ".xpvm_hosts" hostfile in the user's top level directory. This hostfile is of the same format as regular PVM hostfiles, as documented in the *pvmd3* man page, or in Chapter 3 of [6]. In a hostfile, each line should contain the name of exactly one host machine to be added to PVM. If the first character of a line is a '#' then that line is ignored and treated as a comment. If the first character is a '&' then the host will not be added to the virtual machine on startup, but will still be included in the host menu list. Any options specified in the hostfile for specific hosts are maintained by XPVM, independent of whether each host is actually added on startup. Subsequent adds of a host to the virtual machine will use the desired options.

An example of a hostfile is shown below:

```
#
# XPVM Hosts
#
msr.epm.ornl.gov
&bigblu0
&davinci.epm.ornl.gov lo=curtis
```

This file causes machine "msr" to be added to the virtual machine on XPVM startup, and adds "bigblu0" and "davinci" to the host menu list, but does not add them to the virtual machine yet. The "lo=curtis" option is saved for any future XPVM host add of "davinci."

The XPVM Hosts menu also includes two fixed options, "Add All Hosts" and "Other Hosts." The "Add All Hosts" option will attempt to add all hosts currently listed on the Hosts menu to the virtual machine. Specifically, any hosts not already present will be added. The "Other Hosts" feature allows new hosts to be added to the virtual machine (and the Hosts menu) interactively by the user. Clicking on the "Other Hosts" button causes a host name prompt to appear right below the button. The new host is added by simply typing in its name and hitting "<Return>" on the keyboard, or pressing the "Accept" button. The "Other Host" prompt can be lowered without adding a host by pressing the "Cancel" button or re-pressing the "Other Hosts" menu button.

3.3. Manipulating Tasks

There are several ways of manipulating tasks using XPVM. Tasks can be spawned and traced, tasks' tracing options can be modified on-the-fly, and tasks can be

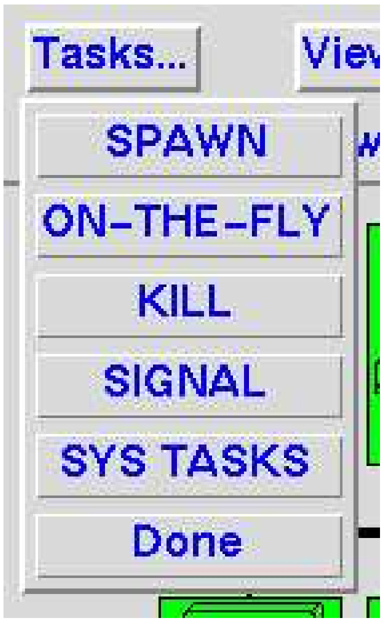


Figure 6: Tasks Menu

signaled and killed. Some system tasks can also be monitored. All of these features are available via the "Tasks..." menu, as shown in Figure 6.

The following subsections describe these features in more detail.

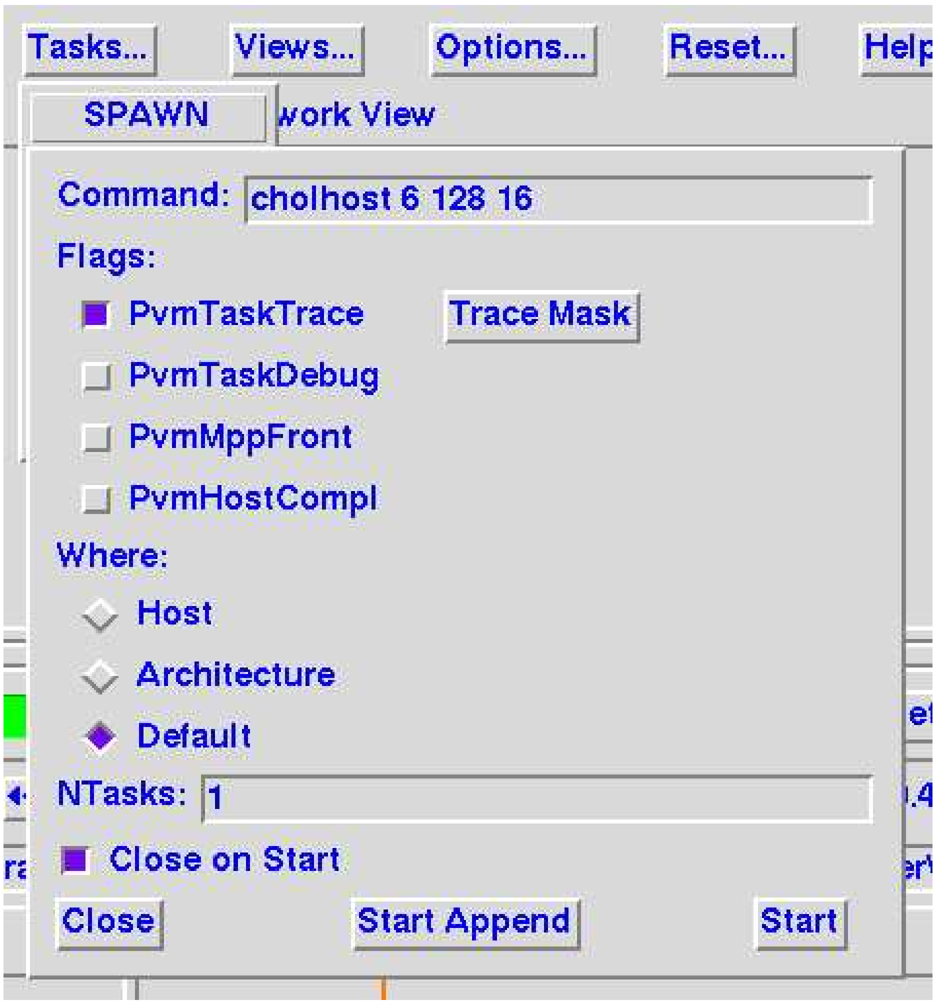
3.3.1. Task Spawning

The easiest way to monitor PVM tasks using XPVM is to spawn the desired tasks directly from XPVM, using the "SPAWN" dialog which is selected from the "Tasks..." menu (see Figure 7). This dialog provides similar functionality to the "spawn" command in the regular PVM console. (For information on tracing tasks spawned outside of XPVM, see Section 3.3.4.)

There are several fields to set in the spawn dialog box before a task, or set of tasks, is spawned:

Command: The "Command" field is used to enter the name of the PVM task to spawn, i.e. the name of the executable file, including any command line arguments for that program, e.g. "foo 1 2 3."

Flags: The four spawn "Flags" available for PVM task spawning are all available for spawning from XPVM. To set or unset a desired flag, simply click on it. The square status box to the left of each flag indicates whether the flag is set or not – filled in (purple) for set, or empty for not set. Note that because XPVM



Command: cholhost 6 128 16

Flags:

[✓ PvmTaskTrace]

PvmTaskDebug

PvmMppFront

PvmHostCompl

Where:

Host

Architecture

[✓ Default]

NTasks: 1

[✓ Close on Start]

Figure 7: SPAWN Dialog

is intended to monitor PVM programs, the "PvmTaskTrace" flag defaults to set. All other flags default to not set.

Trace Mask: Because it may not be desirable in all situations to trace *all* of the PVM system routines, the PVM library instrumentation uses a trace mask to selectively determine which PVM routines are to be traced. When the trace mask for a particular routine is set, trace events are generated on each invocation of that routine by the user application. One event is recorded at the entry to each traced PVM routine. This event contains the values of any calling parameters for the specific invocation of the routine. Another event records the return from each routine, including any returned values or status codes for that invocation.

The user can control precisely which PVM routines are traced for XPVM by using the "Trace Mask" menus on the SPAWN dialog. To set the trace mask press the "Trace Mask" button to the right of the "PvmTaskTrace" flag. This pops up a hierarchical set of menus for turning the tracing on or off for individual PVM calls or groups of calls. The trace events are logically organized into groups at the highest level trace mask menu, with a sub-menu for each trace group. Each trace mask sub-menu has a "trace group" indicator at the top of the sub-menu to show whether the entire group of trace events has been activated or not. Alternately, clicking on a group indicator will force all events in that group to be toggled on or off. At the highest level trace mask menu, an "All Events" indicator functions similarly, to toggle tracing for all events in all groups. XPVM can also modify a task's trace mask dynamically, on-the-fly, using the "On-The-Fly" dialog (see Section 3.3.2 on Page 17.)

Where: If a task is to be spawned on a particular host or type of architecture, the user can specify this by selecting the desired "Where" option. The choices for Where include "Host," "Architecture," or "Default." When the Host or Architecture choices are selected, the SPAWN dialog automatically reorganizes to add an appropriately labeled "Host" or "Arch" prompt for entering the desired spawn destination. Otherwise, if the Default location is selected, the task will be spawned to an arbitrary location at PVM's discretion.

NTasks: The "NTasks" prompt allows the user to specify the total number of tasks to be spawned by each invocation from the SPAWN dialog. This option defaults to a single task.

Close on Start: This flag has nothing to do with the actual spawning of tasks, but is an interface convenience. When set, this option causes the SPAWN dialog to be automatically lowered when the user invokes the actual spawn. This removes the visual obstruction caused by the dialog and reveals any obscured views in time to see the initial tracing information coming from the newly spawned tasks.

Close: This button simply closes the SPAWN dialog, without spawning any tasks or invoking any other actions. All options currently set in the SPAWN dialog will be saved for the next time it is raised.

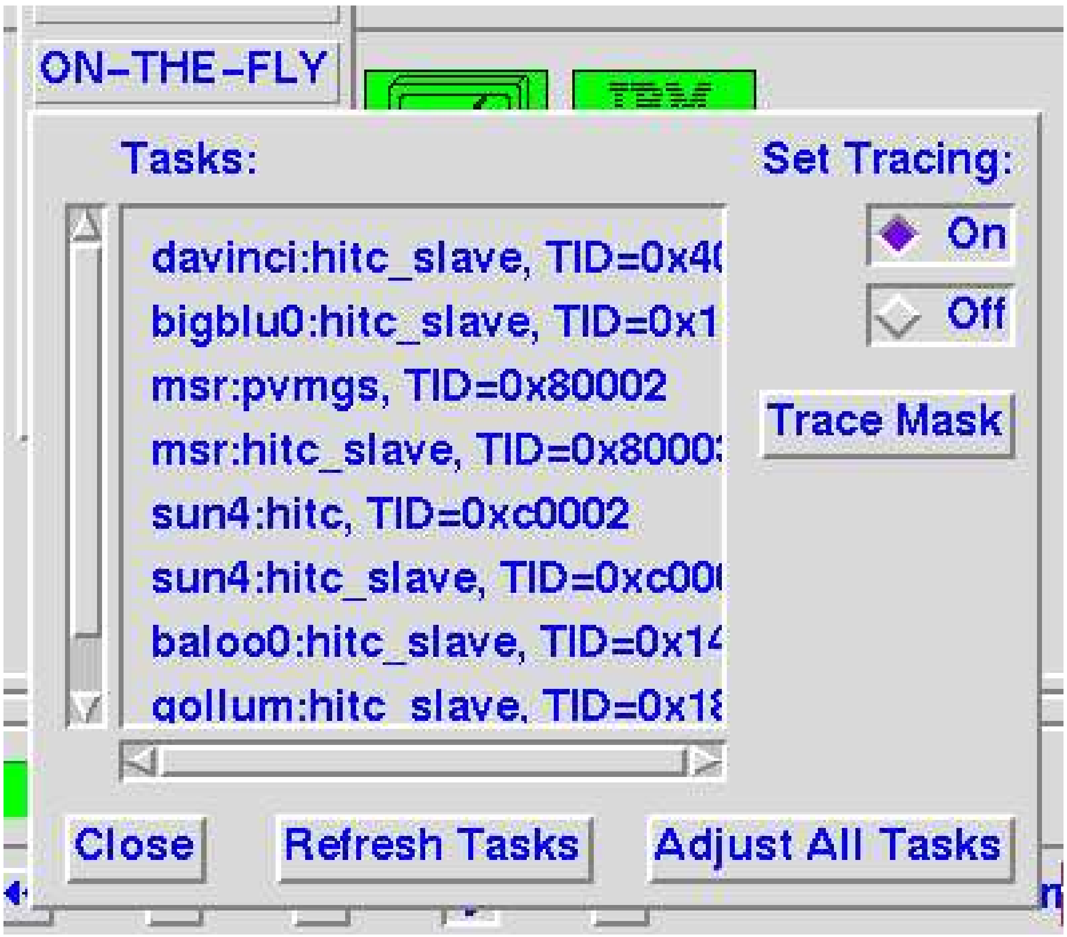
Start Append: The “Start Append” button is used to actually initiate the spawning of tasks in PVM. This button serves the same purpose as the “Start” button described below, except that the “Append” spawn command prevents any resetting of the views or trace files as a result of the spawn. The new tasks' trace events will be “appended” to any currently tracing application, and the trace information will be merged into any existing view displays.

Start: The “Start” button is used to actually initiate the spawning of tasks in PVM. Pressing this button will submit the specified spawn options to PVM and spawn the desired task(s). Upon submission of the spawn via the “Start” button, all views will be reset and the trace file will be cleared for overwriting (with the permission from a user prompt if the current trace file is not empty – before XPVM will destroy an existing trace file, it will always prompt the user and hold execution of the spawn command until the user confirms the overwrite). The same functionality of the “Start” button can be activated by simply hitting “<Return>” at the “Command:” prompt in the SPAWN dialog.

3.3.2. On-The-Fly Task Control

Once a task has been configured for tracing to XPVM (by spawning the task from XPVM, or by manually connecting as described in Section 3.3.4), the user can modify its tracing options on-the-fly using the “On-The-Fly” dialog under the “Tasks...” menu (see Figure 8). A task's tracing can be turned completely on or off by selecting the “On” or “Off” item under the “Set Tracing” label. When the tracing is set to “On,” the trace mask can also be adjusted to control which PVM routines send back trace events to XPVM (see Section 3.3.1 on Page 14).

Once the desired trace settings have been selected on the “On-The-Fly” dialog, they can be applied to any single task by clicking with the left mouse button on that task's label in the “Tasks:” list on the dialog. Similarly, *all* tasks can be updated to the new trace settings by pressing the “Adjust All Tasks” button.



davinci:hitc_slave, TID=0x40
bigblu0:hitc_slave, TID=0x1
msr:pvmgs, TID=0x80002
msr:hitc_slave, TID=0x8000
sun4:hitc, TID=0xc0002
sun4:hitc_slave, TID=0xc00
baloo0:hitc_slave, TID=0x14
gollum:hitc_slave, TID=0x18

Figure 8: On-The-Fly Dialog

XPVM avoids the overhead of continuously polling for PVM tasks while the "On-The-Fly" dialog is up, so the tasks listed may not accurately reflect the existence of all currently executing tasks. To update the task list, press the "Refresh Tasks" button on the "On-The-Fly" dialog.

Pressing the "Close" button on the "On-The-Fly" dialog will simply close the dialog, without modifying any tasks' trace settings.

3.3.3. Task Input & Output

Tasks spawned by XPVM automatically re-direct their "stdout" output to XPVM for display. The XPVM "Task Output" view collects the output and shows it to the user in a scrolling window. This view also allows the saving of task output in a file. (See Section 5.3 on Page 39).

Unfortunately, it is not yet possible to pass interactive user input to tasks spawned by XPVM, or those spawned by the regular PVM console. Any programs that require interactive input from the user at run-time must be spawned outside of XPVM from a Unix shell or other means. To capture trace events from tasks spawned in this way, you must follow the directions below in "Non-XPVM Spawning."

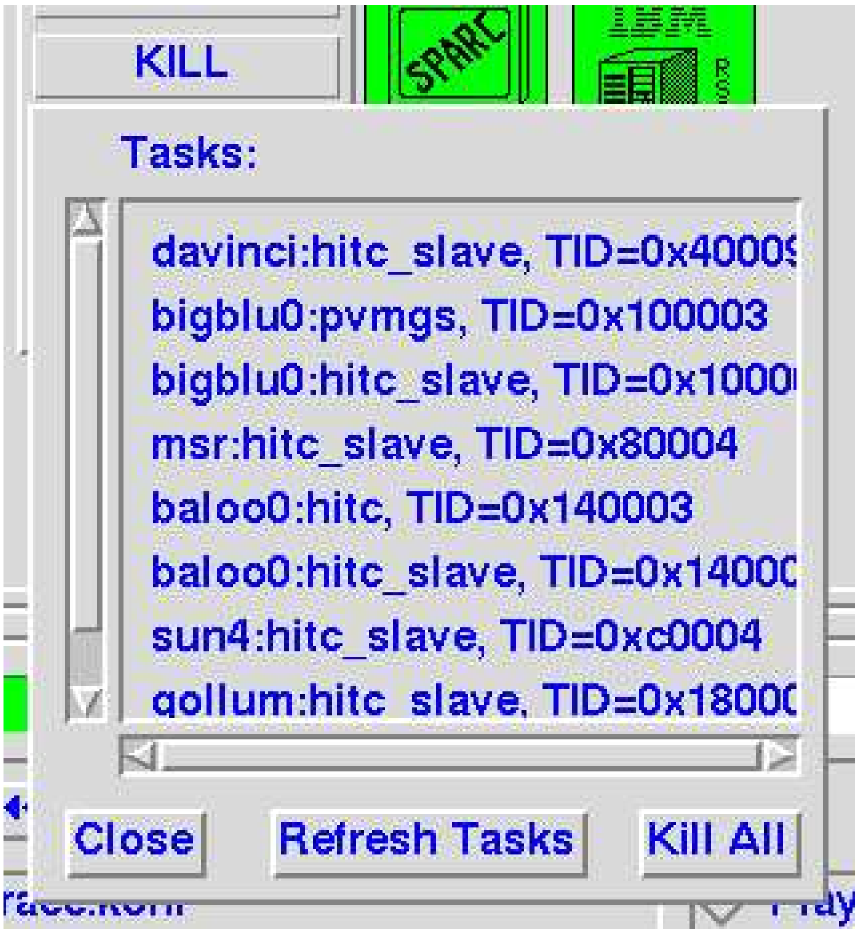
3.3.4. Non-XPVM Spawning

In the case where a PVM task must be started independently from a Unix shell, there is a way to allow trace events to be manually re-directed to XPVM for viewing. XPVM always joins a special PVM group, "xpvm," on startup, as group instance number "0." Therefore, to capture trace events from an independently spawned task, the user application can call pvm_gettid() to obtain the TID for XPVM, and then can set the task's tracing options appropriately. This is shown in the following code segment.

```
#include <pvm3.h>
#include <pvmtev.h>

Pvmtmask trace_mask;

int xpvm_tid;

/* Get XPVM Task ID */

if ( (xpvm_tid = pvm_gettid( "xpvm", 0 )) > 0 )
```

```
{
    /* Set Self Trace & Output Destinations & Message Codes */
    pvm_setopt( PvmSelfTraceTid, xpvm_tid );
    pvm_setopt( PvmSelfTraceCode, 666 );
    pvm_setopt( PvmSelfOutputTid, xpvm_tid );
    pvm_setopt( PvmSelfOutputCode, 667 );

    /* Set Future Children's Trace & Output Dests & Codes */
    /* (optional) */
    pvm_setopt( PvmTraceTid, xpvm_tid );
    pvm_setopt( PvmTraceCode, 666 );
    pvm_setopt( PvmOutputTid, xpvm_tid );
    pvm_setopt( PvmOutputCode, 667 );

    /* Generate Default Trace Mask */
    TEV_INIT_MASK( trace_mask );
    TEV_SET_MASK( trace_mask, TEV_MCAST0 );
    TEV_SET_MASK( trace_mask, TEV_SEND0 );
    TEV_SET_MASK( trace_mask, TEV_RECV0 );
    TEV_SET_MASK( trace_mask, TEV_NRECV0 );
    /* Add Other Desired Events Here */

    /* Set Self Trace Mask */
    pvm_settmask( PvmTaskSelf, trace_mask );

    /* Set Future Children's Trace Mask */
    /* (optional) */
    pvm_settmask( PvmTaskChild, trace_mask );
}

else
    printf( "XPVM not running, cannot trace\n" );
```

Note that this code segment is specifically constructed (including the ordering of the various PVM calls) to work with the PVM 3.3 tracing facility and XPVM 1.0. These calls must be executed in the order shown above. Any deviation from the above code segment, other than the addition or deletion of routines in the trace mask, may result in a failure to properly trace an application.



davinci:hitc_slave, TID=0x4000
bigblu0:pvmgs, TID=0x100003
bigblu0:hitc_slave, TID=0x1000
msr:hitc_slave, TID=0x80004
baloo0:hitc, TID=0x140003
baloo0:hitc_slave, TID=0x1400
sun4:hitc_slave, TID=0xc0004
gollum:hitc_slave, TID=0x1800

Figure 9: KILL Dialog

3.3.5. Killing Tasks

XPVM supports the PVM console "kill" command for killing PVM tasks. This function is accessed by selecting the "KILL" button on the "Tasks" menu, which pops up the KILL dialog box (see Figure 9). To kill a task, simply click on the desired task in the scrolling task list, and the task will be killed using the standard pvm_kill() routine. All of the currently executing tasks can be killed using the (genocidal) "Kill All" button, which functions similarly to a PVM Reset. In either case, the task(s) are killed as soon as the mouse click occurs, without further prompting.

XPVM avoids the overhead of continuously polling for PVM tasks while the KILL dialog is up, so the tasks listed may not accurately reflect the existence of all currently executing tasks. To update the task list, press the "Refresh Tasks" button on the KILL dialog.

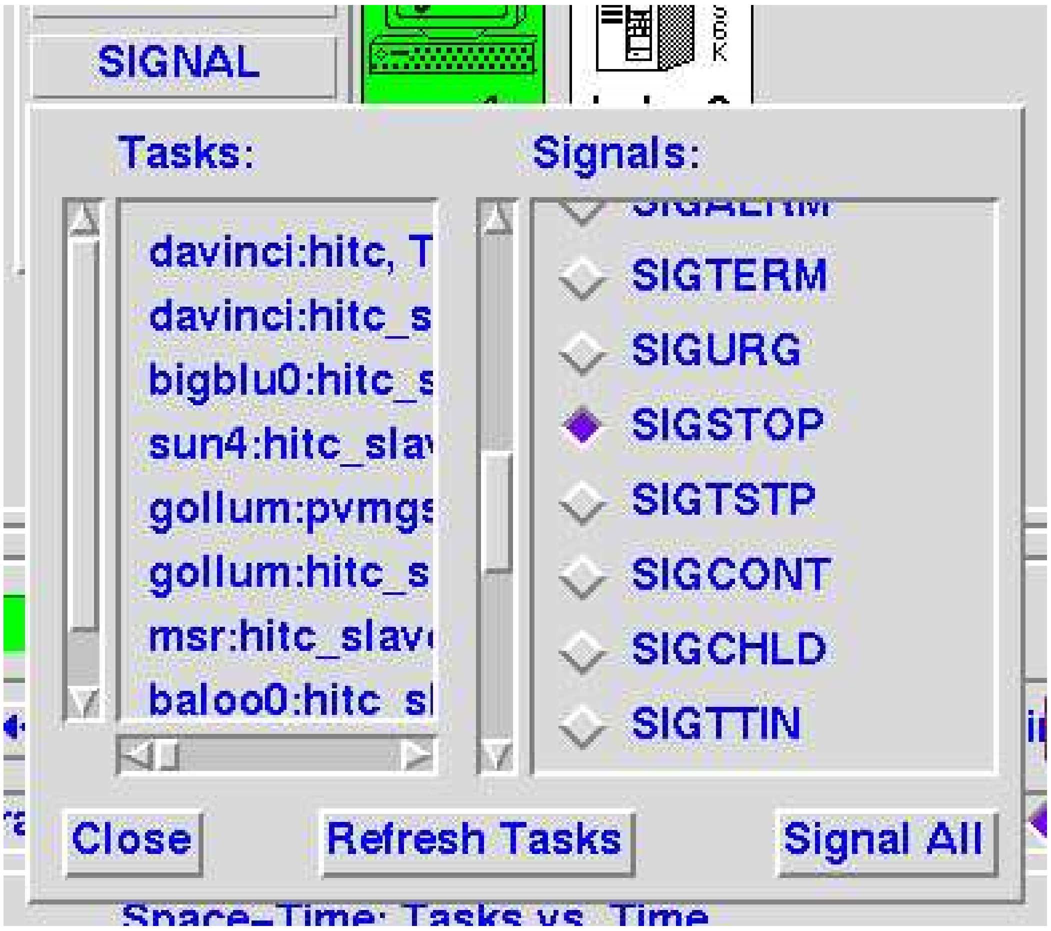


Figure 10: SIGNAL Dialog

3.3.6. Signaling Tasks

XPVM supports the PVM console "sig" command for sending Unix signals to PVM tasks. This function is accessed by selecting the "SIGNAL" button on the "Tasks" menu to raise the SIGNAL dialog box (see Figure 10). To signal a task, first select the desired Unix signal from the scrolling signals menu on the right side of the dialog box. Then choose the desired destination task(s) for the signal. A single task is signaled by clicking on one of the tasks in the scrolling task list on the left side of the dialog. All of the currently executing tasks can be signaled using the "Signal All" button. In either case, the signal(s) are sent as soon as the mouse click occurs, without further prompting.

XPVM avoids the overhead of continuously polling for PVM tasks while the SIGNAL dialog is up, so the tasks listed may not accurately reflect the existence of all currently executing tasks. To update the task list, press the "Refresh Tasks"

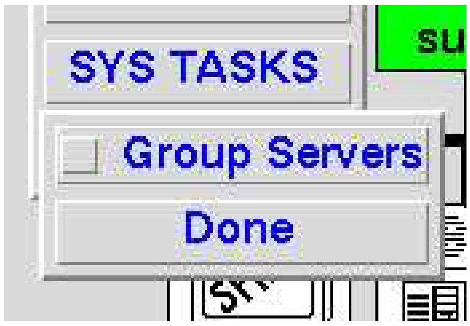


Figure 11: System Tasks Menu

button on the SIGNAL dialog.

3.3.7. System Tasks

In PVM, some tasks are not directly spawned by the user, but are created as needed by the system to serve certain purposes. XPVM can also capture trace events from some of these system tasks, if desired. By selecting the “SYS TASKS” button on the “Tasks” menu, as shown in Figure 11, a sub-menu will be raised showing a list of the system tasks that XPVM currently can monitor. Each system task is listed by name, with a square status box next to it, on the left. If the status box is filled in (purple) then that system task is to be monitored.

The only system task that XPVM 1.0 monitors is the PVM Dynamic Group Server. Note that XPVM automatically executes the group server on startup, and restarts the group server after any XPVM-initiated PVM Resets. The Group Server is needed by XPVM to allow trace event capture from tasks not spawned by XPVM (see Section 3.3.4 on Non-XPVM Spawning). By selecting the Group Server for monitoring, a user can observe the interaction of an application with the Group Server, including messages sent back and forth to implement PVM Group functionality.

3.4. XPVM Options

There are several run-time options which can be set for XPVM, These options include setting the “Action Mode,” which determines how the user interacts with XPVM to query views for details or correlate views to specific times. The “Task Sorting” option provides different ways for the user to arrange or order the tasks in the views, including custom user placement. These options are set via the “Options...” menu, as shown in Figure 12.

These features are described in more detail in the following subsections.

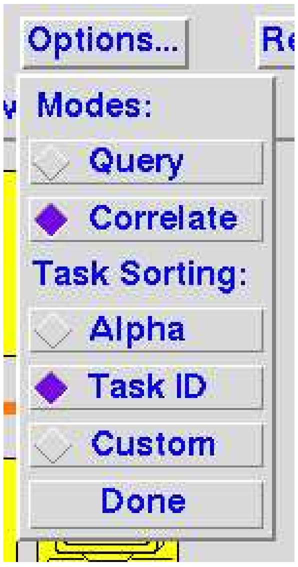


Figure 12: Options Menu

3.4.1. User Action Modes

There are two action modes currently supported in XPVM, “Query” mode and “Correlate” mode. In standard “Query” mode, all views function normally and clicking in any view with the mouse pointer will provide information regarding the details of particular view elements. In the “Correlate” mode, however, the user can defer normal view operation temporarily to *go back in time* and examine earlier program state. By clicking in either of the “Space-Time” or “Utilization” view canvases while in “Correlate” mode, the user selects a “correlate time.” Then the “Network,” “Message Queue” and “Call Trace” views will redraw back to that point in time to provide a more accurate picture of what was happening at that moment.

Typically, because trace events arrive at XPVM slightly out of order due to network delays, these views may not reflect the exact state of your system during standard real time trace play. The “Correlate” mode allows the user to go back and freeze the views at a specific time so that the *actual* program state can be observed (with the benefit of 20/20 hindsight).

The user can hold down the mouse button while selecting the “correlate time” to scroll back and forth through time. If the user “double-clicks” in the “Space-Time” or “Utilization” view canvases while in “Correlate” mode, the “correlate time” will be locked in for detailed analysis of the given time. Another click in either view will release the lock and allow selection of another “correlate time” or return to normal trace play.

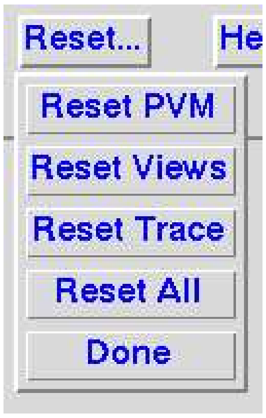


Figure 13: Reset Menu

3.4.2. Task Sorting

Often it is useful to examine sets of tasks in some particular order which is relevant to a high-level, algorithmic arrangement that exists, the order of their creation, or some other arbitrary user requirement. The user can select from three modes of task sorting, including "Alphabetical," "Task ID" or "Custom." In "Alphabetical" task sort mode, which is the default, tasks are sorted alphabetically, first by host name and then by task name. In "Task ID" sort mode, the tasks are sorted numerically by their task IDs, representing their spawn order by host. In "Custom" task sort mode, the user may move tasks around to place them in any desired order. In this mode, the tasks will initially be arranged in the order in which they are received by XPVM. Then the user can "grab" tasks, "drag" them to their new position, and "drop" them into place. This dragging and dropping is done using the task labels in either the "Space-Time" or "Call Trace" view. Normally, clicking on the task labels in these views will pop up a small window with the full task name and ID, but in "Custom" task sort mode these boxes can be dragged up or down to re-arrange the tasks.

3.5. Resetting PVM & XPVM

There are several reset functions provided by XPVM, to allow precise control over different portions of the monitoring system. These are available from the pull-down "Reset..." menu, as shown in Figure 13.

The functions provided are as follows:

Reset PVM: This function performs the same as the "reset" command in the regular PVM console. It resets the virtual machine, killing off all tasks (except

XPVM and any other PVM consoles) and then kills the PVM Group Server(s) and removes all PVM groups. XPVM will then restart the default PVM Group Server and join its "xpvm" group. Note that any trace events already enroute at the time of the PVM Reset will continue to arrive, and XPVM will display their information.

Reset Views: This function clears all XPVM views and resets them to their initial state. Note that any PVM tasks running at the time of the view reset will not be killed, but will no longer be shown in the views. From the point of the view reset, only newly spawned tasks will be shown.

Reset Trace: This function initializes the current trace file for re-use, depending on the current tracing mode. In "Playback" mode the current trace file is rewound to the beginning, to be played again, without resetting the views. In "OverWrite" mode the current trace file is closed and the user is prompted as to whether this trace file can be overwritten (see Section 4.1). If the user selects "Yes" to overwrite the trace file, then the file is cleared and re-opened to save a new trace. The views are not reset, however, and the user may "Start Append" to add additional events to the current view state. Rewinding the trace file in this case will show only the events occurring after the Trace Reset.

Reset All: This all-encompassing reset function applies all of the above resets in an appropriate order to bring XPVM back to its initial startup state. All of PVM, the views and the trace file are reset or initialized.

4. XPVM Monitor Functions

XPVM serves as a real time performance monitor for PVM. The monitor consists of correlated views that allow the user to compare different information about a particular occurrence or time in the program execution. The following subsections discuss the operation of the XPVM monitor functions in more detail, including the use of trace files and full descriptions of each of the monitoring views.

4.1. Trace Files & Trace Control in XPVM

XPVM receives task tracing information in the form of distinct trace event messages via the standard PVM message passing mechanism. PVM uses the existing message channels and PVM Daemons to route the trace event messages.

There are two distinct trace playing modes in XPVM, "Trace OverWrite" mode and "Trace Playback" mode. "Trace OverWrite" mode is used to play traces in "real time," as they are collected, and "Trace Playback" mode allows analysis of traces "post-mortem," by playing back saved trace files.

To select between Trace OverWrite mode and Trace Playback mode in XPVM, there are two buttons on the right side of the trace control panel. For XPVM to save events from currently executing tasks into a trace file, it must be in OverWrite mode, where the trace is played in "real time." Once a trace file has been created in OverWrite mode, the user can switch to Playback mode to replay the trace for post-mortem analysis.

XPVM always saves trace events directly into a trace file, and reads from the trace file using the same mechanism for either mode. To this end, it is always necessary to have a valid, writable trace file to work with, and so XPVM assumes a default trace file name:

```
/tmp/xpvm.trace.$(USER)
```

where "$(USER)" is the login name of the current user of XPVM.

It should be noted that when running in real-time monitor mode, XPVM must receive and decode PVM trace event messages at the same time it is presenting the animated views of program activity. To simplify these operations, they are divided into separate tasks that "share" XPVM's processing time. To avoid significant intrusion into the execution of the PVM programs being monitored, as well as reduce message queue back-up, the priority is given first to unpacking, decoding and saving trace events in the trace file. The reading of events from the trace file to drive the views is secondary, but will interrupt the trace message processing where necessary to maintain some level of fluid animation.

On the first execution of XPVM, the trace file will likely not yet exist, and so XPVM will proceed, without any prompting, to write traces into this file and read them back for trace replay. On each new spawn, however, (as invoked via "Start" from the XPVM SPAWN dialog, see Section 3.3.1), the trace file is overwritten with a new program execution trace. After the first spawn the default trace file will already exist and will contain trace event records, so the user will be prompted before it is overwritten with a new spawn's trace events.

This trace file overwrite prompting takes place in a special dialog box that pops up as necessary to prevent the unintended overwriting. This dialog will be invoked if there are any significant trace events in the given trace file. (Any XPVM-generated trace headers or host status events are ignored when overwriting trace files.) The overwrite dialog contains the name of the current trace file and provides two choices, "Yes" or "No," to determine the fate of the trace file. If the user presses the "Yes" button, the trace file is overwritten with a new trace. If the "No" button is pressed,

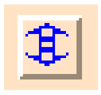


Figure 14: Network / Space-Time Real Estate Button

then the file is not overwritten, and XPVM is forced into "Playback" mode, as then it does not have a trace file location for recording the monitoring events.

The user may change the name of the trace file by editing the "Trace File" entry in the trace control panel of the main window. This trace control panel is located between the "Network" and "Space-Time" views if both are present. Note that the *real estate* in the main window can be appropriately allocated between those two views by grabbing on a button at the left of the trace control panel and dragging it up or down. This button is shown in Figure 14.

The trace files for XPVM are generated in "self-defining data format," or SDDF [1], as designed for representing trace events in the "Pablo" system system [7]. SDDF was chosen over the PICL format [8], as used by the widely known "ParaGraph" system [2], because PICL was not sufficiently flexible to represent all the trace information generated by PVM. SDDF is a de facto standard in the program visualization community, and presents a general trace format, allowing arbitrary data structuring and including semantic information.

Using SDDF, XPVM's trace files include descriptor information for each event type, allowing a variety of analysis tools to interpret and use the trace event data. A list of the trace event descriptors for XPVM trace events is provided via the "-S" option, e.g. "xpvm -S."

In addition to the trace play modes, there are also individual "Play" (>), "Fast-Forward" (>>), "Stop" (□), "Single Step" (| >), and "Rewind" (<<) buttons on the trace control panel, for controlling the actual trace play.

When in "Play" mode, trace events are read, interpreted and graphically depicted continuously until all events have been processed. If no events are currently pending, and XPVM knows that there are still PVM tasks alive, then the trace time indicator will be incremented to follow wallclock time. This represents the passage of time during periods when no trace events are being generated by the PVM tasks.

In "Fast-Forward" mode, trace events are read and interpreted in the same manner as in "Play" mode, but the graphical depiction of trace events in the views is disengaged. To illustrate this in the interface, small pop-up boxes appear in each view to indicate that XPVM is "Fast-Forwarding..." and not drawing any view details. By avoiding the overhead of drawing each individual trace event and updating the screen, this mode greatly increases the speed with which traces can be explored. When the end of the trace file is reached, or when another trace control is selected to switch play modes or stop trace play, XPVM automatically drops out of "Fast-Forward" mode and redraws all of the views with the current display information.

Pressing the "Stop" button image (□) temporarily halts the playing of the trace, and waits for input from the user before proceeding. If trace events are still being generated by PVM tasks, they will continue to be captured and saved into the trace file for playback. Resuming the trace play by pressing Play will simply continue on from where Stop was pressed, remaining in "real time" OverWrite mode. Clearly, XPVM will be behind "real time" at this point, but if the frequency of new trace events is low enough or if "Fast-Forward" mode is used, XPVM can catch back up to true "real time" monitoring.

Pressing the "Single Step" button image (| >) reads a single trace event from the trace file and processes it. After each invocation of Single Step, the trace play ceases and enters the Stop state. It should be noted that for certain trace events, the processing of the event may not produce any visible effect on any of XPVM's views. For example, the trace file likely contains internally generated "host synchronization" events that allow XPVM to help account for clock skew among different hosts. These events do not produce any visible effect, but will still count as a "single" event for the Single Step. Similarly, any task output events will not produce any visual indication if the "Task Output" view is not active, and so on.

Pressing the "Rewind" button image (<<) resets XPVM to the beginning of the current trace file for replay. If XPVM is in "OverWrite" mode then the trace file will be closed and any pending events residing in XPVM's message queue will not be saved in the trace. XPVM will always be set to "PlayBack" mode when "Rewind" is selected.

Aside from the various trace controls on the trace control panel, there is an additional "Time" display that indicates the relative time from the given trace file. The latest time stamp from the trace event sequence is always displayed by the "Time" label on the trace control panel. Note that because events are collected through PVM's message channels from independent remote tasks, they are not necessarily globally ordered in time. Some events will appear to occur "back in time" due to this lack of ordering. Where possible, the views in XPVM compensate for this out-of-time ordering, and will re-arrange themselves to correctly reflect any new information from the "past." XPVM is also tolerant of clock skew, which can cause seemingly "impossible" events to be displayed, such as a message arriving before it is sent. These "tachyon" events are solely a result of the level of fine-grained synchronization between distributed hosts, and currently are not removed or fixed

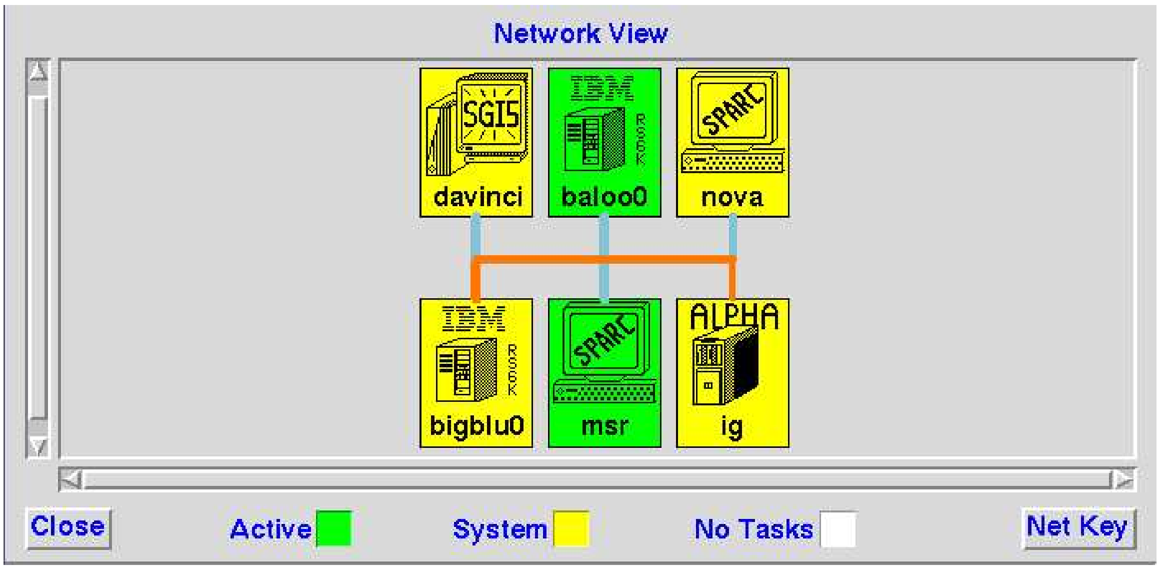


Figure 15: Network View Interface

in XPVM 1.0.

4.2. Network View

The “Network” view displays high-level activity on the hosts in the virtual machine, as shown in Figure 15. Each host is represented by an icon image which shows the architecture and includes the name of the host. The current “Network” view assumes a bus or ethernet topology, and arranges the host icons along a horizontal bus. The icons are illuminated in different colors to indicate the status of the tasks running on each host.

The “Active” color implies that at least one task on that host is busy executing useful user computation. Due to the built-in nature of the tracing facility in PVM 3.3, the only information known about an application is that obtained from PVM system calls. Therefore, “useful user computation” is categorized as any computation outside of a PVM system call. The “System” color means that no tasks are busy executing useful user computation, but that at least one task is busy executing PVM system routines. When there are no tasks executing on a given host, its icon is left uncolored, i.e. colored white or in the background color. The specific colors used for the Active and System cases default to green and yellow, respectively, but are user customizable.

The “Network” view also contains animated depictions of the instantaneous network bandwidth and volume on individual network links. The links from each host to the bus, and the individual bus links, are all animated. The width of a particu-

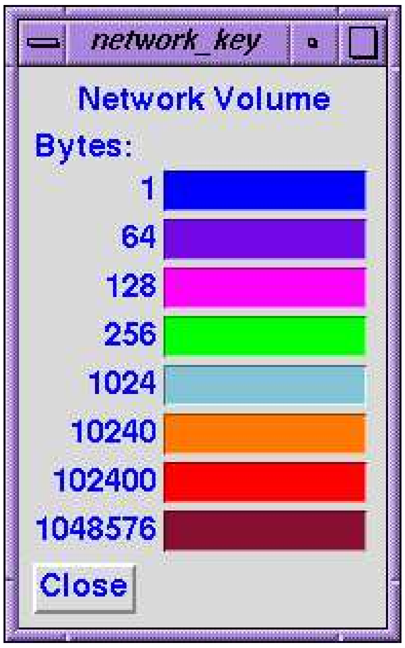


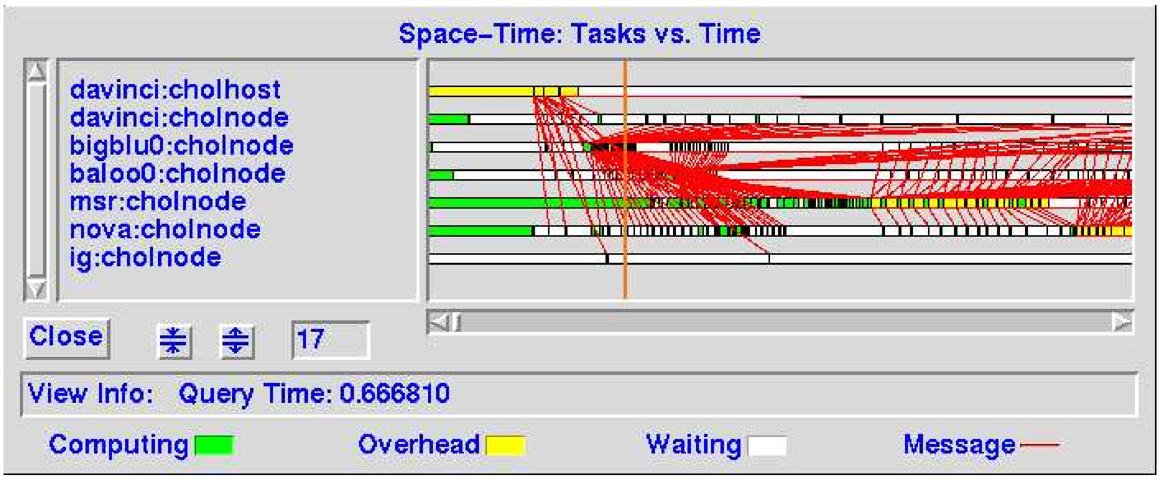
Figure 16: Network Volume Key

lar link corresponds to the bandwidth most recently realized on that link, and the message volume is shown by the color.

The bandwidth is calculated upon message receipt, because it is not known in advance how long a particular message will take to be transmitted. Specifically, the bandwidth is the average communication rate in bytes per second, equal to the ratio of total message size to total travel time. The bandwidth can be used to determine the load and performance of the network, thereby providing insight into the impact the network is having on the application's performance.

The message volume represents a wide spectrum of ranges from bytes to kilobytes to megabytes. The volume is computed on-the-fly by adding and subtracting the size of messages as they enter and leave the network. This information provides an indication of the existence and magnitude of any communication bottlenecks in the application. As mentioned in Section 2.3.1, PVM 3.3 does not provide the actual message size in bytes in send events, so this message volume represents the number of "default-sized" messages in transit. PVM 3.4 will provide actual message size information to improve the accuracy and usefulness of this feature. The color key for the network volume is shown in Figure 16.

4.3. Space-Time View



View Info: Query Time: 0.666810

Figure 17: Space-Time View Interface

The “Space-Time” view shows the status of individual tasks as they execute across all hosts, as shown in Figure 17. Each task is represented by a horizontal bar along a common time axis, where the color of the bar at each time indicates the state of the task.

The “Computing” color shows those times when the task is busy executing useful user computations, that is not executing PVM system calls. The “Overhead” color marks the places where each task executes PVM system routines for communication and task control. The “Waiting” color indicates those time periods spent waiting for messages from other tasks, i.e. the idle time due to synchronization. Communication activity among tasks is also shown, using lines drawn between the task bars at the corresponding message send and receive times. The default colors for the task states are green for “Computing,” yellow for “Overhead,” and white for “Waiting.” The default color for messages is red. A vertical blue line marks the latest trace event time, as shown in the trace control panel “Time” display (see Section 4.1 on Page 29).

4.3.1. Space-Time Queries

More detailed information regarding specific task states or messages can be extracted from the “Space-Time” view by clicking and holding on the view area with the left mouse button. (This is also how the “correlate time” is selected when in “Correlate” mode, see Section 3.4.1 on Page 24). When a task bar is clicked on, the information line at the bottom of the view changes to display the precise times that the task state began and ended, plus information on the content of that state. For “Computing” states, this information contains the results and return code of the last PVM system call for that task, i.e. the call which returned at the beginning of that "Computing" state. For "Overhead" states, the information shows the calling parameters of the PVM system routine that was called at the start of the state. Individual messages can be clicked on as well to determine the time when each send was initiated and the time when the message receipt was completed. Also included is the number of bytes sent and the message code.

Any time the mouse pointer enters the main Space-Time canvas, a blue vertical line appears to mark a constant time axis in the view. The actual time represented by the location of this line is displayed as a "Query Time" in the information line at the bottom of the view. Note that the "Utilization" and "Space-Time" views are tightly coupled, such that whenever a time line is shown in one view, the other view will produce a similar line. This line identifies the same time instant in each view's display area, to help correlate the two representations.

4.3.2. Space-Time Zooming

The "Space-Time" view can be "zoomed" in or out to display different perspectives on the same trace data. To zoom in on a particular area of the "Space-Time" view, click and drag out an area on the main "Space-Time" canvas using the middle mouse button. Moving the mouse pointer left or right will sweep out an area that is to be blown up. When the mouse button is released the swept area will be enlarged horizontally to fill the entire "Space-Time" canvas area. To "unzoom" back to the previous zoom level, or out to some lower magnification level if no zoom had been previously applied, simply press the right mouse button.

To summarize the "Space-Time" mouse controls:

`Mouse Left`	`Mouse Middle`	`Mouse Right`
`Query`	`Zoom IN`	`Zoom Out`

Like the "Query Time" lines above, the "Utilization" view is automatically zoomed in or out to match any changes in the "Space-Time" view's time scale.

4.3.3. Space-Time Task Height

In addition to the horizontal zooming provided to control the level of time detail in the "Space-Time" view, the vertical axis of the view can also be scaled. In cases where either very large or very small numbers of tasks are being displayed, the height of the horizontal task bars can be shrunk and grown to appropriately utilize the screen area. These functions are controlled via two buttons and a text entry at the bottom of the "Space-Time" view, as shown in Figure 18. Clicking on one of the

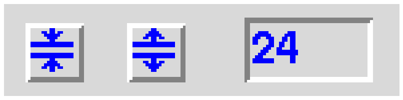


Figure 18: Task Height Interface

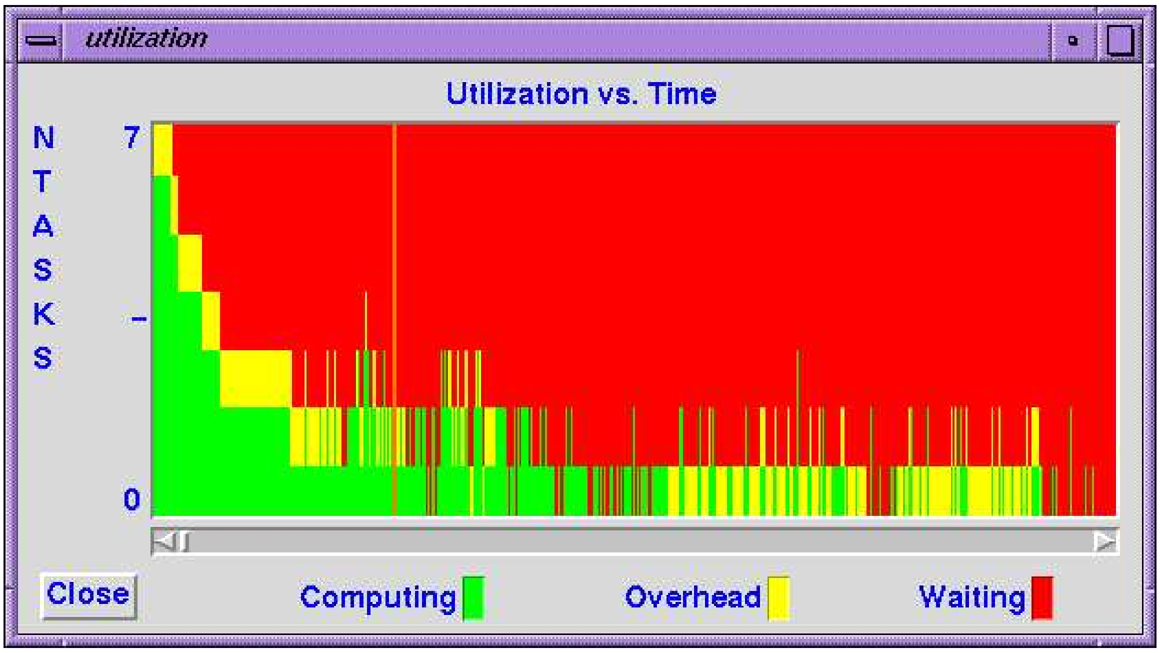


Figure 19: Utilization View Interface

buttons will appropriately shrink or grow the height and spacing of each task state rectangle to either fit more or fewer tasks in the view, respectively. Alternately, the user can simply enter an integer value in the text entry to the right of these buttons, and manually set the precise height (in pixels) of each task line area.

Each task's line area can always be seen by scrolling the “Space-Time” window, so the task height need only be adjusted for convenience. Note that changing the task height in the “Space-Time” view also affects the height and spacing of tasks in the “Call Trace” view (see Section 5.1 on Page 37).

4.4. Utilization View

The “Utilization” view summarizes the “Space-Time” view at each time instant, showing the number of tasks in each of three possible task states: busy “Computing,” in “PVM Overhead,” or idle “Waiting” for a message (see Figure 19). This information is represented by vertical stacks of up to three colored rectangles. One stack is used to represent each time instant and one rectangle is shown for each of the possible task states, with Computing on the bottom, Overhead in the middle, and Waiting on top. The height of each rectangle in the stack is proportional to the number of tasks in that state versus the total number of tasks executing at that time instant. The default colors for the state rectangles are green for "Computing," yellow for "Overhead," and red for idle "Waiting."

The overall utilization for tasks is seen by comparing the relative height of the rectangles over time, such that in cases of good utilization the "Computing" areas are prominent, and for poor utilization the "Overhead" and "Waiting" areas dominate. The "Utilization" view instantaneously scales to the total number of tasks encountered, to provide an accurate representation of the overall utilization. The utilization rectangles fill in from the left as each task's state become known, resulting in a "tapered" leading edge to the right. As tasks complete execution, empty space is left above the rectangle stacks to indicate the number of tasks which are still executing.

As mentioned above, the "Utilization" and "Space-Time" views are tightly coupled and share the same horizontal time scale. Entering either view area with the mouse pointer creates a blue "Query Time" line in both views.

4.4.1. Utilization Zooming

The "Utilization" view can be "zoomed" in or out just like the "Space-Time" view, and the mouse controls to do so are identical. To zoom in on a particular area of the "Utilization" view, click and drag out an area on the main "Utilization" canvas using the middle mouse button. To "unzoom" back to the previous zoom level, press the right mouse button.

To summarize the "Utilization" mouse controls:

```
Mouse Middle        Mouse Right
  Zoom IN             Zoom Out
```

Again, the "Utilization" and "Space-Time" views are correlated and zooming the "Utilization" view automatically zooms the "Space-Time" view to match the "Utilization" view's time scale.

4.5. Message Queue View

The "Message Queue" view shows the instantaneous number and size of messages that are buffered, waiting to be received, for each task. The view consists of a collection of "message stacks," one for each task, that represent the pending messages. Each stack is made up of rectangles that represent the individual pending

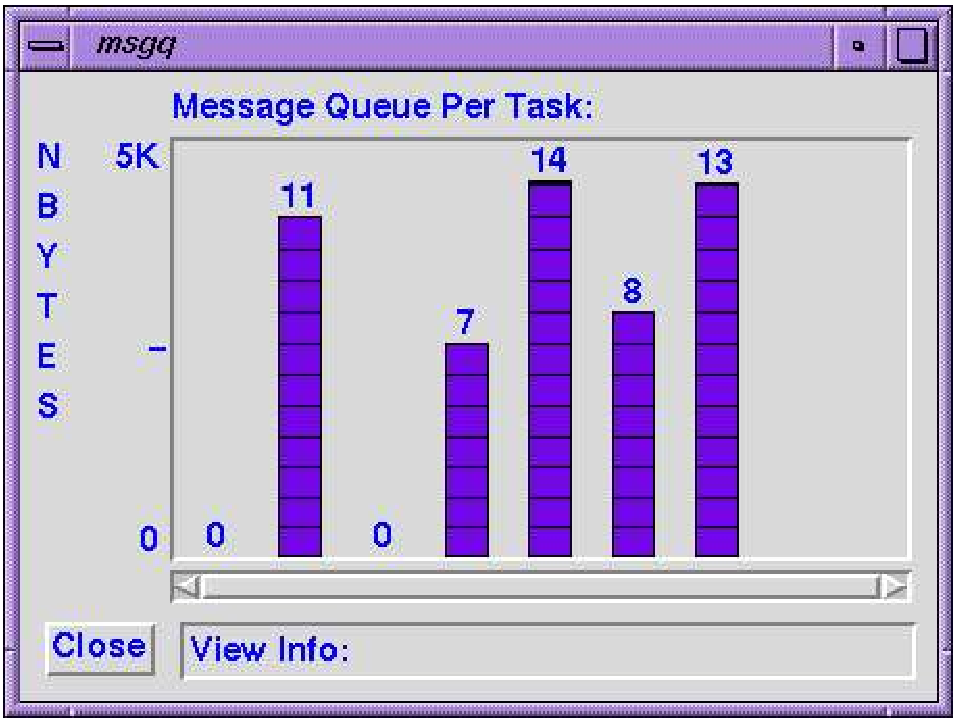
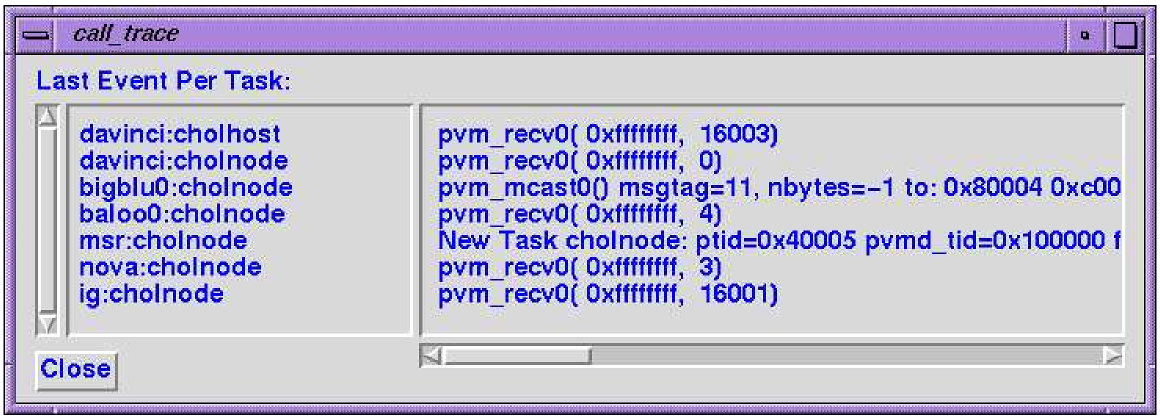


Figure 20: Message Queue View Interface

messages. This information is especially useful in identifying communication "hot spots," much like the message volume animation in the "Network View," but at a higher resolution. In the "Message Queue" view the message bottlenecks among individual tasks can be observed. This view is also "clickable" and "double-clickable" to reveal and closely examine detailed textual information for individual message rectangles. This information includes the source task of the message, the message code, and the number of bytes (if known, see below). A sample of this view is shown in Figure 20.

It should be noted that proper operation of the "Message Queue" view requires information not provided in PVM 3.3 traces. To depict the number of bytes in pending messages, their size must be known at send time. In the PVM 3.3 tracing facility, this information has (unfortunately) been omitted, and so precludes the use of the actual message size. A default message size can be set using the "-M *nbytes*" command line option to XPVM (see Section 2.3.1 on Page 7).

To distinguish the default message size from actual message sizes, different colors are used for the message rectangles. Messages drawn in orange have only the default message size information, whereas messages drawn in purple are accurately drawn to represent the actual message size. For PVM 3.3 usage, the only way in which to obtain actual message sizes is via the "Correlate" action mode (see Section 3.4.1 on



Last Event Per Task:

```
davinci:cholhost    pvm_recv0( 0xffffffff,  16003)
davinci:cholnode    pvm_recv0( 0xffffffff,  0)
bigblu0:cholnode    pvm_mcast0() msgtag=11, nbytes=-1 to: 0x80004 0xc00
baloo0:cholnode     pvm_recv0( 0xffffffff,  4)
msr:cholnode        New Task cholnode: ptid=0x40005 pvmd_tid=0x100000 f
nova:cholnode       pvm_recv0( 0xffffffff,  3)
ig:cholnode         pvm_recv0( 0xffffffff,  16001)
```

Figure 21: Call Trace View Interface

Page 24). Using this mode, the user can *go back in time* and see the state of each task's message queue accurately, with the benefit of 20/20 hindsight.

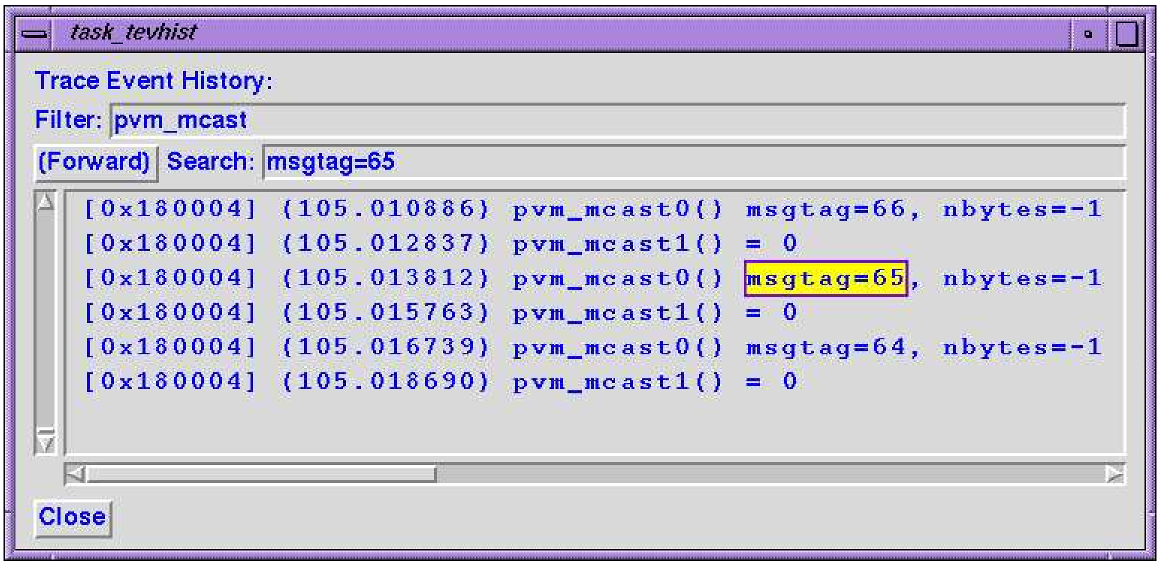
5. XPVM Debugging Features

XPVM provides some simple debugging features which allow access to textual details from trace events and remote task output. There are three views provided for accessing this information, as described in the following subsections.

5.1. Call Trace View

The "Call Trace" view provides a textual record of the instantaneous activity in each task, as shown in Figure 21. For each task, a single line of text is displayed that represents the last PVM system call made by that task, including any calling parameters or result details. As the tasks execute, the text changes to reflect the most recent activity for each task. This view is especially useful in identifying points where a PVM program's execution "hangs" in one or more tasks, say, due to an incorrect message code in a blocking pvm_recv() call.

The "Call Trace" view can be scrolled to view information for large numbers of tasks. In addition, the height and spacing of tasks can be adjusted via the "Space-Time" view controls (see Section 4.3 above). Clicking on a task label in the "Call Trace" view will produce a small pop-up window with the full name of the task (helpful if not fully visible) as well as the task ID of the task and the name of the host on which it executes. Clicking on the actual event text for a task will display the timestamp for that event in the view info display. When in "Custom" task sort mode, clicking on and dragging a task label will re-arrange the order of the task in



Filter: pvm_mcast

Search: msgtag=65

```
[0x180004] (105.010886) pvm_mcast0() msgtag=66, nbytes=-1
[0x180004] (105.012837) pvm_mcast1() = 0
[0x180004] (105.013812) pvm_mcast0() msgtag=65, nbytes=-1
[0x180004] (105.015763) pvm_mcast1() = 0
[0x180004] (105.016739) pvm_mcast0() msgtag=64, nbytes=-1
[0x180004] (105.018690) pvm_mcast1() = 0
```

Figure 22: Event History View Interface

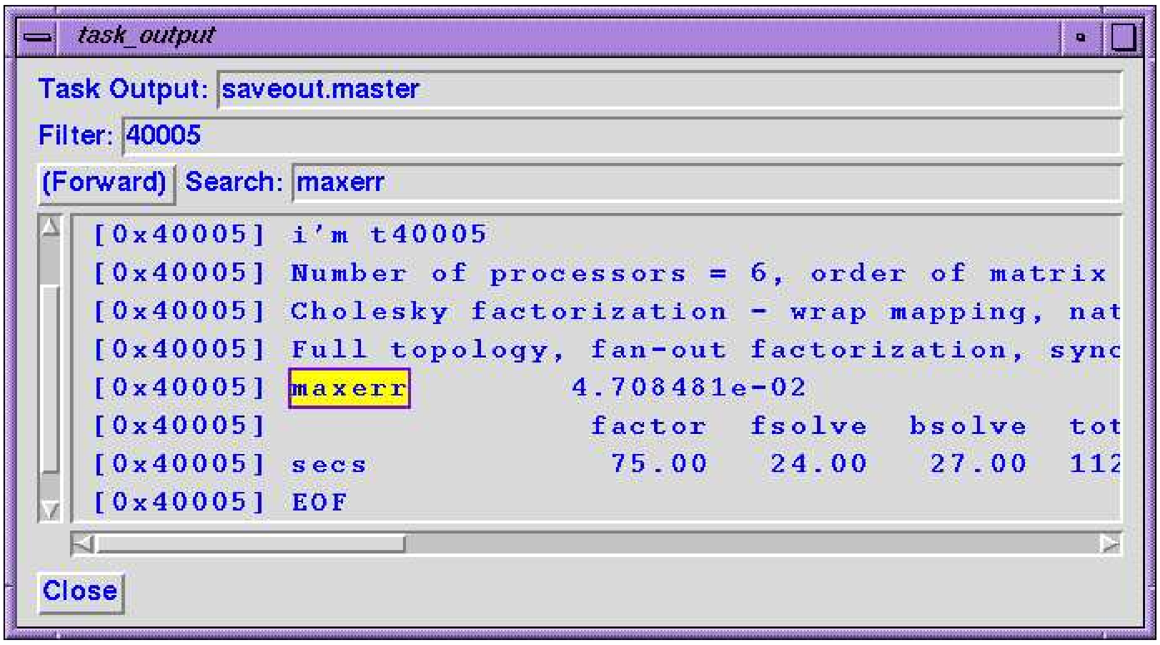
the list (see Section 3.4.2 on Page 25).

5.2. Event History View

The "Event History" view provides access to *all* trace events that have currently been processed in a trace file, as shown in Figure 22. Each line produced in the "Call Trace" view above is concatenated onto a list in a scrolling text window, with the events arranged chronologically in order of arrival at XPVM. Each line includes the task ID for the event and the timestamp for when it occurred, in addition to the standard event description text.

The scrolling text window for the "Event History" view includes several features for manipulating and searching the text. By entering a "Filter" string, the user can select a subset of text lines for easier analysis of specific tasks or events. Only lines which contain the "filter string" are displayed. All text lines can again be displayed by clearing the "filter string" and setting it to "" – or the *empty string*.

Likewise, the user can search through the displayed text lines for specific occurrences by entering a "search string" at the "Search" prompt. Each time the user presses a "<Return>" in the "Search" entry, the next occurrence of the given "search string" is located in the displayed text and highlighted in a yellow box. The search will either start at the first displayed text line in the scrolling window, or will begin with the second character in the current highlighted search match. If the search reaches the bottom of the text list, the search will wrap around back up to the top of the list. The scrolling window automatically adjusts to include each search



```
[0x40005] i'm t40005
[0x40005] Number of processors = 6, order of matrix
[0x40005] Cholesky factorization - wrap mapping, nat
[0x40005] Full topology, fan-out factorization, sync
[0x40005] maxerr        4.708481e-02
[0x40005]                factor  fsolve  bsolve  tot
[0x40005] secs            75.00   24.00   27.00  112
[0x40005] EOF
```

Figure 23: Task Output View Interface

match. The direction of the search through the text can be changed by toggling the "(Forward)" or "(Backward)" label next to the "Search" entry prompt.

In either the "filter string" or the "search string," the user can enter plain text or full regular expressions (as fully documented in the TCL man page for "RegExp.3"). For example, if the user wishes to filter out the events of just a single task, with task ID of 40003, the "filter string" would be "\[0x40003\]" with the "[" and "]" each *escaped* with a "\" to be taken literally.

5.3. Task Output View

The "Task Output" view provides access to the output generated by remotely executing tasks (such as diagnostic prints), and displays each line of output in a scrolling text window, as shown in Figure 23. The task output is automatically captured by XPVM whenever a spawned task writes to the "stdout" standard output device (see Section 3.3.3 on Page 19). The output is captured whether the "Task Output" view is active or not. In fact, the "Task Output" view can be activated at any time during a program's execution, providing access to all output captured up to that point.

If the user wishes to save task output into a file, a file name can be entered at the "Task Output" prompt in the "Task Output" view window. Then all currently displayed (or subsequent) output will be saved into that file in addition to being dumped onto the scrolling window.

The scrolling text window for the "Task Output" view includes several features for manipulating and searching the text. By entering a "Filter" string, the user can select a subset of text lines for easier analysis of specific tasks or events. Only lines which contain the "filter string" are displayed. All text lines can again be displayed by clearing the "filter string" and setting it to "" – or the *empty string*.

Likewise, the user can search through the displayed text lines for specific occurrences by entering a "search string" at the "Search" prompt. Each time the user presses a "<Return>" in the "Search" entry, the next occurrence of the given "search string" is located in the displayed text and highlighted in a yellow box. The search will either start at the first displayed text line in the scrolling window, or will begin with the second character in the current highlighted search match. If the search reaches the bottom of the text list, the search will wrap around back up to the top of the list. The scrolling window automatically adjusts to include each search match. The direction of the search through the text can be changed by toggling the "(Forward)" or "(Backward)" label next to the "Search" entry prompt.

In either the "filter string" or the "search string," the user can enter plain text or full regular expressions (as fully documented in the TCL man page for "RegExp.3"). For example, if the user wishes to filter out the output of just a single task, with task ID of 40003, the "filter string" would be "\[0x40003\]" with the "[" and "]" each *escaped* with a "\" to be taken literally.

6. Future Work

The latest release of XPVM, Version 1.2, is fully forward-compatible with the new tracing facility which will be available in the upcoming PVM 3.4. This new facility supports the buffering of trace events and user-defined trace events, and allows significantly more flexible maintenance of trace event contents in PVM (such as being able to easily add the message size in "send" events, to drive the special features of the "Network" and "Message Queue" views).

The "Network View" will also be enhanced to support a wider range of information regarding network usage and impact. The arrangement of host icons will be appropriately modified to represent a selection of different network topologies. The capability will also be added to zoom in on multiprocessor systems in the "Network" view to see exploded sub-views of the local communication among task nodes in MPPs and shared memory systems.

Other views and enhancements will also be added in the future to support analysis of the new constructs in PVM 3.4, such as context, message mailboxes and message handlers.

7. Correspondence

Mail any questions, problems or suggestions to the author of XPVM, Jim Kohl, at kohl@msr.epm.ornl.gov. Requests are typically answered promptly the same day, or at least within a few days worst case.

8. References

[1] R. A. Aydt, "The Pablo Self-Defining Data Format," University of Illinois at Urbana-Champaign, Department of Computer Science, February 1993.

[2] M. T. Heath, J. A. Etheridge, "Visualizing the Performance of Parallel Programs," *IEEE Software*, Volume 8, Number 5, September 1991, pp. 29-40.

[3] B. A. Myers, "The State of the Art in Visual Programming and Program Visualization," Report Number CMU-CS-88-114, Computer Science Department, Carnegie Mellon, 1988.

[4] J. K. Ousterhout, "Tcl: An Embeddable Command Language," 1990 Winter USENIX Conference.

[5] J. K. Ousterhout, "An X11 Toolkit Based on the Tcl Language," 1991 Winter USENIX Conference.

[6] G.A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, V.Sunderam, "PVM: Parallel Virtual Machine, A User's Guide and Tutorial for Networked Parallel Computing," The MIT Press, 1994.

[7] D. Reed, R. Olson, R. Aydt, T. Madhyastha, T. Birkett, D. Jensen, B. Nazief, B. Totty, "Scalable Performance Environments for Parallel Systems," Proceedings of the Sixth Distributed Memory Computing Conference, IEEE Computer Society Press, April 1991.

[8] P. H. Worley, "A New PICL Trace File Format," Technical Report ORNL/TM-12125, Oak Ridge National Laboratory, Oak Ridge, TN, October 1992.