October 1993

Order Number: 312545-002

Paragon™ Application Tools User's Guide

Intel® Corporation

Revision	History	Date
-001	Original Issue	04/93
-002	Revision	10/93

WARNING

Some of the circuitry inside this system operates at hazardous energy and electric shock voltage levels. To avoid the risk of personal injury due to contact with an energy hazard, or risk of electric shock, do not enter any portion of this system unless it is intended to be accessible without the use of a tool. The areas that are considered accessible are the outer enclosure and the area just inside the front door when all of the front panels are installed, and the front of the diagnostic station. There are no user serviceable areas inside the system. Refer any need for such access only to technical personnel that have been qualified by Intel Corporation.

CAUTION

This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at his own expense.

Preface

This manual describes ParAide - the parallel application interactive development environment for the Paragon system. ParAide provides a comprehensive set of tools for application programmers who design applications for the Paragon system. ParAide also provides graphical tools that allow easy visual access to the application tools.

For window displays, menus, commands, buttons, keyboard accelerators, and dialog boxes, the Paragon graphical tools follow the Motif style guide. This manual assumes the Motif window manager functionality. The Paragon graphical tools can, however, be run under window managers other than the Motif window manager.

The sections of this manual that describe the X resources used to configure the Paragon graphical tools assume some familiarity with X resources.

Organization

Chapter 1	Describes ParAide, the graphical user interface that serves as a launching point to the other graphical tools in the Paragon toolset (SPV, XIPD, ParaGraph, XProf and XGprof), assists you in loading parallel applications, and allows you to open text files into an editor.
Chapter 2	Describes the Interactive Parallel Debugger (IPD).
Chapter 3	Describes XIPD, the graphical front end to IPD.
Chapter 4	Describes **prof** and **gprof**, the program profiling tools for the Paragon system.
Chapter 5	Describes XProf, the graphical front end to **prof**.
Chapter 6	Describes XGprof, the graphical front end to **gprof**.
Chapter 7	Describes ParaGraph, a graphical performance visualization tool for analyzing the performance of parallel applications on the Paragon system.
Chapter 8	Describes **pmake**, the parallel make utility for the Paragon system.

Notational Conventions

This manual uses the following notational conventions:

Bold	Identifies command names and switches, system call names, reserved words, and other items that must be used exactly as shown.
Italic	Identifies variables, filenames, directories, processes, user names, and writer annotations in examples. Italic type style is also occasionally used to emphasize a word or phrase.
`Plain-Monospace`	Identifies computer output (prompts and messages), examples, and values of variables. Some examples contain annotations that describe specific parts of the example. These annotations (which are not part of the example code or session) appear in *italic* type style and flush with the right margin.
`Bold-Italic-Monospace`	Identifies user input (what you enter in response to some prompt).
`Bold-Monospace`	Identifies the names of keyboard keys (which are also enclosed in angle brackets). A dash indicates that the key preceding the dash is to be held down *while* the key following the dash is pressed. For example: **`<Break>`** **`<s>`** **`<Ctrl-Alt-Del>`**
[ ]	(Brackets) Surround optional items.
...	(Ellipsis dots) Indicate that the preceding item may be repeated.
|	(Bar) Separates two or more items of which you may select only one.
{ }	(Braces) Surround two or more items of which you must select one.

Applicable Documents

For more information, refer to the *Paragon™ System Technical Documentation Guide.*

Comments and Assistance

Intel Supercomputer Systems Division is eager to hear of your experiences with our products. Please call us if you need assistance, have questions, or otherwise want to comment on your Paragon system.

U.S.A./Canada Intel Corporation
Phone: 800-421-2823
Internet: support@ssd.intel.com

Intel Corporation Italia s.p.a.
Milanofiori Palazzo
20090 Assago
Milano
Italy
1678 77203 (toll free)

France Intel Corporation
1 Rue Edison-BP303
78054 St. Quentin-en-Yvelines Cedex
France
0590 8602 (toll free)

Intel Japan K.K.
Supercomputer Systems Division
5-6 Tokodai, Tsukuba City
Ibaraki-Ken 300-26
Japan
0298-47-8904

United Kingdom Intel Corporation (UK) Ltd.
Supercomputer System Division
Pipers Way
Swindon SN3 IRJ
England
0800 212665 (toll free)
(44) 793 491056 (*answered in French*)
(44) 793 431062 (*answered in Italian*)
(44) 793 480874 (*answered in German*)
(44) 793 495108 (*answered in English*)

Germany Intel Semiconductor GmbH
Dornacher Strasse 1
85622 Feldkirchen bei Muenchen
Germany
0130 813741 (toll free)

World Headquarters
Intel Corporation
Supercomputer Systems Division
15201 N.W. Greenbrier Parkway
Beaverton, Oregon 97006
U.S.A.
(503) 629-7600 (Monday through Friday, 8 AM to 5 PM Pacific Time)
Fax: (503) 629-9147

If you have comments about our manuals, please fill out and mail the enclosed Comment Card. You can also send your comments electronically to the following address:

techpubs@ssd.intel.com

Table of Contents

Chapter 5 XProf

Chapter 6 XGprof

Chapter 7 ParaGraph

Chapter 8
The Parallel Make Utility

List of Illustrations

List of Illustrations

List of Illustrations

List of Illustrations

List of Illustrations

List of Tables

ParAide 1

This chapter describes ParAide, the graphical user interface that serves as a launching point to the other graphical tools in the Paragon toolset. These tools include SPV, XIPD, ParaGraph, XProf and XGprof. ParAide also assists you in loading parallel applications and provides a means to open text files into an editor.

ParAide makes it easier for you to bring up other graphical tools and to load programs into the mesh. Dialog boxes for the tools free you from having to learn the command names and arguments. The graphical depiction of the allocated partitions and their mesh locations also provides an added level of familiarity with your Paragon system.

ParAide also provides online, context-sensitive help.

Using ParAide

This section describes how to use ParAide to launch other graphical tools, load programs, and manage files. Detailed information about the individual ParAide dialogs can be found in the *Windows, Menus, and Commands* section of this chapter.

Invoking ParAide

To invoke ParAide, use the **paraide** command as follows:

paraide [-[**no**]**menus**] [-[**no**]**icons**] [-[**no**]**shell**] [**-rows** *minrows*] [**-cols** *mincols*]
[*X Toolkit parameters*]

The **paraide** command parameters are defined as follows:

-[no]menus	[Do not] display the menu bar. **-menus** is the default.
-[no]icons	[Do not] display the icon strip. **-icons** is the default.
-[no]shell	[Do not] display the shell panel. **-shell** is the default.

-rows *minrows*	Sets the minimum number of rows for the shell panel scrolling text area. The number of rows might be greater when the main window is created.
-cols *mincols*	Sets the minimum number of columns for the shell panel scrolling text area. The number of columns might be greater when the main window is created due to the text area being forced to stretch across the window.

X Toolkit parameters

The standard parameters supported by the X Toolkit (refer to the *X Toolkit Intrinsics Programming Manual*).

For complementary options, such as **-menus** and **-nomenus**, the option occurring last on the command line takes precedence if both are included on the command line. You can not specify the options **-nomenus**, **-noicons**, and **-noshell** together on the same command line. ParAide must display at least one of the areas in its main window. ParAide prints an error message and exits if all the **-no** options are used together.

When you invoke ParAide, it displays the main window. Figure 1-1 shows the ParAide main window.

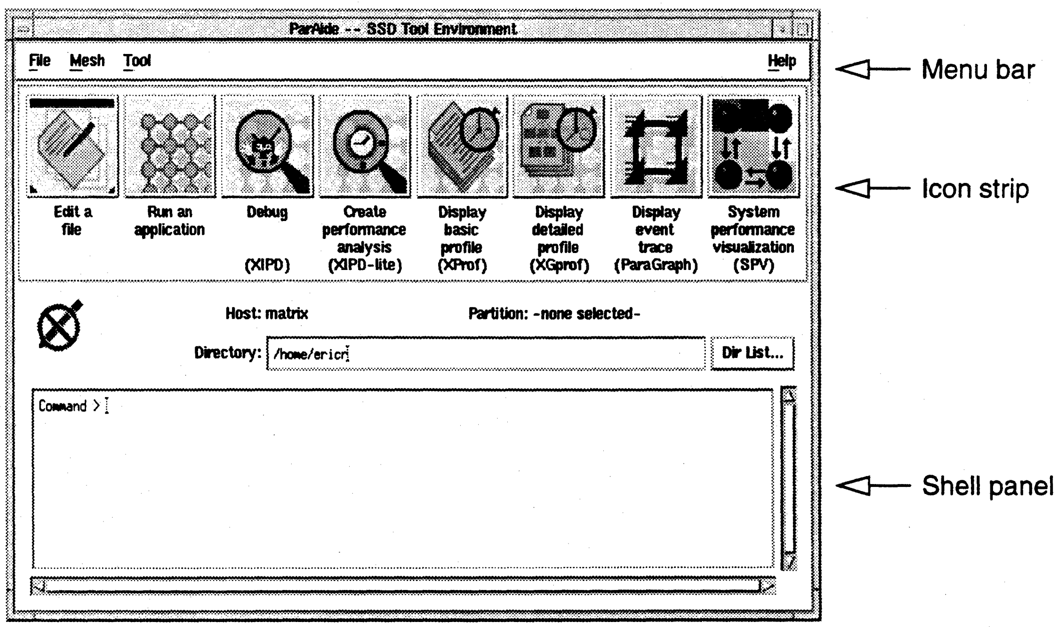

ParAide -- SSD Tool Environment
File
Mesh
Tool
Help
Edit a file
Run an application
Debug (XIPD)
Create performance analysis (XIPD-lite)
Display basic profile (XProf)
Display detailed profile (XGprof)
Display event trace (ParaGraph)
System performance visualization (SPV)
Host: matrix
Partition: -none selected-
Directory:
/home/ericr
Dir List...
Command >
Menu bar
Icon strip
Shell panel

Figure 1-1. ParAide Main Window

The main window is divided into the following regions:

Menu bar	Provides access to the menus that control the features of ParAide. The menu bar includes the *File*, *Mesh*, *Tools*, and *Help* menus.
Icon strip	Allows you to quickly bring up other graphical tools from the Paragon toolset, edit a file, or run an application.
Shell panel	Contains a user shell running on the Paragon system.

Launching Graphical Tools

You can select a graphical tool from the *Tools* menu or from the icon strip. If there are arguments that can be passed to the tool, ParAide displays a dialog box. You can specify arguments in the dialog box and then execute the tool. The graphical tools are SPV, XIPD, ParaGraph, XProf, and XGprof.

Loading Applications

The *Mesh* menu provides dialogs to control loading an application into the mesh. You first select a partition and the nodes into which the application is loaded. You select the partition and nodes from the *Select Partition* dialog. After selecting a partition and nodes, you can use the *Load Program* dialog to load programs into the selected nodes.

ParAide displays the command to execute your application in the shell panel so you can see the command ParAide built from the dialog items you selected. Any terminal output of the program also appears in the shell panel.

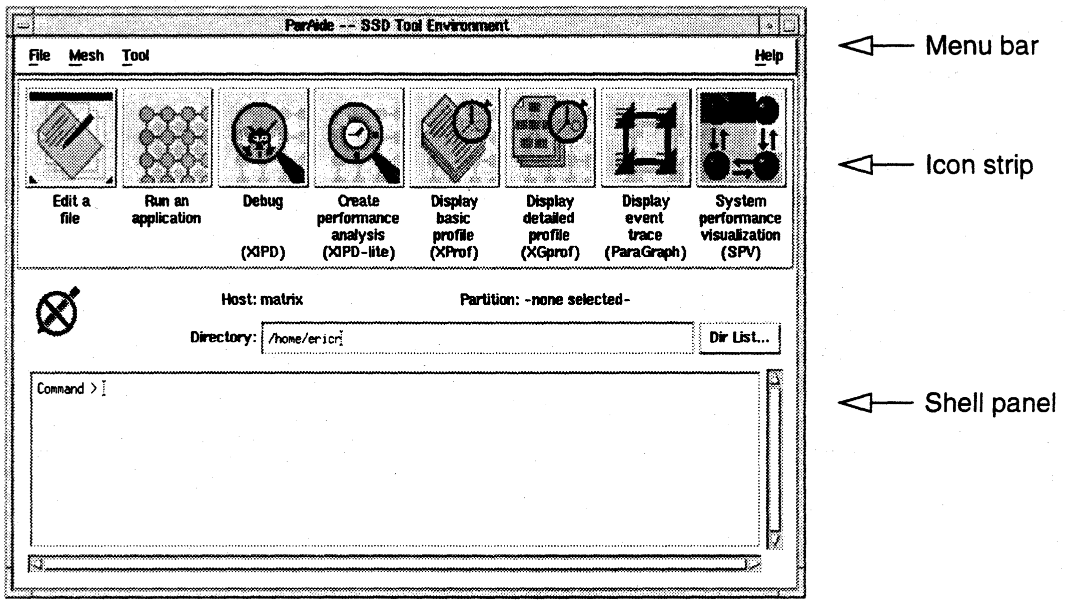
The load command is inserted into what is known as "load history," which is viewable from the load history dialog. From this dialog, you can select a previous load command and have it issued to the shell panel again. This avoids re-using the load dialog to re-execute the same command.

Managing Files

The *File* menu provides dialogs for basic file management. This includes creating a new file and opening an existing file. For new files and opened files, ParAide displays a new command window running the editor specified by your environment.

Windows, Menus, and Commands

The main window of ParAide contains menus, icons, and a command shell area. Figure 1-2 shows the ParAide main window .


ParAide -- SSD Tool Environment
File
Mesh
Tool
Help
Edit a file
Run an application
Debug (XIPD)
Create performance analysis (XIPD-lite)
Display basic profile (XProf)
Display detailed profile (XGprof)
Display event trace (ParaGraph)
System performance visualization (SPV)
Dir List...
Menu bar
Icon strip
Shell panel

Host: matrix

Partition: -none selected-

Directory: /home/ericr

```
Command >
```

Figure 1-2. Main Window

The menu bar across the top of the main window contains the *File*, *Mesh*, *Tool*, and *Help* menus. The icon strip beneath the menu bar contains icons for invoking SPV, XIPD, ParaGraph, XProf, and XGprof. The command shell panel contains the output of a shell executing on the Paragon system. It also contains controls for setting the current directory and enabling you to enter commands into the shell text area.

File Menu

The *File* menu contains dialogs to open files and to exit ParAide. You can pull down the *File* menu by selecting the menu with the pointer or by pressing the menu's mnemonic key **F**.Figure 1-3 shows the *File* menu.

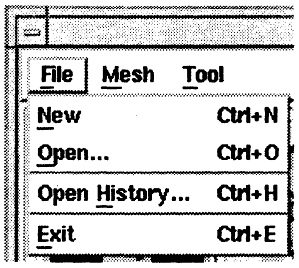


Figure 1-3. File Menu

New

New brings up a new, empty editing program. The default keyboard accelerator for *New* is *Control-N*. When ParAide creates an editor window, it starts an X command shell application that executes the editing program defined by your environment.

Open...

Open... displays the file selection dialog. If you select a file, ParAide creates a new editing program window that contains the selected file. The default keyboard accelerator for *Open...* is *Control-O*.

Open History...

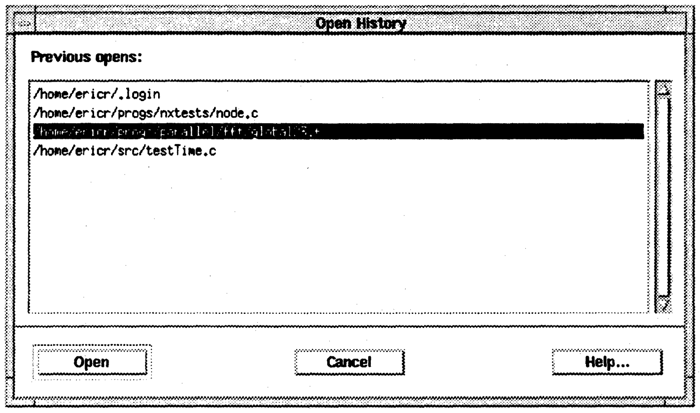
Open History... displays the *Open History* dialog. *Open History...* is enabled after you select a file for editing. The default keyboard accelerator for *Open History...* is *Control-H*.

Exit

Use *Exit* to quit ParAide. ParAide displays a question dialog to ask if you really want to quit. ParAide is exited if you select *Yes*. The default keyboard accelerator for *Exit* is *Control-E*.

Open History Dialog

ParAide displays the *Open History* dialog when you choose the *Open History...* menu item in the *File* menu. This dialog reopens a file opened from a previous selection of the *File* menu's *Open...* menu item. Figure 1-4 shows the *Open History* dialog.



Previous opens:

```
/home/ericr/.login
/home/ericr/progs/nxtests/node.c
/home/ericr/progs/parallel/fft/global[illegible]
/home/ericr/src/testTime.c
```

Figure 1-4. Open History Dialog

Previous opens

This is a list of all unique file names that have previously been opened. You can only select one item at a time. When you select a file name, ParAide enables the *Open* button.

Open

Open creates a new editor window that contains the selected file's contents. The *Open* button is enabled when a file is highlighted in the *Previous opens* list.

Cancel

Cancel dismisses the *Open History* dialog and no file is opened.

Help...

Help... displays help topic text about the *Open History* dialog.

Mesh Menu

The selections in the *Mesh* menu allow you to select a partition and load programs into the mesh. Figure 1-5 shows the *Mesh* menu.

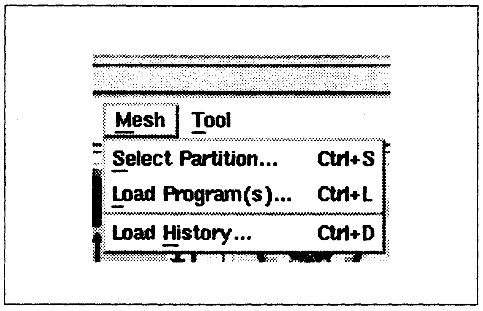


Figure 1-5. Mesh Menu

Select Partition...

Select Partition... displays the *Select Partition* dialog. The default keyboard accelerator for *Select Partition...* is *Control-S.*

Load Program(s)...

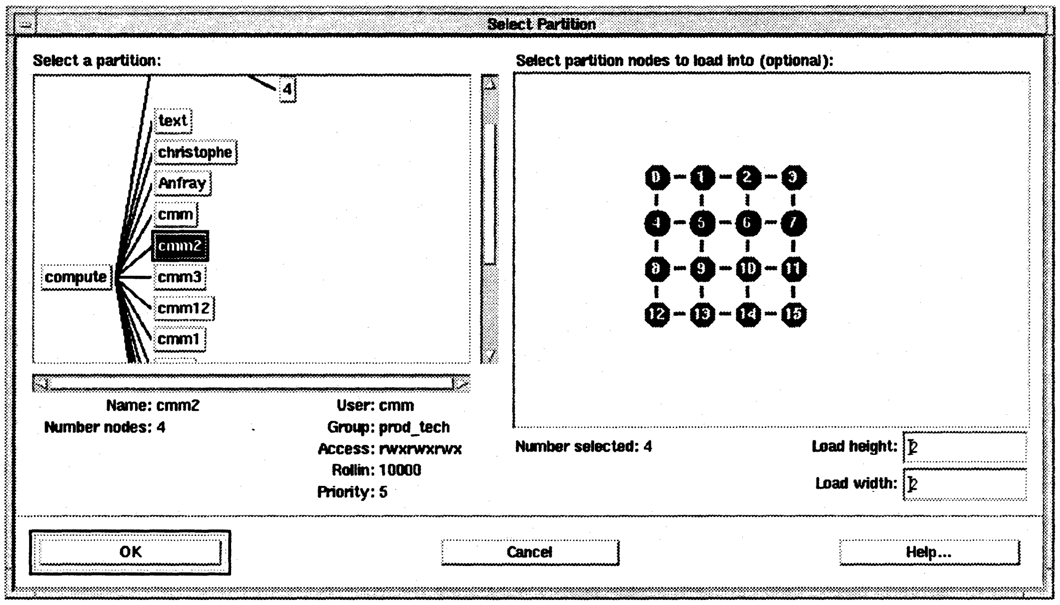
Load Program(s)... display the load program dialog. *Load Program(s)...* is enabled after a partition and nodes have been selected from the *Select Partition* dialog. The default keyboard accelerator for *Load Program(s)...* is *Control-L.*

Load History...

Load History... displays the load history dialog. *Load History...*is enabled after a command to load a program has been issued from the *Load Program(s)...* dialog. The default keyboard accelerator for *Load History...* is *Control-D.*

Select Partition Dialog

Use the *Select Partition* dialog to select a partition to load into and to set the load mesh nodes and size. You must select a partition from the partitions currently allocated on the Paragon system. Figure 1-6 shows the *Select Partition* dialog.



Name: cmm2
Number nodes: 4
User: cmm
Group: prod_tech
Access: rwxrwxrwx
Rollin: 10000
Priority: 5
Number selected: 4
Load height: 2
Load width: 2

Figure 1-6. Select Partition Dialog

Select a partition

This field contains a scrollable tree that represents the hierarchy of the computation partitions in the mesh. A sub-partition is drawn as a child branch off of its parent partition. The current partition selection is inverted in color. When you select a partition, the fields beneath the tree are updated. The fields are described as follows:

Name	Name of the selected partition.
Number nodes	Number of nodes contained within the partition.
User	User name of the partition's creator (and owner).
Group	Group name of the partition's creator.
Access	Access permissions to the partition. This is similar to access permissions to a file. In order to load and execute programs within the mesh, the mesh permissions must be configured to grant you execute permission.

Rollin	The rollin quantum for the partition.
Priority	The partition's maximum priority.

Select partition nodes to load into

This scrollable region represents the physical mesh. When you select a partition from the partition tree, this mesh is updated to reflect where the selected partition resides in the physical mesh. You must select which of the partition's nodes are used for loading a program or a set of programs. Each node has three states:

Not in Partition	The node doesn't belong to the selected partition and cannot be selected for loading a program.
Not Selected	The node belongs to the selected partition but has not been selected for loading.
Selected	The node belongs to the selected partition and has been selected for loading.

The node states are indicated by their coloring. The default coloring for the states is listed in the *Default Configuration* section.

Number selected

This field counts the number of nodes selected from the partition. If you do not select any nodes, ParAide assumes you do not want to load a program.

Load height and Load width

These fields determine the height and width of the mesh containing the nodes selected for loading a program. For example, if you select a sixteen node partition and you choose to load into eight of the partition's nodes, the product of the values in the *Load height* and *Load width* fields must equal eight (for example, one and eight, two and four, four and two, or eight and one). ParAide calculates default values for these fields based on the number of nodes selected. You can use these default values or enter your own values.

OK

When you select *OK*, ParAide checks the mesh height and width against the number of selected partition nodes to be sure they match. If they match, ParAide dismisses the *Select Partition* dialog and displays the partition name in the shell panel. If they do not match, ParAide displays an error dialog and the *Select Partition* dialog remains.

Cancel

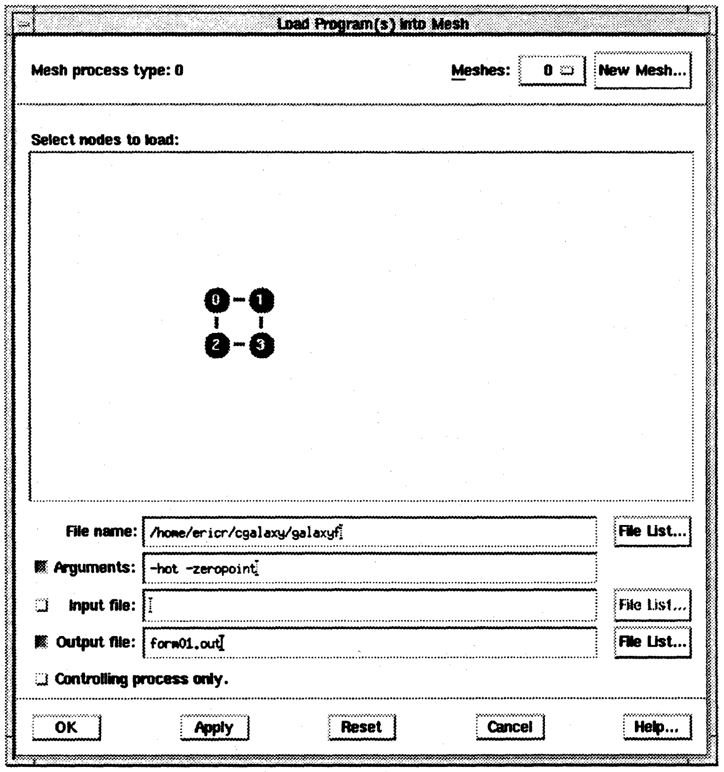
Cancel dismisses the *Select Partition* dialog with no change in the selected partition and nodes.

Help...

Help... displays help topic text about the *Select Partition* dialog.

Load Program Dialog

Use the *Load Program* dialog to load programs into the mesh. Each mesh has a process type number, with the default mesh having process type zero. You can add other process types with the *New Mesh* button. For each process type, ParAide allocates a new mesh. Figure 1-7 shows the *Load Program* dialog.


Load Program(s) into Mesh

Mesh process type: 0

Meshes: 0

Select nodes to load:

File name: /home/ericr/cgalaxy/galaxyf

[✓] Arguments: -hot -zeropoint

[ ] Input file:

[✓] Output file: form01.out

[ ] Controlling process only.

Figure 1-7. Load Program Dialog

Mesh process type

This field contains the process type of the current mesh. When you allocate a new mesh process type or change the display to an existing mesh, this field is updated to the new process type.

New Mesh...

New Mesh... displays a dialog that allows you to add a new process type. Figure 1-8 shows the *New Mesh Process Type* dialog.

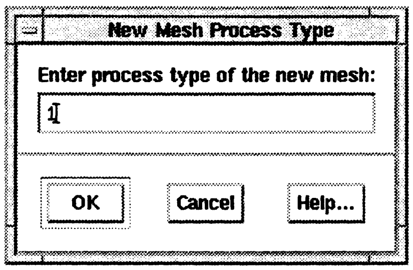


Figure 1-8. New Mesh Process Type Dialog

You can type a new process type into the text field. When you select *OK*, the dialog is dismissed and a new mesh appears in the *Load Program* dialog. The process type is added to the *Meshes* pop-up menu and the text field is updated to the new process type.

If you select *Cancel*, no new process type is allocated.

Selecting *Help...* provides help topic text about allocating a new process type.

Meshes

The *Meshes* pop-up menu contains all of the allocated process types. When you select an entry from this list, the nodes are updated to the corresponding mesh and the *Mesh process type* label displays the selected process type.

Select nodes to load

You can select nodes in the mesh by clicking on them or by holding the mouse button down and dragging the mouse pointer down and to the right, causing a rectangle to form around nodes. Any nodes in the rectangle when you release the mouse button become selected. If you hold down the *Control* key during the rectangle-drag, the nodes within the rectangle "toggle" their state. Unloaded nodes change to *Selected* and selected nodes change to *Unloaded*.

Nodes are colored to represent three different states:

Unloaded	Nothing has been loaded into the node. If you select the node, it changes to *Selected.*
Selected	The node has been selected for loading. The next load operation for the current mesh loads the given program into this node. If you select the node, it changes to *Unloaded.*
Loaded	The node has already been selected and loaded into. Once loaded, you can not select the node again and the node does not respond to any attempt to select it.

File name

This field contains the name of the executable to be loaded into the selected nodes. You can type the executable file name into this field, or select the *File List...* button and find the file with the *Select a File to Open* dialog.

Arguments

If this option is on, the arguments in the adjacent text field are passed as command line arguments to the program being loaded.

Input file

If this option is on, the file name listed in the adjacent text field is used as the standard input for the loaded program.You can type the file name into this field or select the *File List...* button and find the file with the *Select a File to Open* dialog.

Output file

If this option is on, the file name listed in the adjacent text field is used to redirect the standard output stream of the loaded program. You can type the file name into this field or select the *File List...* button and find the file with the *Select a File to Open* dialog.

Standard error is not redirected, because ParAide issues the command to your shell, and the various shells use different mechanisms for sending both standard output and standard error to the same file.

Controlling process only

If this option is on, ParAide does not issue certain command line options to load the program.

By default, ParAide assumes you are loading a program compiled with the **-nx** option, and passes options such as **-pn**, **-on**, and **-pt** as command line arguments to the program. If you are loading a controlling process however, you should select *Controlling process only* to suppress passing these options.

OK

When you select *OK*, ParAide records the specified load in the *Load History* dialog and dismisses the *Load Program* dialog. ParAide displays an error dialog and does not dismiss the *Load Program* dialog if any of the following occur:

- no file name is given
- no nodes are selected and *Controlling process only* is not set
- nodes are selected and *Controlling process only* is set

If there are no errors, the command to run the program is issued to your shell. The command and the output of the command are displayed in the shell panel.

Apply

When you select *Apply*, ParAide notes the load of the specified file into the selected nodes. The selected nodes change to the *Loaded* state. ParAide displays an error dialog if any of the following occur:

- no file name is given
- *Controlling process only* is set
- no nodes are selected and *Controlling process only* is not set

Reset

Reset eliminates all changes since the last *Apply*.

Cancel

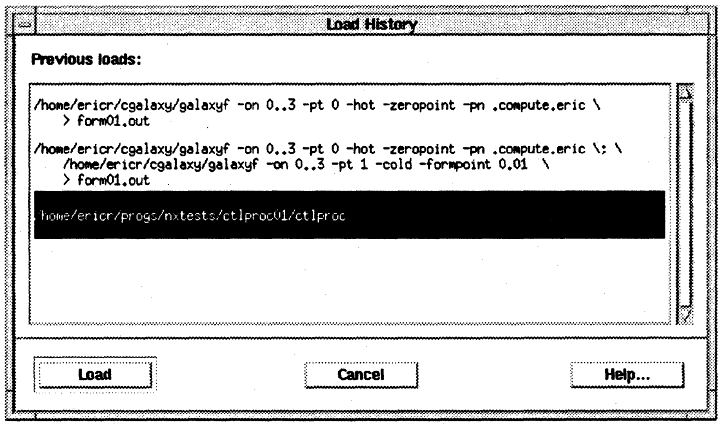
Cancel dismisses the *Load Program* dialog and no program load command is issued.

Help...

Help... displays help topic text about the *Load Program* dialog.

Load History Dialog

The load history dialog contains a list of all the unique load commands that ParAide has issued. To reissue a command, select a single command from the list and then select *Load*. Figure 1-9 shows the *Load History* dialog.



```
/home/ericr/cgalaxy/galaxyf -on 0..3 -pt 0 -hot -zeropoint -pn .compute.eric \
    > form01.out

/home/ericr/cgalaxy/galaxyf -on 0..3 -pt 0 -hot -zeropoint -pn .compute.eric \; \
    /home/ericr/cgalaxy/galaxyf -on 0..3 -pt 1 -cold -formpoint 0.01  \
    > form01.out

/home/ericr/progs/nxtests/ctlproc01/ctlproc
```

Figure 1-9. Load History Dialog

Previous loads list

This field contains a scrolling list that includes all load commands that ParAide has issued. When you select an item from the list, ParAide enables the *Load* button.

Load

When you select *Load*, the highlighted command is issued to the shell panel and ParAide dismisses the *Load History* dialog. ParAide enables the *Load* button when you select an item from the load list.

Cancel

Cancel dismisses the *Load History* dialog

Help...

Help... displays help topic text about the *Load History* dialog.

Tool Menu

The *Tool* menu contains selections to launch other graphical tools, create a new shell, make a program, or create another instance of ParAide. Figure 1-10 shows the *Tool* menu.

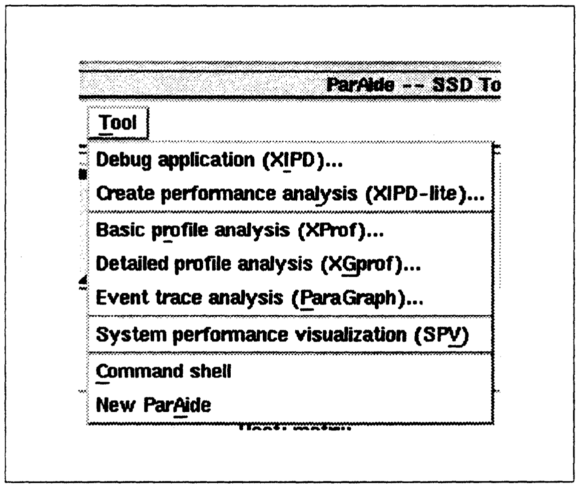

ParAide -- SSD To
Tool
Debug application (XIPD)...
Create performance analysis (XIPD-lite)...
Basic profile analysis (XProf)...
Detailed profile analysis (XGprof)...
Event trace analysis (ParaGraph)...
System performance visualization (SPV)
Command shell
New ParAide

Figure 1-10. Tool Menu

Debug application

This menu item displays the XIPD dialog, with the *performance analysis only* option not chosen.

Create performance analysis

This menu item displays the XIPD dialog, with the *performance analysis only* option chosen.

Basic profile analysis

This menu item displays the *Go to XProf* dialog.

Detailed profile analysis

This menu item displays the *Go to XGprof* dialog.

Event trace analysis

This menu item displays the *Go to ParaGraph* dialog.

System performance visualization

This menu item executes the System Performance Visualization Tool (SPV). For a complete description of SPV, refer to the *Paragon™ System Performance Visualization Tool User's Guide.*

Command Shell

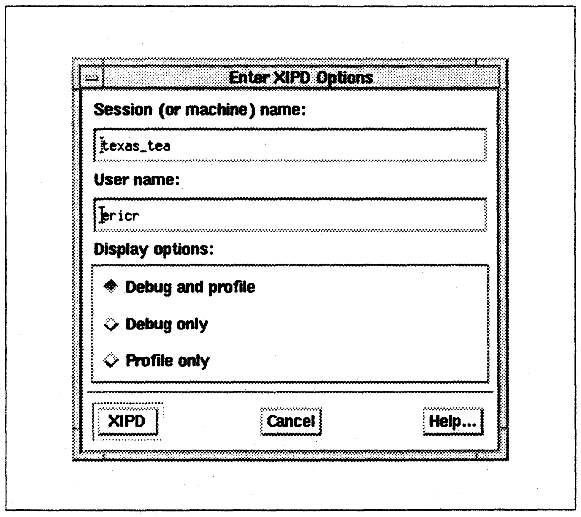
Command Shell creates a new *mxterm* X terminal command window.

New ParAide

New ParAide creates an new instance of ParAide.

XIPD Dialog

Use the *Enter XIPD Options* dialog to invoke XIPD. Figure 1-11 shows the *Enter XIPD Options* dialog.


Enter XIPD Options

Session (or machine) name: texas_tea

User name: ericr

Display options:

[✓ Debug and profile]

Debug only

Profile only

Figure 1-11. Enter XIPD Options Dialog

Session name

This field contains the name of the session file to be restored by XIPD. If you do not have a session file yet, or want to start a new one, this field should contain the name of a Paragon machine. This field defaults to the name of the machine that ParAide is running on.

User name

This field contains the account name XIPD uses to start a new session. This field defaults to the current user's name.

Display options

The display options establish what aspects of XIPD should be displayed. If you do not want to obtain performance information from the debugger, you can inhibit the debugger elements of XIPD by selecting *Profile only* mode. The valid display options are the following:

Debug and profile	Enables both debugging and performance profiling interface elements.
Debug only	Enables only debugging interface elements.
Profile only	Enables only profiling interface elements.

XIPD

XIPD executes XIPD with the specified options.

Cancel

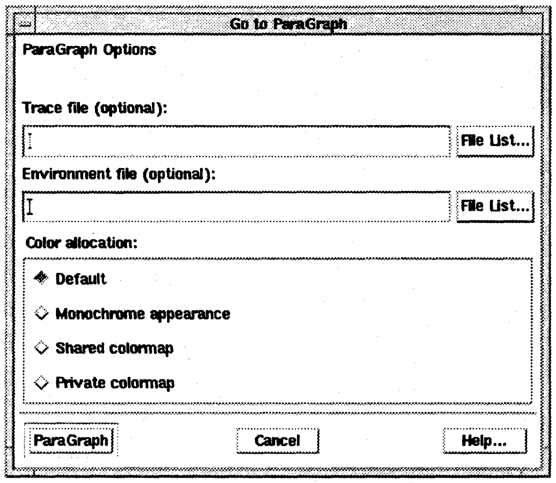
Cancel dismisses the *Enter XIPD Options* dialog and XIPD is not executed.

Help...

Help... displays help topic text about the *Enter XIPD Options* dialog.

Go to ParaGraph Dialog

Use the *Go to ParaGraph* dialog to invoke ParaGraph. Figure 1-12 shows the *Go to ParaGraph* dialog.


Go to ParaGraph

ParaGraph Options

Trace file (optional):

File List...

Environment file (optional):

File List...

Color allocation:

◆ Default

◇ Monochrome appearance

◇ Shared colormap

◇ Private colormap

ParaGraph | Cancel | Help...

Figure 1-12. Go to ParaGraph Dialog

Trace File

The *Trace file* field contains the name of a ParaGraph trace format file that ParaGraph opens as part of its initialization. You can type a file name into this field, or select the *File List...* button and select the file with the *Select a File to Open* dialog. You can also leave the field blank.

Environment file

The *Environment file* field restores a file that contains the layout of ParaGraph's report windows. You can type a file name into this field, or select the *File List...* button and select the file with the *Select a File to Open* dialog. You can also leave the field blank.

Color allocation

Color allocation allows advanced control over the ParaGraph colormap allocation and should, in general, be left on *Default*. If ParaGraph has problems allocating enough color cells, you should select *Private colormap* to correct the problem.

The settings are described as follows:

Default	ParaGraph executes without any specific color allocation directives.
Monochrome appearance	ParaGraph displays in monochrome (black and white).
Shared colormap	ParaGraph executes with the shared colormap. With this setting ParaGraph may not be able to allocate enough colors for certain functions.
Private colormap	ParaGraph executes with its own private colormap. While ParaGraph has enough colors with this setting, there will be a colormap "flash" when you move between ParaGraph and other X applications on the screen.

ParaGraph

ParaGraph executes ParaGraph with the specified dialog settings.

Cancel

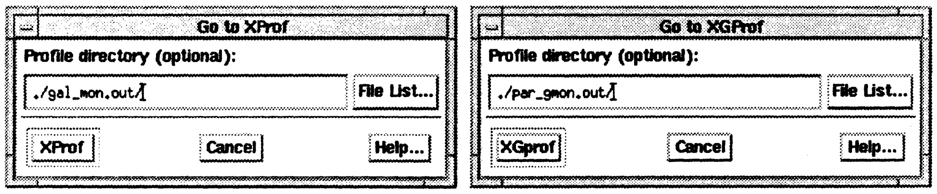
Cancel dismisses the *Go to ParaGraph* dialog and ParaGraph is not executed.

Help...

Help... displays help topic text about ParaGraph.

XProf and XGprof Dialogs

You use the *Go to XProf* and *Go to XGprof* dialogs to execute XProf and XGprof. Figure 1-13 shows the *Go to XProf* and *Go to XGprof* dialogs.



Profile directory (optional): ./gal_mon.out

Profile directory (optional): ./par_gmon.out

Figure 1-13. Go to XProf and Go to XGprof Dialogs

Profile directory

This field contains the name of the performance directory given to XProf or XGprof to open during its initialization. You can type a file name into this field, or select the *File List...* button and select the file with the *Select a File to Open* dialog. You can also leave the field blank.

XProf

XProf executes XProf and dismisses the *Go to XProf* dialog.

XGprof

XGprof executes XGprof and dismisses the *Go to XGprof* dialog.

Cancel

Cancel dismisses the dialog and the tool is not executed.

Help...

Help... displays help topic text about the dialog.

Help Menu

The *Help* menu, available from the main window, provides various levels of help. This menu is always enabled. Through the *Help* menu, you can obtain context-sensitive help and browse through all of the help topics. Figure 1-14 shows the *Help* menu.

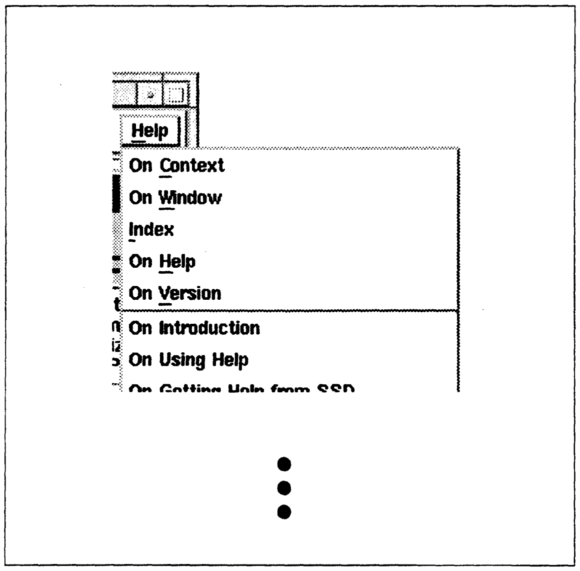


Figure 1-14. Help Menu

On Context

The *On Context* menu item enables context-sensitive help and changes the mouse pointer to a question mark. You can then click on areas of the ParAide interface. If there is a help entry for the selected area (such as the list area of the main window), ParAide displays the help topic text for that feature. You can also obtain context help from the keyboard by pressing the *Help* key (typically mapped to F1) while the keyboard focus is directed to the desired interface feature.

On Window

Displays the help topic text for the main window.

Index

Displays the *Index* dialog. All help topics for ParAide are listed in the *Index* dialog.

On Help

Displays the help topic text that explains how to use all of the aspects of help.

On Version

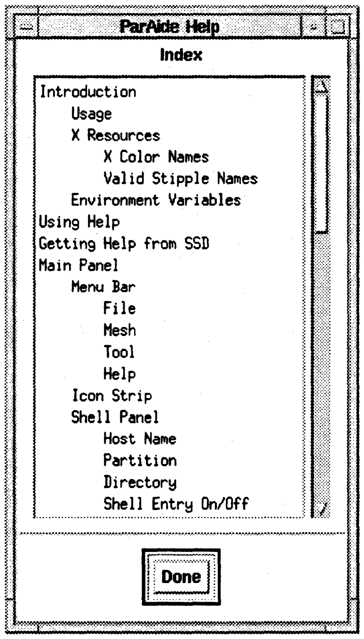
Displays a dialog containing ParAide version information.

Additional Topics

The names of all the major ParAide help topics follow the *On Version* entry on the *Help* menu. When you select a topic, ParAide displays help text for the selected topic.

Help Index

ParAide displays the *Index* dialog when you choose the *Index* menu item from the *Help* menu. The help index contains all the topics and sub-topics of help available for ParAide. Sub-topics are indented underneath major topics. Figure 1-15 shows the *Index* dialog.



```
Introduction
    Usage
    X Resources
        X Color Names
        Valid Stipple Names
    Environment Variables
Using Help
Getting Help from SSD
Main Panel
    Menu Bar
        File
        Mesh
        Tool
        Help
    Icon Strip
    Shell Panel
        Host Name
        Partition
        Directory
        Shell Entry On/Off
```

Figure 1-15. Help Index Dialog

Help Topics

This field contains a scrollable list of all ParAide help topics. Select a topic from the list to display the help text for that topic.

Done

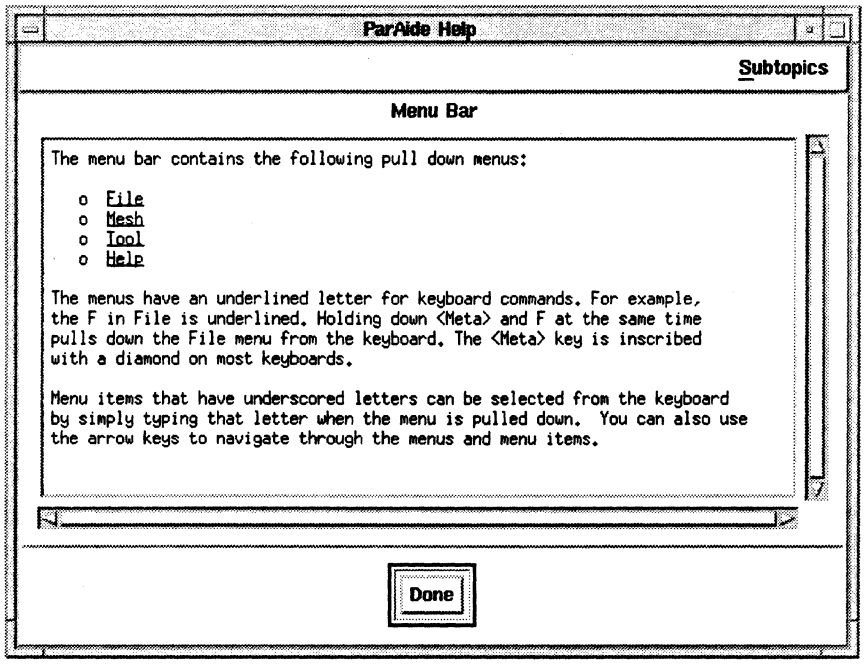
Select *Done* to dismiss the help index dialog.

Help Topic

The help topic dialog appears when you request help for a particular topic. You can select a topic by any of the following:

- selecting the topic in the help *Index* dialog
- selecting a dialog's *Help* button
- using context sensitive help
- selecting a major topic from the *Help* menu.

Figure 1-16 shows a help topic dialog.



```
The menu bar contains the following pull down menus:

   o  File
   o  Mesh
   o  Tool
   o  Help

The menus have an underlined letter for keyboard commands. For example,
the F in File is underlined. Holding down <Meta> and F at the same time
pulls down the File menu from the keyboard. The <Meta> key is inscribed
with a diamond on most keyboards.

Menu items that have underscored letters can be selected from the keyboard
by simply typing that letter when the menu is pulled down.  You can also use
the arrow keys to navigate through the menus and menu items.
```

Figure 1-16. Help Topic Dialog

Subtopics

If a topic has subtopics, the help topic dialog contains a *Subtopics* pull-down menu. All of the subtopics are included on the menu. Select an item from the *Subtopics* menu to display help text for the subtopic.

Help Title

The title identifies which topic the help text describes.

Help Text

This field contains the help text for the topic. You can select the text and paste it into other clients.

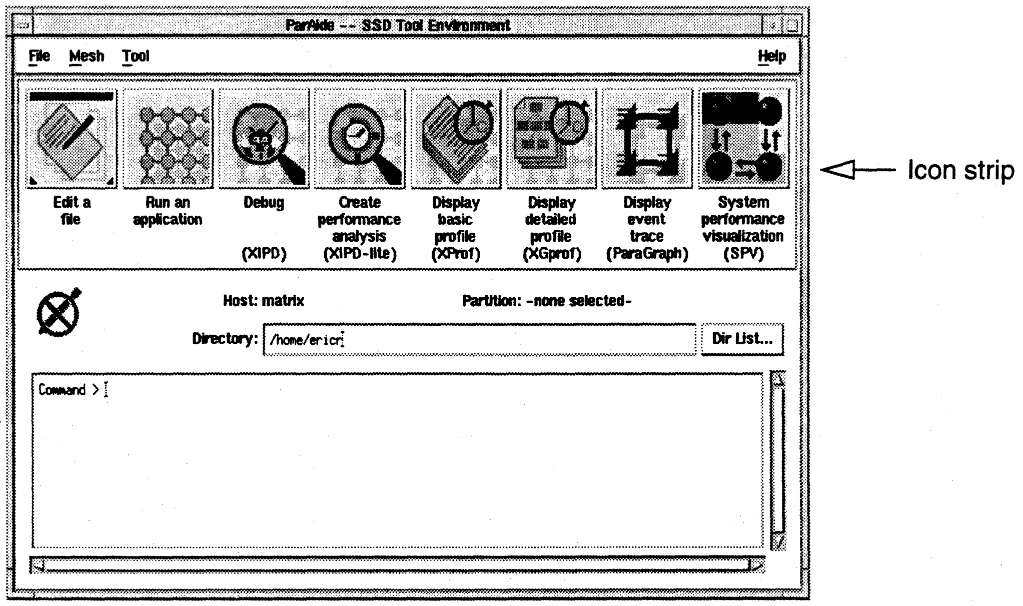
Items that are underlined in the text represent links to other help topics. Clicking on an underlined text entry displays the help topic dialog for that entry.

Done

Use the *Done* button to dismiss the help topic dialog.

Icon Strip

The ParAide icon strip is located beneath the menu bar in the main window. Figure 1-17 shows the icon strip.


ParAide -- SSD Tool Environment
File
Mesh
Tool
Help
Edit a file
Run an application
Debug (XIPD)
Create performance analysis (XIPD-lite)
Display basic profile (XProf)
Display detailed profile (XGprof)
Display event trace (ParaGraph)
System performance visualization (SPV)
Icon strip
Dir List...

Host: matrix

Partition: -none selected-

Directory: /home/ericr

```
Command >
```

Figure 1-17. Icon Strip

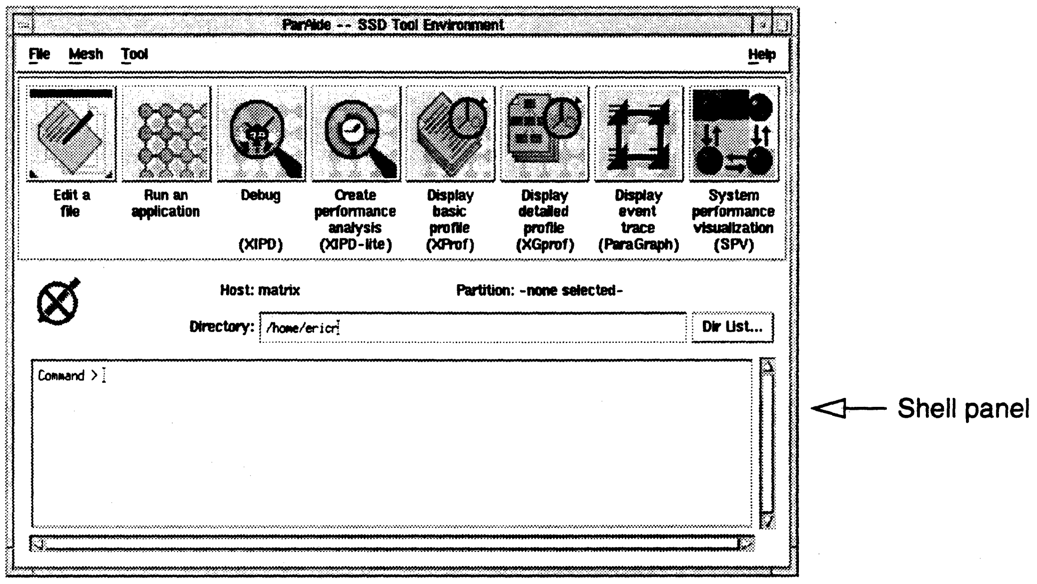
Icons are present for each graphical tool in the Paragon toolset. This includes SPV, XIPD, ParaGraph, XProf and XGprof. You select the icon of the desired tool to launch the tool. In the case of SPV, it is executed directly without a dialog. For the other tools, ParAide displays the corresponding dialog. Refer to the *Tool Menu* section of this chapter for information on the tool dialogs.

You can also initiate a file editing session from the icon strip. If you select the *Edit a file* icon, ParAide displays the file selection dialog. When you select a file, a new editor window for the file is displayed.

When you choose the *Run an application* icon, ParAide displays the load dialog and the partition selection dialog (if necessary).

Shell Panel

The ParAide shell panel is located beneath the icon strip in the main window. Figure 1-18 shows the shell panel.


ParAide -- SSD Tool Environment
File Mesh Tool
Help
Edit a file
Run an application
Debug (XIPD)
Create performance analysis (XIPD-lite)
Display basic profile (XProf)
Display detailed profile (XGprof)
Display event trace (ParaGraph)
System performance visualization (SPV)
Dir List...
Shell panel

Host: matrix Partition: -none selected-

Directory: /home/ericr

```
Command >
```

Figure 1-18. Shell Panel

You can use the shell panel to change directories, indicate the current host name and partition name, and view load commands. You can also use the shell area to enter shell commands if the shell entry protection is turned off.

The shell program run by ParAide is determined by the *SHELL* environment variable. If *SHELL* is not defined, ParAide looks up your account configuration and uses the shell defined by the account entry.

Shell entry allowed icon

By default, the command shell running in the shell panel is for viewing only. You can not type commands into the shell area. If you want to enter commands to the shell, you must remove the "Do Not" circle around the pencil icon in the upper left area of the shell panel. This icon represents whether or not you can make shell entries. You toggle the state of the icon by selecting it. Figure 1-19 shows the two different states of the shell entry icon.

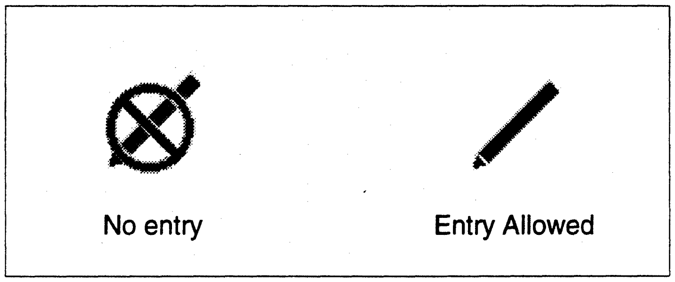


Figure 1-19. Shell Entry Icon

Host

This label identifies the Paragon system on which ParAide is running.

Partition

This label lists the name of the partition you selected. If you have not selected a partition, the label shows the following:

```
   --none selected--.
```

Directory

This is a text field containing the name of the current directory. You can move to a new directory by typing in the directory name and pressing *Return* or by selecting the *Dir List...* button and choosing a directory with the *Select a File to Open* dialog.

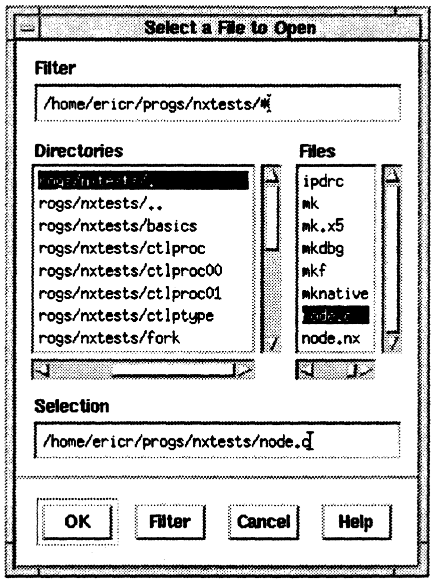
Command Shell

This is a scrolling text region containing the output of your shell running on the Paragon system. This is not a login shell, so you can only use a limited number of interactive commands. You can not use anything that requires screen-management. Initially, this text region is view-only and can not be edited. You can select the contents and paste them into other X clients.

In order to edit in the shell and enter commands, the shell entry icon must be in the correct state.

Selecting Files and Directories

The *Select a File to Open* dialog is displayed when you select the *Open...* menu item or a *File List...* or *Dir List...* push button. You can use this dialog to select both files and directories. Figure 1-20 shows the *Select a File to Open* dialog.



Filter: /home/ericr/progs/nxtests/*

Directories:

```
rogs/nxtests/.
rogs/nxtests/..
rogs/nxtests/basics
rogs/nxtests/ctlproc
rogs/nxtests/ctlproc00
rogs/nxtests/ctlproc01
rogs/nxtests/ctlptype
rogs/nxtests/fork
```

Files:

```
ipdrc
mk
mk.x5
mkdbg
mkf
mknative
node.c
node.nx
```

Selection: /home/ericr/progs/nxtests/node.c

Figure 1-20. Select a File to Open Dialog

Filter

This text field contains the filter for displaying files. All directories are displayed, but you can filter out files to control the length of the list. You can use standard UNIX shell expressions in the filter field. For example, to display all C-language source code files in a directory, you could enter the following in the filter field:

*/home/username/src/**.c

The *.c expression causes ParAide to match all files that end in .c. Therefore, all files that end with the extension *.c* are displayed.

If you want to list all the files in a directory, make sure the filter ends with an asterisk.

Directories

All directories in the current directory (specified by the filter) are listed in this field. Selecting a directory makes it the current directory. Directories are never filtered out.

Files

All files in the current directory that pass through the filter are listed in this field. It is possible for this list to be empty (represented by a [ ] in the file list) if no files match the filter, or if the current directory is empty. Make sure the filter ends with an asterisk if you want to list all the files.

If you select a file, the file is listed in the file selection dialog and ParAide automatically invokes *OK*, dismissing the dialog.

Selection

The *Selection* field contains the current file selection. If you select *OK*, this file is returned by the *Select a File to Open* dialog to the dialog from which it was displayed. For example, if the *Select a File to Open* dialog was displayed from the *Load Program* dialog, pressing *OK* copies the name of the file in the *Selection* field to the *Load Program* dialog's text field as the name of the executable to load.

Dialog Buttons

OK	Passes the entry in *Selection* to the originating dialog and dismisses the *Select a File to Open* dialog.
Filter	Obtains a new file list, and possibly a new directory list.
Cancel	Dismisses the *Select a File to Open* dialog and passes nothing to the originating dialog.
Help	Displays help topic text about the *Select a File to Open* dialog.

Configuring ParAide

You can configure ParAide by using an X resource file. You can make the resource entries in your *.Xdefaults* file (which resides in your home directory) or in a file named *Paraide* (located either in your home directory or in a directory specified by the *XAPPLRESDIR* environment variable). The entries in the *.Xdefaults* file take precedence over the *Paraide* file entries.

Along with the resources corresponding to the standard X toolkit command line options, the following ParAide application resources are provided to configure ParAide:

Paraide.showMenus	A boolean (True / False) value that controls creation of the menu bar for the ParAide main window
Paraide.showIconStrip	A boolean (True / False) value that controls creation of the icon strip for the ParAide main window.
Paraide.showShell	A boolean (True / False) value that controls creation of the shell panel for the ParAide main window
Paraide.showIconLabels	A boolean (True / False) value that controls the display of description labels beneath icons.
Paraide.shellRows	An integer that defines the minimum number of rows the shell panel scrolling text region should have.
Paraide.shellColumns	An integer that defines the minimum number of columns the shell panel scrolling text region should have.
Paraide.nodeSize	An integer that defines the size (in pixels) of the nodes drawn on the mesh.
Paraide.nonPartitionNodeColor	A color name that defines the color for mesh nodes that don't belong to the currently selected partition. Also used to color nodes that haven't been loaded.
Paraide.partitionNodeColor	A color name that defines the color for mesh nodes that belong to the currently selected partition. Also used to color nodes that have been selected for loading.
Paraide.partitionSelectedNodeColor	A color name that defines the color for mesh nodes that have been selected to be loaded into. Also used to color nodes loaded with a program.

Paraide.nonPartitionNodeStipple
A stipple pattern name that defines the stipple to be used with nodes that don't belong to the currently selected partition. Also used to render nodes that haven't been loaded.

Paraide.partitionNodeStipple A stipple pattern name that defines the stipple to be used with mesh nodes that belong to the currently selected partition. Also used to render nodes that have been selected for loading

Paraide.partitionSelectedNodeStipple
A stipple pattern name that defines the stipple to be used with mesh nodes that have been selected to be loaded into. Also used to render nodes loaded with a program.

You can use an optional stippling pattern for drawing the nodes. If you use this pattern, it must be one of the following strings (or else ParAide ignores the resource setting):

dotsVeryDark	Almost all the pixels drawn.
dotsDark	Most of the pixels drawn.
dotsNorm	Half the pixels drawn.
dotsLight	Some of the pixels drawn.
dotsVeryLight	Very few of the pixels drawn.
lineDiagHatch	Hatched diagonal lines.
lineDiagLeft	Diagonal lines slanted to the left.
lineDiagRight	Diagonal lines slanted to the right.
lineHatch	Hatched horizontal and vertical lines.
lneHoriz	Horizontal lines.
lineVert	Vertical lines.

Default Configuration

The following are the ParAide default resource settings:

Paraide.showMenus	True
Paraide.showIconStrip	True
Paraide.showShell	True
Paraide.shellRows	10
Paraide.shellColumns	80
Paraide.nodeSize	28
Paraide.nonPartitionNodeColor	#acacac (color) #cccccc (grayscale) black (monochrome)
Paraide.partitionNodeColor	#00daa00 (color) #e0e0e0 (grayscale) black (monochrome)
Paraide.partitionSelectedNodeColor	#40e9f8 (color) #ffffff (grayscale) black (monochrome)
Paraide.foregroundColor	white (color) black (grayscale) black (monochrome)
Paraide.backgroundColor	#2f689e (color) #b5b5b5 (grayscale) white (monochrome)
Paraide.nonPartitionNodeStipple	*Null-String* (color/grayscale) black (monochrome)
Paraide.partitionNodeStipple	*Null-String* (color/grayscale) "dotsDark" (monochrome)
Paraide.partitionSelectedNodeStipple	*Null-String* (color/grayscale) "solid" (monochrome)

Null-String means that the stipple resource is not defined, and by default, an empty, null string is used to indicate that no pattern should be used to stipple the node's status.

Environment Variables

The following environment variables are used directly by ParAide:

EDITOR	Name of the editor program to execute when a file is to be edited (from *New*, *Open...*, or from the open history dialog). If not defined, ParAide executes the */usr/bin/vi* program.
TGI_EDITOR	Name of the editor program that overrides the contents of *EDITOR*. This is useful for defining an editor to be used specifically with ParAide.
NX_DFLT_PART	Name of the default partition that ParAide selects within the partition selection dialog. If not defined, ParAide defaults to selecting the *.compute* partition.
SHELL	The shell program to execute within the shell panel.

Note

ParAide creates a new X terminal window that executes the editor program unless the program's name begins with x, in which case ParAide assumes the editor is X-based and doesn't need a terminal window.

Interactive Parallel Debugger 2

The Interactive Parallel Debugger (IPD) is a complete symbolic, source-level debugger for parallel programs that run under the Paragon™ OSF/1 operating system. Beyond the standard operations that facilitate the debugging of serial programs, IPD offers custom features that ease the task of debugging parallel programs.

Through a command-line interface, which includes on-line help, you can examine and modify running processes. Among the features specifically designed to aid debugging in a parallel environment are facilities to help debug message-passing, and the ability to set a command context to apply commands to multiple processes running on multiple nodes. With these facilities, you can set breakpoints in selected processes, monitor the queues of messages passing among processors, and display stack tracebacks and the values of registers or variables.

IPD lets you debug parallel programs written in the following programming languages:

- C
- Fortran
- i860™ assembly language

The IPD command and display syntax for variables follows the language convention of the program being debugged.

This chapter describes the features of IPD and provides some guidelines for using IPD. For a complete description of the IPD commands, refer to the *Paragon™ Interactive Parallel Debugger Reference Manual.*

Compiling for Debugging

To compile for debugging, you should use the **-g** compiler switch. The **-g** switch is equivalent to -Mdebug -Mframe -O0. These switches have the following effects:

-Mdebug	Generate symbol and line number information.
-Mframe	Generate stack frames on function calls. (Default **-Mnoframe**.) Debugging code without stack frames generated on function calls will result in stack tracebacks that have missing calls when you use the **frame** command.
-O0	Optimization off. If you do not turn optimization off, access to individual source lines will be decreased, and display or modification of variables and registers will probably have unpredictable results.

You can debug programs not compiled for debugging, but your ability to debug will be very limited.

When you are debugging code compiled with the **-nx** or **-lnx** switches, you can debug the code running on the nodes, or by setting the context to **(host)**, you can debug the controlling process in the service partition. If you compile without either of these switches, you are running in the service partition, and the debug context is automatically set correctly.

Invoking IPD

To invoke IPD, enter the **ipd** command as follows:

ipd

You can cause IPD to execute a set of commands automatically when you invoke it if you put the desired commands in a file named *.ipdrc* in your home directory. The *.ipdrc* file is often used to define configuration information, such as a list of convenient aliases and command line variables.

NOTE

You cannot specify a partition in which to debug programs on the **ipd** command line. You must use the IPD **load** command to specify a partition for debugging. For a complete description of the **load** command, refer to the *Paragon^{TM} Interactive Parallel Debugger Reference Manual.*

IPD Commands

The IPD commands fall generally into three categories: execution control, program display, and debug environment. Table 2-1, Table 2-2, and Table 2-3 list the IPD commands associated with these functions. You can abbreviate any command, keyword, or switch to the minimum number of characters required to uniquely identify it. For example, for the **process** command, all of these abbreviations are valid: **proces**, **proce**, **proc**, **pro**, **pr** or **p**. If the command abbreviation is ambiguous, IPD displays an error message and asks you to retype the command.The tables also show the minimum abbreviation for each command.

When you issue an IPD command, IPD first searches the IPD alias list before it matches a command to the IPD command table. You can alias any command to one or more characters for your convenience. Your *.ipdrc* file in your home directory can contain a set of **alias** commands that define convenient aliases for those commands that you use most during a debug session. These definitions are automatically included whenever you invoke IPD.

For a complete description of the IPD commands, refer to the *Paragon™ Interactive Parallel Debugger Reference Manual.*

Table 2-1. Execution Control Commands

Command	Minimum Abbreviation	Description
break	**b**	Set and display breakpoints
continue	**conti**	Continue processes stopped by command or by a breakpoint
flush	**fl**	Set flush policy for event trace buffers.
instrument	**i**	Add, remove, or display program instrumentation for performance data collection
kill	**k**	Terminate processes
remove	**rem**	Remove breakpoints
rerun	**rer**	Restart the application without reusing command line arguments
run	**ru**	Restart the application, reusing any previous command line arguments
step	**ste**	Execute the next source statement or instruction
stop	**sto**	Stop execution of processes
trace	**tr**	Set or display tracepoints
wait	**wai**	Wait until processes stop running
watch	**wat**	Set or display watchpoints

Table 2-2. Program Examination and Modification Commands

Command	Abbreviation	Description
assign	**as**	Assign a new value to a program variable, register, or memory location
disassemble	**disa**	Display assembler listing of i860 node program code
display	**disp**	Display the value of a program variable, register, or memory location
frame	**fr**	Display the runtime activation stack
list	**li**	List source code of loaded program
msgqueue	**ms**	Display messages sent but not yet received
recvqueue	**rec**	Display posted receives not yet satisfied
process	**p**	Display current state of processes
type	**ty**	Display type of variable

Table 2-3. Debug Environment Commands

Command	Abbreviation	Description
alias	**al**	Set or display command aliases
unalias	**una**	Delete command aliases
context	**conte**	Set the current node and process context
quit or **exit**	**q**	Exit IPD
exec	**exe**	Read in and execute a command file
source	**so**	Set or display the source directory search path list
help or **?**	**h**	Display IPD commands and syntax
load	**loa**	Load node programs
log	**log**	Record the debug session
more	**mo**	Turn terminal scrolling on or off
set	**se**	Set or display command line variables
status	**sta**	Display current IPD status
unset	**uns**	Delete command line variables
system or **!**	**sy**	Execute a Paragon OSF/1 command

The only commands you can issue prior to the **load** command are those in Table 2-3, with the exception of the **context** command. All other IPD commands must act on a process, so a debug context must exist. You cannot form a context until you have loaded a program.

Syntax of IPD Commands

IPD command lines have the following general form (where *full_command* denotes an IPD command and all appropriate arguments):

full_command[; *full_command*;] ... [#*comment*]

full_command The form of a *full_command* can be one of the following:

command arguments

command -*switch arguments*

command (*context*) -*switch arguments*

command	One of the IPD commands
arguments	Command arguments specific to each command. If the command accepts a number of arguments then the arguments must be separated by spaces. The order of command line arguments depends upon the command. For example, the order of the arguments for **assign** is significant, but not for **remove**. Refer to each command description to determine if the command line argument order is important.
-*switch*	A command option shown in boldface and preceded by a dash is a command line switch. Whether a switch has a following argument depends upon the command. Command line switches with no following argument can appear anywhere on the command line after the command name. Switches with a following argument are usually position-dependent. You should refer to each command description to determine if the command line keyword and argument order is important.
(*context*)	The context argument is always defined within parentheses. The context argument defines the set of processes and nodes that are the target of the IPD command (see the **context** command). The context argument must appear immediately after the command and before all other arguments.

;	The semicolon is a command separator. Multiple commands may appear on the same command line separated by a semicolon. The exceptions to this rule are the **alias**, **set**, **system**, **load**, **run**, and **rerun** commands and comments.

# *comments*	A comment can be entered either at the end of a command line, starting with a pound sign (#) followed by a space, or on a line by itself, indicated by a pound sign (#) as the first character of a command line. All following characters to the end of the line, are considered comment characters and are not interpreted by IPD. This includes semicolons.

To specify an address or value in a number base other than decimal, it must have a leading zero, followed by the first letter of the base. In octal, it must have a leading *0o*. A hexadecimal value must have a leading *0x*. The leading zero is required.

For all IPD commands, a *filename* argument refers to a Paragon OSF/1 pathname where the tilde (~) character denotes your home directory. IPD only substitutes your environment variable *$HOME* for the tilde; IPD does not expand *~user* names.

Using IPD

The following sections describe how to use IPD to debug your applications.

Context, Execution Point, and Scope

To use IPD, you need to understand *debug context*, *execution point*, and *scope*. The context defines the nodes and processes under debug — those to which the IPD commands refer. The execution point is the point in a process just before the next statement to be executed. Each process has its own execution point. The scope of a variable is within those parts of a program where it is recognized and accessible. The execution point determines what variables are in scope and what file a line number refers to.

The context determines the nodes and the processes on those nodes that an IPD command affects. When you enter IPD and execute the **load** command, you use the same syntax for loading your program that you use from the shell. IPD loads the program and sets the initial default context. If the program specifies partitions and nodes internally, it works as if it were loaded from the shell. If you do not specify a context for the commands whose syntax allows you to specify a context, IPD uses the default context; the context used by the command (either default or specified) is referred to as the *current context*. You can change the default context with the **context** command. The default context is shown as the IPD prompt.

The following example shows using the context command to set the context to process type 0 on all compute nodes.

```
ipd> context (all:0)
(all:0) >
```

The following example sets the context to the controlling (host) process.

```
ipd> context (host)
(host) >
```

The following example shows using the context command to display the default context.

```
(all:0)> context
                 Current   Previous
Processors       Ptype     Ptypes        Program
==========       =======   ========      =======
(all)                  0                 gauss
```

The Paragon OSF/1 operating system allows you to change your program's process type (ptype) with a call to **setptype()**. After a call to **setptype()**, the process has a new process type, but still owns the old process type for the duration of the application's execution (even if the process is gone, the process type is not reusable). In this case, IPD interprets the old and new process types as alternate names for the same process, so the call to **setptype()** does not invalidate the default context.

Some of the IPD commands require the context to include only processes running the same object module. A *load module* is an executable object module that you have loaded onto the system with the IPD **load** command. These commands are **assign**, **break**, **disassemble**, **display**, **instrument**, **list**, **trace**, **type**, and **watch**.

IPD gives you several ways to determine the current scope and context, such as the display of the current context as the prompt, and the **context**, **frame**, and **process** commands. While you have access to any point in the program(s) that you have loaded using IPD, if the current execution point is not within the routine or program to which you want access, you need to prefix the variable name with the routine name and/or the file name on the command line. Likewise, you need to set the context either with the **context** command or within a given command to make sure that the command applies to the nodes and/or processes that you want it to.

Consider the following example. The **frame** command displays procedures that have been activated as you execute the program. For this program, the **frame** command tells you that nodes 0 through 2 are blocked in the **flick**() system call called from the **gdhigh**() system call; node 3 is waiting in a different routine, the **msgwait**() system call in the *shadow* routine:

```
(all:0) > frame
  ***** (0..2:0) *****
    _flick()     [_flick.c{}0x00023fe8]
    _gdhigh()     [_gdhigh.c{}0x000240f8]
    gdhigh_()     [gdhigh_.c{}0x0001e9dc]
    gauss()     [gauss.f{}#72]
    main()     [pgfmain.c{}0x000001ac]
  ***** (3:0) *****
    _flick()     [_flick.c{}0x00023fe8]
    _msgwait()     [nxlib.c{}0x0002011c]
    shadow()     [gauss.f{}#209]
    gauss()     [gauss.f{}#58]
    main()     [pgfmain.c{}0x000001ac]
```

If multiple processes in the current context would result in identical display of information, the information is displayed only once, preceded by a line displaying the context to which the information applies. In the previous example, nodes 0 through 2 were doing the same thing; node 3 has a separate display because the information is different.

The following command line asks for the display of the value of the variable *iam*, which is in the *shadow* routine. You need to make sure the scope is correct:

```
(all:0) > disp iam
 *** ERROR: search failed
 ***      Not found: iam
```

The error message indicates that the variable is not in the current scope; while the **frame** command showed that the nodes executing the program stopped in the **flick**() routine, the variable you are looking for is in the *shadow* routine. The following is displayed if you qualify the variable name with the name of the routine:

```
(all:0) > disp shadow()iam
 ** gauss.f{}shadow()iam **

 *** ERROR: cannot read memory
 ***       error: (all:0) Stack frame for procedure not found
```

This failure is due to an incorrect context, so you need to override the default context; in this case, node 3 is the only one executing the shadow routine.

```
(all:0) > disp (3:0) shadow()iam
 ** gauss.f{}shadow()iam **
  ***** (3:0) *****
  iam = 1
```

Loading a Program for Debugging

Use the IPD **load** command to load applications for debugging. The arguments to the **load** command can include special arguments recognized by the **-nx** runtime start-up routine, such as **-sz**, **-pn**, and so on. The load command sets the default context. For parallel applications that use the special **-nx** runtime start-up routine, the default context is automatically set to include all compute processes that have the same ptype as the first program specified on the command line. For all other applications, the **load** command sets the default context to (host).

The following example loads the file *gauss* (compiled with the **-nx** option) on all nodes in the partition named foo, and sets the process type to 99.

```
ipd > load gauss -pn foo -pt 99
 *** reading symbol table for gauss... 100%
 *** loading program...
 *** initializing IPD for parallel application...
 *** load complete
(all:99) >
```

The following example loads the file *gauss* on 3 nodes in the .compute partition, sets the process type to 99, redirects input to come from the file *gauss.dat*, and passes the program the additional argument "100". Note that the input redirect is specified before the command line arguments for *gauss*. IPD supports input redirect, but does not currently support output redirect.

```
ipd > load gauss < gauss.dat -sz 3 -pt 99 100
 *** reading symbol table for gauss... 100%
 *** loading program...
 *** initializing IPD for parallel application...
 *** load complete
(all:99) >
```

The following example loads the file *gauss1* on node 0 in the .compute partition, sets the process type to 1, loads the file *gauss2* on nodes 1..3 in the .compute partition, and sets the process type to 2.

```
ipd > load gauss1 -on 0 -pt 1 \; gauss2 -on 1..3 -pt 2
 *** reading symbol table for gauss1... 100%
 *** loading program...
 *** initializing IPD for parallel application...
 *** reading symbol table for gauss2... 100%
 *** load complete
(0:1) >
```

The following example loads the file *gauss* (compiled without the **-nx** option).

```
ipd > load gauss
 *** reading symbol table for gauss... 100%
 *** loading program...
 *** load complete
(host) >
```

Controlling the Debug Environment

Control over the debug environment allows you to customize aspects of your debugging session to save time. This includes the following:

- Defining command aliases and setting debug variables.
- Recording debug sessions
- Creating and executing IPD command files
- Accessing online help.

Defining Aliases

You can customize the debug environment by defining aliases and debug variables. Aliases are your versions of the IPD commands, and debug variables are your versions of strings used in IPD commands. This allows you to create convenient shortcuts to commands you use most commonly.

The following example shows using the **alias** command to set an alias for the **step** command

```
(all:0) > alias s step
```

In the following example, the **alias** command is used to list all the current aliases.

```
(all:0) > alias
  Alias     Command String
  ======    ==============
  x         exec -echo
  c         continue ; wait
  s         step
```

Recording Debugging Sessions

You can record all or part of your debug session. Executing the IPD **log** command records all subsequently entered IPD commands and their responses in a log file.

The following example turns on session recording to the file *gauss.log*.

```
(all:0) > log gauss.log
```

You can also use the **log** command to display the name of the current log file, as shown in the following example.

```
(0:0) > log
       Log file: gauss.log
```

Command Files

You can create a file consisting of a set of IPD commands that you intend to execute more than once, and execute this file from within the debugger. In addition, you can create a special file containing commands that are to be executed whenever you invoke IPD. This file must be named *.ipdrc* and must reside in your home directory. This file can be used, for example, to define your standard alias and debug variable definitions.

The following example uses the **exec** command to execute the command file *picf*, which consists of the following lines:

```
load main -on 0 \; node -on 1..3
context (1..3:0)
break #84
break #90
```

When you execute this file, you get the following results:

```
ipd> exec -echo picf
ipd> ++ load main -on 0 \; node -on 1..3
       *** reading symbol table for main... 100%
       *** loading program...
       *** initializing IPD for parallel application...
       *** reading symbol table for node... 100%
       *** load complete
(0:0)> ++ context (1..3:0)
(1..3:0) > ++ break #84
(1..3:0) > ++ break #90
(1..3:0) >
```

Online Help

On-line help is also available as you use IPD. If you enter the **help** or **?** commands, IPD displays a brief description of all IPD commands. If you include the name of a command on the **help** command line, IPD displays detailed help for that command.

Controlling Program Execution

IPD gives you control over the execution of your program by allowing the following:

- Running, halting, and single-stepping through programs.
- Setting breakpoints, tracepoints, and watchpoints.
- Instrumenting programs for performance monitoring

Running a Program

You can start execution from the beginning of the program, continue after halting within the program, or single-step through the program.

When you issue a **run**, **rerun**, or **continue** command, execution of the specified processes is started, and a prompt is displayed, allowing you control over command entry while the program is running. If you issue the **wait** command, the prompt is not returned until all processes within the context stop, unless you issue a keyboard interrupt. It is important to be aware that executing processes are allowed to write to *stdout* and *stderr* in only two situations:

- Before each IPD prompt.
- During execution of a **wait** command.

While another command is executing, processes can only read from the keyboard if you issue a **wait** command.

NOTE

If you enter the command **continue;wait** to start the node processes when you are in the **host** context, the host process waits for an event to occur (such as a breakpoint) before the ipd prompt returns. If the host process starts the node processes and encounters **nx_waitall()** before encountering an event, the ipd prompt does not appear. You must interrupt the **wait** command to get the ipd prompt. This does not interrupt the host process. It instructs IPD to stop waiting for an event and return control to the user. Once control is returned you can use the context command to set your context to the node processes.

Breakpoints, Tracepoints, and Watchpoints

IPD allows you to set and remove breakpoints, tracepoints, and watchpoints. You can set breakpoints and tracepoints at procedure calls, source line numbers of executable statements, and instruction addresses. You can set watchpoints on variables or at data addresses. Breakpoints and watchpoints halt program execution. Tracepoints print a message, but do not halt execution.

The following example sets a breakpoint at line number 175 in the file *gauss.f*. The break occurs at the beginning of the tenth execution of the line 175 for process type 0 on nodes 1, 2, and 3.

```
(all:0) > break (1..3:0) gauss.f{}#175 -after 10
```

The following example sets a tracepoint at the procedure *shadow*() in the current source file for node 0, process type 0 only.

```
(0:0) > trace shadow()
```

The following example sets a watchpoint on write to the address *0x0401b7a8* for nodes 0 and 1, and process type 0.

```
(all:0) > watch (0..1:0) -write 0x0401b7a8
```

Performance Monitoring

You can instrument a program for performance monitoring with the IPD **instrument** command. This allows you to use **prof**, **gprof**, and ParaGraph to analyze your programs. For a complete description of **prof** and **gprof**, refer to Chapter 4. For a description of ParaGraph, refer to Chapter 7.

The following example gathers **prof** performance data on an application, *my_app*, for its entire run.

```
ipd > load my_app
(all:0) > instrument -prof
(all:0) > run
```

Examining and Modifying Programs

IPD provides numerous ways to examine and modify your program to aid in debugging, including the following:

- Source code listing.
- Program disassembly.
- Message queue display.
- Program variable, memory address, register, and stack traceback display.
- Assignment of new values to program variables, registers, and memory addresses.

Source Code Listing

With the **list** command, you can list source code from the current execution point, from a specified procedure, or from a source line number, specifying the number of lines to be listed. Line numbers are displayed in the listing.

In the following example the current context is (1:0). The **list** command is issued after the main program encounters a code breakpoint to display each source line you are stepping through.

```
(1:0) > run ; wait
 Context         State        Reason    Src/Obj Name   Procedure        Location
 =======         =====        ======    ============   =========        ========
*(1:0)           Breakpoint   C Bp 1    gauss.f        shadow           Line 180

(1:0) > step ; list,1
 Context         State        Reason    Src/Obj Name   Procedure        Location
 =======         =====        ======    ============   =========        ========
*(1:0)           Stepped                gauss.f        shadow           Line 180
  ***** (1:0) *****
  File: ./gauss.f
180:       if(iam.eq.0) then
(1:0) > step ; list
 Context         State        Reason    Src/Obj Name   Procedure        Location
 =======         =====        ======    ============   =========        ========
*(1:0)           Stepped                gauss.f        shadow           Line 194
  ***** (1:0) *****
  File: ./gauss.f
194:         leftid = irecv(type, a(1,1), length)
```

Program Disassembly

For debugging on a more detailed level, the **disassemble** command allows you to display assembly code.

In the following example, the current context is *(all:0)* in a Fortran program. The disassemble command is used to disassemble 16 instructions, starting at the procedure *shadow*().

```
(all:0) > disa shadow(),16
  ***** (all:0) *****
  gauss.f{}shadow() + 0x0
00000b18: ec1f1001  orh        0x1001, r0, r31
00000b1c: e7ff1c00  or         0x1c00, r31, r31
00000b20: 1fe01801  st.l       fp, 0(r31)
00000b24: a3e30000  mov        r31, fp
00000b28: 1fe00805  st.l       r1, 4(r31)
```

```
00000b2c: 1c7f87fd  st.l        r16, -4(fp)
00000b30: 1c7f8ff9  st.l        r17, -8(fp)
00000b34: 1c7f97f5  st.l        r18, -12(fp)
00000b38: 1c7f9ff1  st.l        r19, -16(fp)
00000b3c: 1c7fa7ed  st.l        r20, -20(fp)
00000b40: 1c7fafe9  st.l        r21, -24(fp)
00000b44: 1c7fb7e5  st.l        r22, -28(fp)
  gauss.f{}shadow()#165
00000b48: 147cffe9  ld.l        -24(fp), r28
00000b4c: 1470fffd  ld.l        -4(fp), r16
00000b50: 139d0001  ld.l        r0(r28), r29
00000b54: 12110001  ld.l        r0(r16), r17
(all:0) >
```

Message Queue Display

In parallel programs running on multiple nodes, many program errors are connected with messages passed among processes. IPD commands allow you to display queues of messages sent but not yet received, and receives that have been posted but not yet filled.

The following example displays all messages sent to process type 0 that have not been received.

```
(all:0) > msgq
 *** Unreceived messages in (all:0)
       Source            Destination       Msg Type   Msg Length
 =================  =================  ==========  ==========
 (0:0)               (2:0)                        2        7912
 (2:0)               (3:0)                        1        6048
 (1:0)               (3:0)                        2        7912
```

The next example displays all receives that have not been satisfied by an incoming message.

```
(all:0) > recvq
 *** Unsatisfied receives posted in (all:0)
Call Type   Recv Posted By        For Msg From      Msg Type   Msg Len   Handler
========= ================== ================== ========= ========= ========
CSEND      (0:0)               (2:1)              100        8
```

Variable, Address, and Register Display

The **display** command allows you to ensure that your program variables, registers, and memory addresses have the expected intermediate values. In addition, you can use the **frame** command to display a stack traceback, listing the routines accessed, and the files in which those routines are located. If a routine is compiled to produce debug information, line numbers are displayed; if not, memory addresses are displayed.

The following example displays the variable named *iam* in process 0 on node 0.

```
(0:0) > display iam
   *** gauss.f{} gauss() iam ***
   ***** (0:0) *****
   (0:0) iam = 4
```

In the following example, after a program was executed, it hung up, so execution was stopped. The **frame** command traces the stack to provide a history of the routines called, starting from the most recent. In this example, node 3 is found to have a different history than nodes 0, 1, and 2.

```
(all:0) > frame
  ***** (0..2:0) *****
    __flick()     [_flick.s{}0x00023fe8]
    _gdhigh()     [_gdhigh.c{}0x000240f8]
    gdhigh_()     [gdhigh_.c{}0x0001e9dc]
    gauss()     [gauss.f{}#72]
    main()     [pgfmain.c{}0x000001ac]
  ***** (3:0) *****
    __flick()     [_flick.s{}0x00023fe8]
    msgwait_()     [msgwait_.c{}0x0002011c]
    shadow()     [gauss.f{}#209]
    gauss()     [gauss.f{}#58]
    main()     [pgfmain.c{}0x000001ac]
```

Assigning Values

Another important feature is the ability to assign a new value to a program variable or memory location for the current run. This gives you the opportunity to see the result of such a change without having to edit and recompile your program before you know what the change will accomplish.

The following example assigns a new value to the variable *nbrnodes* in the current scope, using a context different from the default.

```
(all:0) > assign (3:0) nbrnodes=3
(all:0) > disp nbrnodes

 ** gauss.f{}shadow()nbrnodes **
  ***** (0..2:0) *****
  nbrnodes = 0
  ***** (3:0) *****
  nbrnodes = 3
```

The next example assigns a new value to the variable *iam* in the procedure *shadow*(), using the current context.

```
(3:0) > assign shadow()iam = 2
(3:0) > display shadow()iam

 ** gauss.f{}shadow()iam **
  ***** (3:0) *****
  iam = 2
```

Debugging Hints

The following sections provide additional information you should be aware of when using IPD.

Multi-Line Calls and Statements

Breakpoints and tracepoints may be set on only the last line of a multi-line C function call, because line number information is generated only for the last line of the call. In the following example, the breakpoint must be set on the line where the *l* is:

```
printf( "%d %d %d %d\n",
                      i,
                      j,
                      k,
                      l );
```

For multi-line Fortran statements, breakpoints and tracepoints can be set only on the first line of the statement. In the following example, the breakpoint must be set on the "print *" line:

```
      print *,
    &       'is ',
    &       'a ',
    &       'multi-line statement.'
```

Referencing Unnamed Fortran Main Programs

Fortran programs are not required to have a PROGRAM statement. If the PROGRAM statement is omitted, the main routine is given the name *_unnamed()*. You need to be aware of this when you are qualifying breakpoints or variables in the main routine.

Displaying Fortran Variable Types

Fortran data types are represented as shown in Table 2-4. The display of some of the variable types (those shown with "<---" after them) may be unexpected. This is because the debug information generated by the compiler is not sufficient to distinguish the declared type from the type displayed by IPD in these instances.

Table 2-4. Fortran Variable Type Display

Declared type	Represented as
character var	CHARACTER*1 var
character*n var	CHARACTER*n var
character*n var(x,y)	CHARACTER*n var(x,y)
logical*1 var	LOGICAL*1 var
logical*1 var(x)	LOGICAL*1 var(x)
logical*1 var(x,y)	LOGICAL*1 var(x,y)
logical*2 var	INTEGER*2 var <---
logical*4 var	INTEGER var <---
logical var	INTEGER var <---
integer*2 var	INTEGER*2 var
integer*4 var	INTEGER var
integer var	INTEGER var
real*4 var	REAL var
real var	REAL var
real*8 var	DOUBLE PRECISION var
double precision var	DOUBLE PRECISION var
complex var	COMPLEX var
complex*8 var	COMPLEX var
complex*16 var	DOUBLE COMPLEX var

Using Keyboard Interrupts

The following information is for using keyboard interrupts during program execution:

- There are critical sections in the debugger where IPD does not allow you to interrupt it from the keyboard. This is necessary because there are data structures (for keeping track of processes, breakpoints, etc.) that must be synchronized at all times. You are not allowed to interrupt during the modification of these data structures.

- If you are sure that IPD has hung up and is not going to respond, using the control-backslash (**<`Ctrl-\ `>**) key sequence kills the debugger.

XIPD 3

This chapter describes XIPD, a graphical interface to the Interactive Parallel Debugger. XIPD makes it easier for you to debug your parallel applications, because XIPD provides continuous update of node status, indicates which routines are currently executing, and notes where the execution point is in the code. XIPD also allows you to debug your parallel applications without having to master the syntax of IPD's command language.

XIPD graphically depicts the status of the nodes that are loaded with your applications. You can see at a glance which nodes are executing, stopped at a break point, or terminated. Each routine has its own code window, from which you can set breakpoints, find which lines are currently executing, display variable values, or modify variable values.You can also examine the message queues in the mesh to find which nodes have posted unreceived sends and which nodes are blocked waiting to receive a message.

XIPD also provides online, context-sensitive help.

Certain choices you make while using XIPD are saved to a session file. You can recall this information the next time you use XIPD to reduce the amount of time you have to spend providing startup information to XIPD.

You can use XIPD as a stand-alone graphical interface or from ParAide, the graphical front end to the Paragon toolset. Chapter 1 describes ParAide. XIPD also allows you to gather performance information and provides a quick path to execute performance monitoring tools such as ParaGraph, XProf, and XGprof.

Using XIPD

This section describes how to use XIPD to debug parallel applications. For detailed information about individual menus and dialogs, refer to the section *Windows, Menus, and Commands.*

Table 3-1 lists the IPD commands and the level of support that XIPD provides for each command.

Table 3-1. XIPD Support of IPD Commands (1 of 2)

Command	Support	Explanation
alias	none	Not used.
assign	partial	Can not assign to addresses or registers.
break	partial	Set everywhere a program is loaded. Can't set a breakpoint on an address. No *count* argument.
context	none	Not used.
continue	partial	No **-nosignal** option
disassemble	none	Not used.
display	partial	Only one variable at a time. No display of address space or registers.
exec	none	Not used.
exit	none	Not used (**quit** is used).
flush	full	Used after instrumentation.
frame	full	Used based on current viewpoint.
help	none	Not used.
instrument	full	Fully supported.
kill	full	Always used with **-force** option.
list	partial	Only used to list an entire function.
load	full	Fully supported.
log	none	Not used.
more	partial	**more -off** is part of initialization.
msgqueue	partial	No **-type** option.
process	partial	No **-change** or **-loadfile**.
quit	full	Used to exit.
recvqueue	partial	No **-type** option
remove	partial	No **-all** option.
rerun	partial	Used partially. Cannot change arguments.
run	none	Not used.
set	none	Not used.
source	partial	No **-add** or **-remove** options.

Table 3-1. XIPD Support of IPD Commands (2 of 2)

Command	Support	Explanation
status	none	Not used.
step	partial	no *count* parameter.
stop	full	Fully supported.
system	full	Fully supported.
trace	partial	Set everywhere a program is loaded. Can not be set for an address. No *count* argument.
type	none	Not used.
unalias	none	Not used.
unset	none	Not used.
wait	partial	Only as part of **continue;wait** command to start a controlling process.
watch	partial	Set everywhere a variable's program is loaded. Can not be set for an address. No *count* argument.

Invoking XIPD

To invoke XIPD, use the **xipd** command as follows:

xipd [*session_name*] [*session_option*] [*control_option*] [*display_option*] [*X Toolkit parameters*]

The command parameters are defined as follows:

session_name	The name of a previously-saved XIPD session. If the session file exists, XIPD uses the contents to pre-initialize certain aspects of XIPD. If the file does not exist, *session_name* is assumed to be the name of the Paragon system that XIPD should be used with.

session_option can be any of the following:

-user *account_name*	The login account name to use for starting a new session on a Paragon system. Use this option when you are not restoring a session and you want to specify a login account name.
-host *paragon_name*	The name of the Paragon system on which XIPD is being used. Use this option when you are not restoring a session.

control_option can be any of the following:

-login	Forces XIPD to display the login panel.
-nologin	Skips the login panel and starts the session.
-delay seconds	Establishes the time-out period for XIPD to wait for a response from IPD. This is rarely needed and typically only used when debugging applications that can deadlock when the **step** command is issued. After the given whole number of seconds (the default is 60), XIPD interrupts IPD if IPD has not responded to the XIPD command.

display_option can be any of the following:

-[no]debug	[Do not] create interface elements specific for debugging applications (like setting breakpoints or stepping execution). The default is **-debug**.
-[no]analysis	[Do not] create interface elements specific for instrumenting programs for performance analysis. The default is **-analysis**.
-[no]symbols	[Do not] use symbol shapes to represent node status. The default is **-symbols**.
X Toolkit parameters	The standard parameters supported by the X Toolkit (refer to the *X Toolkit Intrinsics Programming Manual*).

For complementary options (like **-debug** and **-nodebug**), the option occurring later in the command line takes precedence if both are present. Also, the options **-nodebug** and **-noanalysis** cannot be used together on the same command line.

When you invoke XIPD, it displays the startup dialog. The following sections describe the XIPD session history file, how to create a new XIPD session, and how to start up XIPD.

Session History File

XIPD uses session file to restore your choices from the last invocation of XIPD that used the specified session. The history information includes the following:

- The name of the Paragon system used.
- The Internet address of the Paragon system,
- Your login name on the Paragon system.
- The size of the last mesh used, including height and width.

The session file is typically stored in your home directory. When XIPD creates a session file, the name you specify for the file is prefaced with a period. For example, if you name a session *Galaxy*, XIPD creates a *.Galaxy* session file in your home directory.

If you prefer to locate your session files in another directory, you can define the environment variable *XIPDHOME* to a directory to store the session files. For example, if you defined *XIPDHOME* to */home/username/misc/sessions,* the session *Galaxy* would be created with the file name */home/username/misc/sessions/.Galaxy.*

Creating a New Session

Beginning a new session involves starting a new shell on the Paragon system and establishing the session name, if you are not using information from a previous session.

Start-up

When XIPD begins execution without using restored session information, the first dialog displayed is the start-up dialog. This dialog is also used for logging on to the Paragon system from a workstation. Figure 3-1 shows the start-up dialog.

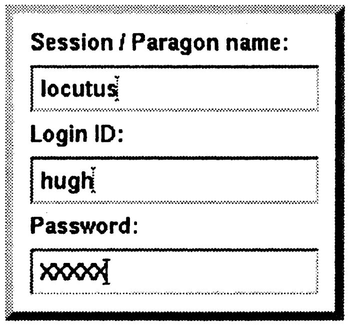

Session / Paragon name: locutus
Login ID: hugh
Password: xxxxx

Figure 3-1. Start-up Dialog

Session / Paragon name

This field contains the name of a previous session to be restored or the name of the Paragon system that XIPD logs you onto.

If you started XIPD with a session name and the corresponding session file was read in, this field contains the full name of the Paragon system stored in the session file. If a session name was given but no session file was found, this field contains the given session name, which XIPD assumes to be the name of the Paragon system. If you start XIPD without any session name, this field is blank and you must type in the name of a Paragon system or a session name.

Login ID

This field contains your login identification on the Paragon system.

If you start XIPD without old session information, this field is initialized to your identification on the machine running XIPD, which is usually the same identification as your Paragon system identification. You can override this with the **-user** command line option.

Password

This field contains your password on the Paragon system. XIPD echoes an *X* for each keystroke you enter.

This field can remain blank for any machine that you can perform an *rlogin* onto without providing a password. If you are using a different account, or if XIPD is running on a remote workstation, a password is required.

To be able to start a new shell on the Paragon system without having to provide a password, have your system administrator add the machine name of the Paragon system to the */etc/hosts.equiv* file that resides on the Paragon system.

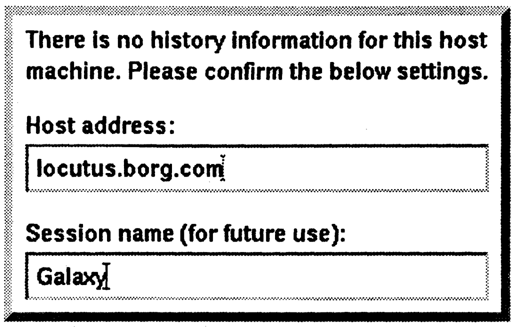
Control Buttons

OK	Starts a new shell on the Paragon system. If the host name is invalid or XIPD can not start a new shell with the given account name and password, an error dialog is displayed and you are given a chance to correct the start-up dialog entries.
Cancel	Abandons starting a new shell on the Paragon system and XIPD exits.
Help	Displays help text about the start-up panel.

Session Name

The session name dialog appears after a successful login without a session file. XIPD assumes that no session file information exists yet for this Paragon system or that you want to create a different session file. You will not see the history dialog again if you continue to use XIPD with the session file stored with the session name dialog.

Figure 3-2 shows the session name dialog.



There is no history information for this host machine. Please confirm the below settings.

Host address:

locutus.borg.com

Session name (for future use):

Galaxy

Figure 3-2. Session Name Dialog

Host address

This field is for confirmation purposes only. When logging on to the Paragon system, XIPD looks up the Internet name and address. This is usually correct, but you can modify the field. If you change this field, XIPD confirms that the new entry is a valid machine address.

Session name

This becomes the session name that you can use for future reference. For example, if you logged on to a machine called *locutus.borg.com* and want to create a new session for a galaxy formation study, you could enter the name *Galaxy* in this field. You could then start future XIPD sessions with the *session_name* command line argument *Galaxy*, and all saved information would be restored from your last session.

Control Buttons

OK	Confirms the dialog entries and dismisses the dialog. The *Host address* field must be a valid machine address and the *Session name* field must not be empty. XIPD displays an error dialog if there are any problems with your entries, and the session dialog remains on screen.
Reset	Undoes all the changes to the field contents.
Help	Displays help text about the session name dialog.

Once you have completed the start-up, XIPD displays the main window. Figure 3-3 shows the main window.

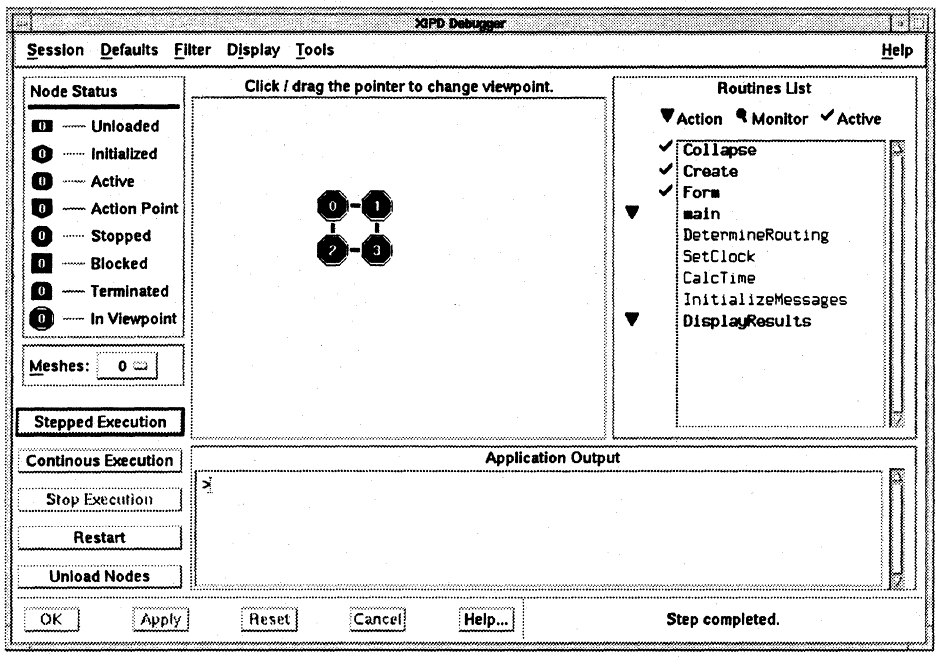


Figure 3-3. XIPD Main Window

The following sections describe some of the basic functions of XIPD.

Loading a Program

To load a program, you must select an existing partition and determine the size of the mesh to reside within the partition with the mesh configuration dialog in the main XIPD window. You can load multiple programs and allocate new process types.

Selecting a Mesh and Nodes

a You must first log on to a Paragon system with XIPD before you can load any programs into IPD. This step is skipped if you use a session file that identifies the Paragon system to be used as the same machine that XIPD is running on. The login dialog is displayed if XIPD is running on a workstation instead of a Paragon system.

After you have logged on to the Paragon system, you are asked to select a partition and to select which nodes within the partition compose the mesh for loading programs. All partitions are displayed in a scrollable tree that represents the hierarchy of the partitions. When you select a partition, information about it is listed beneath the menu and the location of the partition in the entire mesh is displayed. You can then select the nodes belonging to the partition for loading, and enter the height and width of the load mesh consisting of these nodes.

The height and width of the load mesh entered are checked against the number of nodes selected from the partition. After you choose the partition and load mesh, you are ready to load programs.

Loading Programs into the Mesh

The mesh you selected is drawn on the screen. To load a program into a node, select the node (by clicking on it) and enter a file name. You can select more than one node at once if you hold down the mouse button and drag it, forming a rectangle. All unloaded nodes that are within this rectangle are selected. Once you press the *OK* button, XIPD loads the program into the selected nodes.

If you want to load more than one program, you should select the nodes, enter the program name, and select *Apply* instead of *OK*. The selected nodes are changed to denote that they are loaded and cannot be selected anymore.

Some parallel applications use process types. The XIPD load mesh defaults to process type zero. To allocate a new process type, you must select the *New Mesh* button. XIPD requests a process type number. If you provide one, XIPD allocates a new mesh for operations pertaining to this process type. XIPD organizes its meshes according to process types, and an option menu on the load dialog lets you switch between the different process types.

Once the load is completed, XIPD gathers what debug information it can about the programs and you can then begin execution and examination of the loaded programs.

Establishing a Viewpoint

The main window viewpoint shows the status of the mesh. By looking at the viewpoint mesh, you can tell which nodes are executing, at a break point, interrupted, or terminated. The viewpoint is also used to limit the amount of information gathered by XIPD and can be used to exert specific execution control over the nodes.

Nodes in the mesh are considered either "in" or "out" of the viewpoint. If a node is in the viewpoint, it has a border drawn around it. When IPD requests information, XIPD asks only about nodes that are in the viewpoint. Therefore, you can reduce the viewpoint down to only a few nodes. Information, such as pending messages, is required for only for those nodes, reducing unwanted information.

Clicking on a node in the mesh toggles its state between in or out of the viewpoint.You can change the status of a group of nodes by dragging the mouse pointer and enclosing the nodes in the resulting rectangle.

Like the load panel, the viewpoint mesh is organized by process types. The information for only one process type can be displayed in the viewpoint at one time. If you have multiple process types (allocated with the load dialog or during program execution) they are placed in the main window *Meshes* pop-up option menu. Selecting a process type from this menu changes the mesh in the viewpoint to the process type.

NOTE

The XIPD viewpoint is not equivalent to the IPD context. The inclusion or exclusion of nodes from the viewpoint is used to reduce the amount of information gathered about the nodes. Viewpoint only applies to execution if the *Viewpoint Applies to Execution* item under the *Defaults* menu is selected.

Working with a Program's Source Code

The main window *Routine List* displays all the routines that XIPD is aware of. To see the source code for a routine, click on the routine name in the list. XIPD displays a code window for the selected routine. Initially, all known routines are displayed in the routine list. The list can, however, be filtered according to the settings in the *Filter* menu. This allows you to list only certain routines, such as routines that have breakpoints set, for example.

In order for a routine name to be displayed in the routine list, the following must be true:

- The source file that contains the routine is compiled with debug information.
- The source code for the routine has been found.

XIPD defaults to searching for an application's source code once the debug-compiled application is loaded. Typically, this source code is within the same directory as the loaded application and XIPD finds the code. If XIPD cannot find the code, or if you decide to turn off XIPD's default searching, you are asked to help find the source code. If you cancel the source code search, any routines that reside within the unlocated source are not added to the routine list.

Setting Action Points

To set an action point (such as a breakpoint), bring up a routine's code window and click next to the line where you want the action point set. An action point icon shows up next to the line. If you click on the action point icon, XIPD removes the action point.

If at least one action point is set for a routine, a breakpoint symbol is put next to the routine name in the routine list.

If you click next to a line that isn't executable (such as a comment), XIPD sets the action point on the next executable line in the routine.

The code window *Action* pull-down menu allows you to specify what kind of action point should be set for an executable line. The default is a breakpoint.

Setting Data Watchpoints

The *Set Data Watch Point* option in the code window displays a dialog from which you can set a data watchpoint for a program. This causes the program to stop when the variable is written to or accessed (read or written).

Obtaining Performance Data

The instrumentation dialog (brought up with the *Tools* menu) sets monitor points within a program. You must select which tool is going to be used for analysis (prof, gprof, or ParaGraph), enter the file name (or directory name) where the output should be saved, and indicate where performance monitoring should start and stop within the code.

Executing a Program

When execution begins, XIPD notes which routines are currently executing. An active icon is put next to the name of an executing routine in the routine list. If the code window is displayed, the currently executing lines are noted with active icons. There might be more than one line executing at a time in a routine. The *Find Active Lines* button in the code window scrolls the code window to each active line and displays a message about which nodes are executing the line.

A source code line is considered active if the line is the current point of execution. A line is not considered active if the line is a call in progress to another routine (such as *crecv*()).

Examining and Changing Variables

The *Display Data Value* option in the code window displays a dialog from which you can obtain values for variables within the routine. The nodes selected within the current viewpoint control which nodes are requested for the variable's value.

The *Assign Data Value* option in the code window displays a window from which you can change the value of a variable. Like the value examination dialog, the nodes selected in the current viewpoint control which variables are changed to the value given.

Executing the Program and Examining Messages

Once you have loaded a program and set action points, the program can begin execution. During execution, you can ask XIPD to retrieve and display runtime information, such as variable values, message queue contents, and node execution trace backs.

Execution Control

The execution controls consist of a set of vertically arranged buttons on the main panel: *Stepped Execution*, *Continuous Execution*, *Stop Execution*, *Restart*, and *Unload Nodes*. These buttons are disabled when they do not apply to the current state of the nodes. For example, if no nodes are executing, the *Stop Execution* button is disabled. The buttons are also locked-out (a busy watch cursor appears over them) when IPD is busy processing a command sent by XIPD. Once XIPD receives the response, the buttons are unlocked. This avoids a buildup of commands while IPD is processing a command.

By default, XIPD generates execution commands for all nodes. For example, when you select *Stepped Execution*, all nodes that can step are given the **step** command. You can change this by setting the *Viewpoint Applies to Execution* item in the *Defaults* menu. If this option is set, XIPD sends execution commands only to the nodes that are part of the viewpoint.

Stepped Execution or *Continuous Execution* begins execution of whatever has been loaded into the mesh. Since *Continuous Execution* issues a **continue** command, execution on a node continues until a breakpoint is encountered, the node's program terminates, or you select the *Stop Execution* button.

The *Stepped Execution* button issues a **step** command to all nodes capable of stepping.

The *Stop Execution* button issues a **stop** command to all nodes that are executing. This is necessary when nodes become blocked while waiting to receive a message.

The *Restart* button issues a **rerun** command to all nodes that are loaded.

The *Unload Nodes* button unloads all loaded programs from the mesh. A question dialog is displayed to ask you if this is what you really want to do. If so, all programs are killed and the load dialog is displayed.

Check Messages and Execution Tracebacks

The pending sends and receives posted by nodes are displayed when you select the *Pending Sends* or *Pending Receives* menu items. Only messages related to nodes within the current viewpoint are displayed. All nodes must be stopped to request this information from IPD.

XIPD displays the execution trace dialog when you select the *Traceback* menu item. This dialog displays where a node currently is executing and how it got there. Only the nodes that are contained in the current viewpoint are displayed in the traceback.

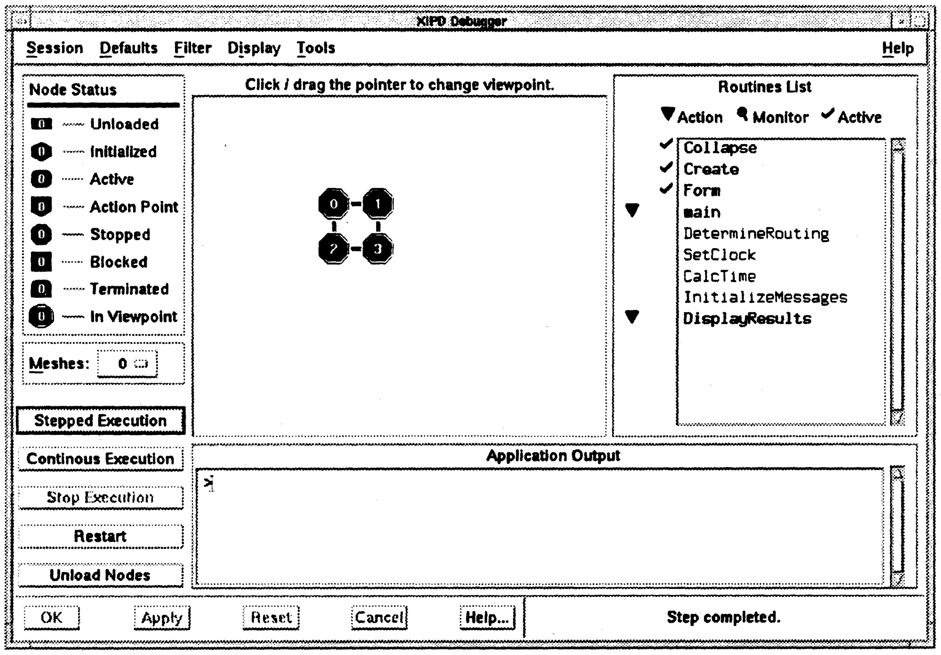
On-Line Help

There are several ways to access online help for XIPD. The most direct is to choose *Index* from the *Help* menu. This displays a list of all XIPD help topics. Selecting a topic displays help text for that topic. The *Help* menu also contains the names of the major topics. Selecting a major topic displays the help text for the topic.

Context-sensitive help provides help about a particular part of the XIPD interface. The *On Context* menu item enables context-sensitive help and changes the mouse pointer to a question mark. You can then click on areas of the XIPD interface. If there is a help entry for the selected area, XIPD displays the help topic text for that feature. You can also obtain context help from the keyboard by pressing the *Help* key (typically mapped to F1) while the keyboard focus is directed to the desired interface feature.

Windows, Menus, and Commands

The main window of XIPD contains menus and control regions for XIPD. Figure 3-4 shows the main window.



Step completed.

Figure 3-4. Main Window

The menu bar across the top of the main window contains the following menus:

- *Session*
- *Defaults*
- *Filter*
- *Display*
- *Tools*
- *Help*

The inner region of the main window contains the following areas:

- a node status legend
- a mesh process type option menu
- a set of execution control buttons
- a node viewpoint panel
- a routine list
- a region for application output
- a set of standard control buttons
- a region for one line status messages

The appearance of XIPD changes if analysis or debugging are turned off. For example, the *Tools* menu is not present when you invoke XIPD with the **-noanalysis** command line option, and the *Stepped Execution* button is not present when you invoke XIPD with the **-nodebug** command line option

Session Menu

The session menu contains items associated with using the Paragon system, including the following:

- selecting a mesh within a given partition
- loading a program
- establishing runtime arguments to the program
- setting display preferences
- locating the source code for a loaded program
- executing a system command directly
- exiting XIPD (and IPD)

Figure 3-5 shows the session menu.

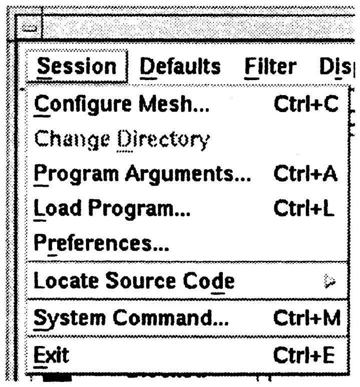


Figure 3-5. Session Menu

Configure Mesh

Configure Mesh displays the mesh configuration dialog. This item is enabled once you have logged onto the Paragon system. If you have already selected a mesh, a question dialog appears to make sure you want to abandon anything loaded into the current mesh in order to select a new mesh. The default keyboard accelerator for this item is *Control-C*.

Change Directory

Change Directory displays a file selection dialog to allow you to change to a specific directory before IPD is executed. This is needed if the programs being debugged must execute in a specific directory (for example, a program designed to load a local input file in its own directory). XIPD normally executes IPD from your home directory. *Change Directory* is only enabled during mesh configuration.

Program Arguments

Program Arguments displays the program arguments dialog. *Program Arguments* is enabled once you choose a mesh for loading. If a program is being loaded with command line arguments or with standard input redirection, choose this item and make entries into the program arguments dialog before you select *Apply* or *OK* for the load. The default keyboard accelerator for this item is *Control-A*.

Load Program

Load Program displays the program load dialog unless you select it when there is already at least one program loaded into the current mesh. If this is the case, you are informed that the mesh must be unloaded first before it can be reloaded. This item is enabled when you select a mesh. The default keyboard accelerator for this item is *Control-L.*

Preferences

Preferences displays the XIPD preferences dialog. *Preferences* is always enabled.

Locate Source Code

Locate Source Code is a pull-right menu item, meaning you must select the item and then move the mouse to the right into a subsequent menu popped-up next to this menu item. This sub-menu contains the name of all loaded programs that are compiled with debug information. If there are no loaded programs that are compiled for debug, *Locate Source Code* is not enabled.

When you choose an item in the sub-menu, XIPD displays the source location dialog for the program contained in the sub-menu item. This allows you to bring up the source location dialog for a specific program. This is useful to find a program's source files to obtain routine information about each file.

System Command

System Command displays the system command dialog. This item is enabled once you have successfully logged on the Paragon system. The default keyboard accelerator for this item is *Control-M*.

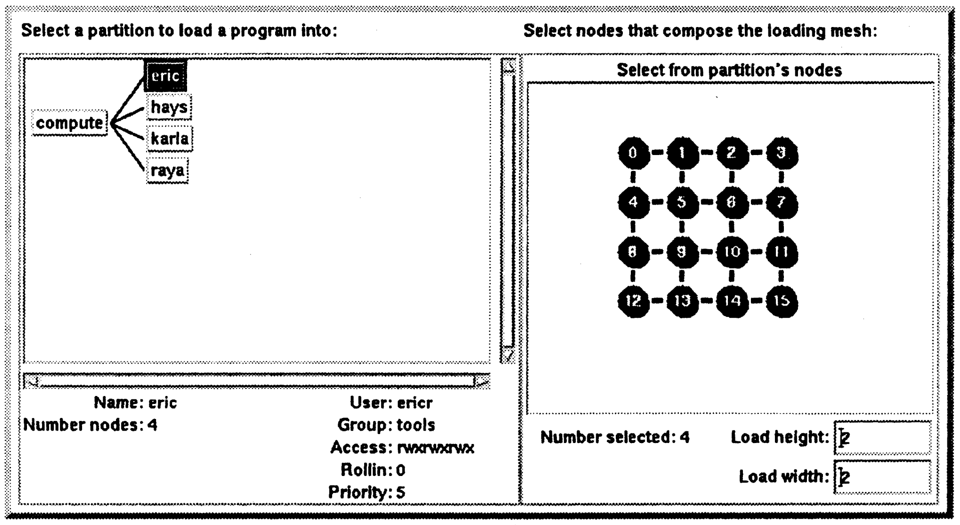
Exit

Exit quits XIPD. XIPD displays a question dialog to ask if you really want to quit. If you choose *Yes*, XIPD exits. Otherwise, XIPD continues to execute. The default keyboard accelerator for this item is *Control-E*.

Mesh Configuration Dialog

Use the mesh configuration dialog to select a partition to load into and to set the load mesh size. The configuration dialog appears automatically after you have logged onto the Paragon system and the history dialog, if displayed, has been completed.

You must select a partition from all of the partitions currently allocated on the Paragon system. You must also enter the height and width of the mesh you want to use within the selected partition. Figure 3-6 shows the mesh configuration dialog.


Select a partition to load a program into:
Select nodes that compose the loading mesh:
Select from partition's nodes
compute
eric
hays
karla
raya
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Name: eric
Number nodes: 4
User: ericr
Group: tools
Access: rwxrwxrwx
Rollin: 0
Priority: 5
Number selected: 4
Load height: 2
Load width: 2

Figure 3-6. Mesh Configuration Dialog

Select a partition...

This field contains a scrollable tree that represents the hierarchy of the computation partitions in the Paragon system's mesh. A sub-partition is drawn as a child branch off of its parent partition. The current partition selection is noted by being inverted in color. When you select a partition, the fields beneath the tree are updated.

The fields have the following meaning:

Name	Name of the selected the partition.
Number nodes	Number of nodes contained within the partition.
User	User name of the partition's creator (and owner).
Group	Group name of the partition's creator.
Access	Access permissions to the partition. This is similar to access permissions to a file. In order to load and execute programs within the mesh, the mesh permissions must be configured to grant you execute permission.
Rollin	The rollin quantum for the partition.
Priority	The partition's maximum priority.

If the environment variable *NX_DFLT_PART* is defined to a valid partition name, XIPD selects that partition initially. Otherwise, the initial selection is the *.compute* partition.

Select mesh...

This scrollable region represents the physical mesh. When you select a partition from the partition tree, this mesh is updated to reflect where the selected partition resides in the physical mesh. You must select which of the partition's nodes are used for loading a program or a set of programs. Each node has three states:

Not in Partition	The node doesn't belong to the selected partition and cannot be selected for loading a program.
Not Selected	The node belongs to the selected partition but has not been selected for loading.
Selected	The node belongs to the selected partition and has been selected for loading.

The node states are indicated by their coloring. The default coloring for the states is listed in the *Default Configuration* section.

Number selected

This field counts the number of nodes selected from the partition.

Load height and Load width

These fields determine the height and width of the mesh containing the nodes selected for loading a program. For example, if you select a sixteen node partition and you choose to load into eight of the partition's nodes, the product of the values in the *Load height* and *Load width* fields must equal eight (for example, one and eight, two and four, four and two, or eight and one). XIPD calculates default values for these fields based on the number of nodes selected. If the previous session's height and width match the number of nodes selected, the session's saved height and width are used instead of the default height and width

Control Buttons

OK	When you select *OK*, XIPD checks the mesh height and width against the number of selected partition nodes to be sure they match. If they match, XIPD dismisses the configuration dialog and displays the load dialog. If they do not match, XIPD displays an error dialog and the configuration dialog remains.
Cancel	*Cancel* dismisses the configuration dialog. XIPD cannot load anything until the mesh has been selected. The configuration dialog can be redisplayed by selecting the *Configure Mesh* menu item.
Help	*Help* displays help topic text regarding mesh configuration.

Load Dialog

Use the load dialog to load programs into the mesh. Each mesh has a process type number, with the default mesh having process type zero. You can add other process types with the *New Mesh* button. For each process type, XIPD allocates a new mesh.

Figure 3-7 shows the load dialog.

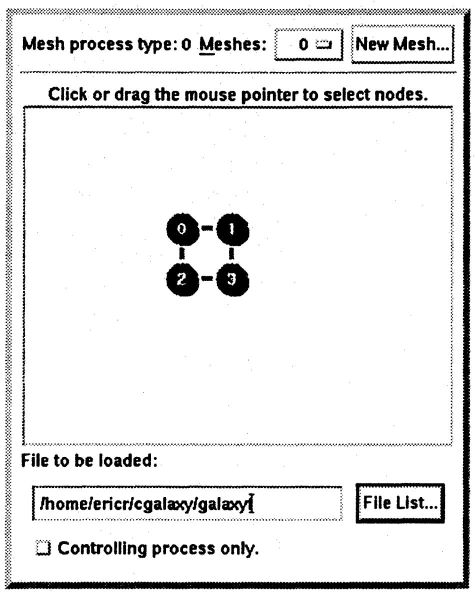

Mesh process type: 0 Meshes: 0
New Mesh...
Click or drag the mouse pointer to select nodes.
0
1
2
3
File to be loaded:
/home/ericr/cgalaxy/galaxy
File List...
❑ Controlling process only.

Figure 3-7. Load Program Dialog

Mesh process type

This field contains the process type of the current mesh. When you allocate a new mesh process type or change the display to an existing mesh, this field is updated to the new process type.

Meshes

The *Meshes* pop-up menu contains all of the allocated process types. When you select an entry from this list, the nodes are updated to the corresponding mesh and the *Mesh process type* label displays the selected process type.

Figure 3-8 shows the mesh options menu.

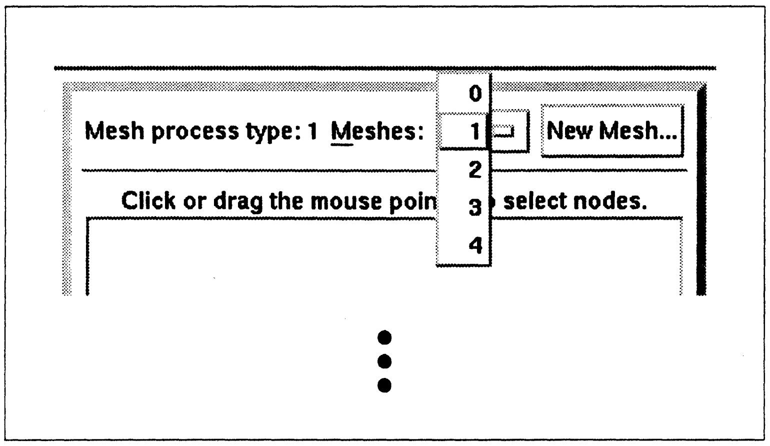

0
Mesh process type: 1 Meshes: 1
New Mesh...
2
Click or drag the mouse poin 3 select nodes.
4

Figure 3-8. Mesh Options Menu

New Mesh

New Mesh... displays a dialog that allows you to add a new process type. Figure 3-9 shows the new mesh process type dialog.

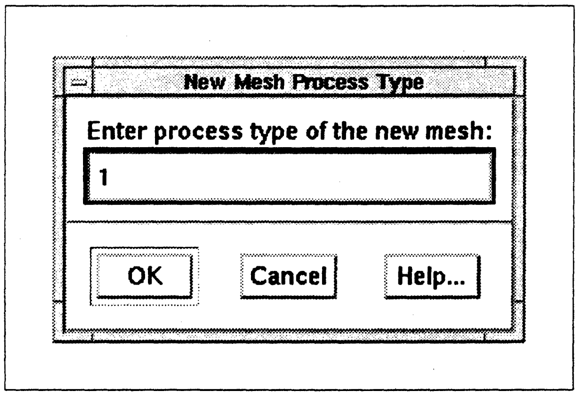


Figure 3-9. New Process Type Dialog

You can type a new process type into the text field. When you select *OK*, the dialog is dismissed and a new mesh appears in the load dialog. The process type is added to the *Meshes* pop-up menu and the text field is updated to the new process type.

If you select *Cancel*, no new process type is allocated.

Selecting *Help* provides help topic text about allocating a new process type.

Node Selection

You can select nodes in the mesh by clicking on them or by holding the mouse button down and dragging the mouse pointer down and to the right, causing a rectangle to form around nodes. Any nodes in the rectangle when you release the mouse button become selected. If you hold down the *Control* key during the rectangle-drag, the nodes within the rectangle "toggle" their state. Unloaded nodes change to *Selected* and selected nodes change to *Unloaded.*

Nodes are colored to represent three different states:

Unloaded	Nothing has been loaded into the node. If you select the node, it changes to *Selected.*
Selected	The node has been selected for loading. The next load operation for the current mesh loads the given program into this node. If you select the node, it changes to *Unloaded.*
Loaded	The node has already been selected and loaded into. Once loaded, you can not select the node again and the node does not respond to any attempt to select it.

File to be loaded

This field contains the name of the executable to be loaded into the selected nodes. You can type the executable file name into this field, or select the *File List...* button and find the file with the file selection dialog.

File List

File List displays a file list dialog next to the load dialog. If you select a file from this dialog, the file name is copied into the *File to be loaded* field.

Controlling process only

This toggle button indicates that the program to be loaded is a controlling process that loads programs into the mesh (XIPD is notified by IPD about the loaded programs). In this case, no nodes should be selected for loading, since this is the responsibility of the controlling process and you should only use *OK* to load the program.

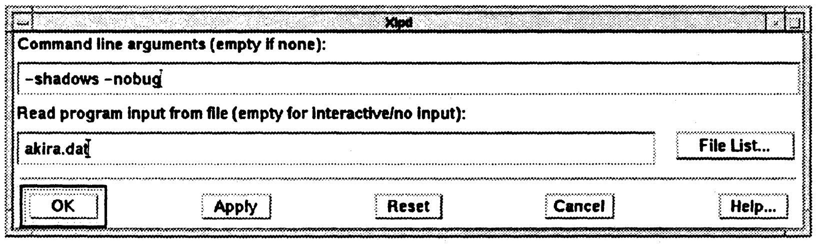
The *Controlling process only* toggle button is disabled after you select *Apply,* since selecting *Apply* indicates that multiple parallel programs are to be loaded.

Control Buttons

OK	When you select *OK*, XIPD notes the load of the given file name into the selected nodes and dismisses the load dialog. An error dialog appears in the following cases: • no file name is given • no nodes are selected and *Controlling process only* is not set • nodes are selected and *Controlling process only* is set. If there is an error, XIPD does not dismiss the load dialog.
Apply	When you select *Apply*, XIPD notes the load of the given file name into the selected nodes. The selected nodes change to the *Loaded* state. An error dialog appears in the following cases: • no file name is given • *Controlling process only* is set • *Controlling process only* is not set and no nodes are selected
Reset	Changes all nodes in the *Selected* state to the *Unloaded* state.
Cancel	All changes since the last *Apply* are undone and the load dialog is dismissed. Some nodes might be loaded if you have selected *Apply*. If so, XIPD issues the **load** command.
Help	Displays help topic information about the load dialog.

Program Arguments Dialog

The program arguments dialog establishes runtime arguments to the loaded programs and redirects standard input to come from a file. Figure 3-10 shows the program arguments dialog.


Xipd

Command line arguments (empty if none):

-shadows -nobug

Read program input from file (empty for interactive/no input):

akira.dat

Figure 3-10. Program Arguments Dialog

Command line arguments

Any text typed in this field is passed to the **load** command as command line arguments to the next program that you select *OK* or *Apply* for in the load dialog.

Input file

This field contains the name of a file to be used as standard input for the loaded programs (rather than reading from the keyboard). You can type in the file name or select *File List* to display the file selection dialog. For a multi-program load, you can only specify one input file. XIPD uses the contents of the *Input file* field at the point you select *OK* in the load dialog for the **load** command sent to IPD.

File List

File List displays the file selection dialog. If you select a file, the name is copied into the *Input file* field.

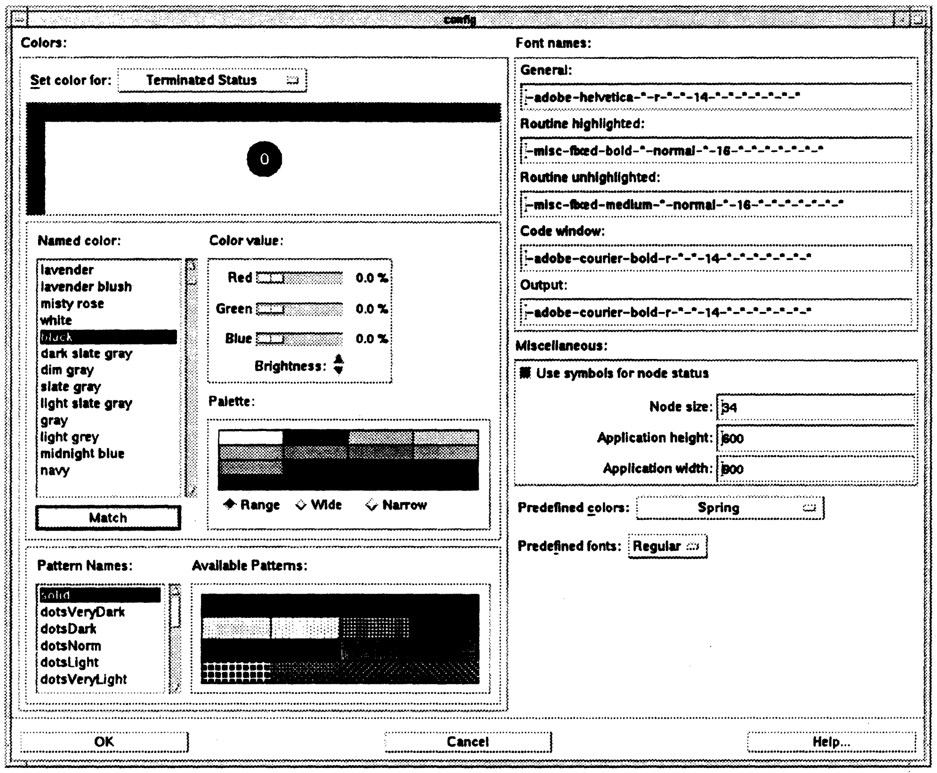
Dialog Buttons

OK	Dismisses the program arguments dialog and uses the information entered for the next loaded program.
Apply	Establishes the field entries to be used with the next loaded program. The dialog stays on the screen.
Reset	Replaces any changes to the two fields with what the fields contained at the last *Apply* or (if *Apply* hasn't been selected) when the dialog was displayed.
Cancel	Discards any changes to the two fields and dismisses the dialog.
Help	Displays help topic text about the program arguments dialog.

Preferences Dialog

The preferences dialog configures the appearance of XIPD. Changes are written to your *XIpd* resource file. Figure 3-11 shows the preferences dialog.

You are encouraged to use the entries under the *Predefined colors* and *Predefined fonts* option menus for setting XIPD colors and fonts for the best visual effect.



Set color for: Terminated Status

Named color: lavender, lavender blush, misty rose, white, black (selected), dark slate gray, dim gray, slate gray, light slate gray, gray, light grey, midnight blue, navy

Red 0.0 %
Green 0.0 %
Blue 0.0 %
Brightness:

Pattern Names: solid (selected), dotsVeryDark, dotsDark, dotsNorm, dotsLight, dotsVeryLight

Font names:

General: -adobe-helvetica-*-r-*-*-14-*-*-*-*-*-*

Routine highlighted: -misc-fixed-bold-*-normal-*-16-*-*-*-*-*-*-*

Routine unhighlighted: -misc-fixed-medium-*-normal-*-16-*-*-*-*-*-*-*

Code window: -adobe-courier-bold-r-*-*-14-*-*-*-*-*-*

Output: -adobe-courier-bold-r-*-*-14-*-*-*-*-*-*

Miscellaneous:

■ Use symbols for node status

Node size: 34

Application height: 600

Application width: 800

Predefined colors: Spring

Predefined fonts: Regular

Figure 3-11. Preferences Dialog

Set color for

This button pops up an option menu containing the various elements of XIPD for which you can choose a color value. A representation of the color and appearance of the element is drawn beneath the option menu when you select an item from the pop-up menu. The menu contains the following options:

General Background	Sets selection to the background color used for drawing most XIPD interface elements.
General Foreground	Sets selection to the foreground color used for drawing most XIPD interface elements.
Unloaded Status	Sets selection to the appearance of nodes that have not been loaded with a program.
Initialized Status	Sets selection to the appearance of nodes that have been loaded and are stopped at the first executable statement.

Active Status	Sets selection to the appearance of nodes that are executing.
Action Point Status	Sets selection to the appearance of nodes that are stopped at an action point (for example, a breakpoint).
Stopped Status	Sets selection to the appearance of nodes that have stopped execution at a non-breakpoint.
Blocked Status	Sets selection to the appearance of nodes that are stopped when they were blocked waiting to receive a message.
Terminated Status	Sets selection to the appearance of nodes that have terminated.
Mesh Bright	Sets selection to the color used for drawing the highlighted region of the mesh that connects nodes.
Mesh Normal	Sets selection to the color used for drawing the normal region of the mesh that connects nodes.
Mesh Dark	Sets selection to the color used for drawing the shaded region of the mesh that connects nodes.
Source Code Foreground	Sets selection to the color of the text used in the source code window, along with the text foreground color used in some of the report windows.
Source Code Background	Sets selection to the background color used in the source code window, along with the background color used in some of the report windows.

Named color

This is a list of all the colors in the */usr/lib/X11/rgb.txt* file. Selecting a color from this list sets the color for the current object selection (for example, *Active Status*) and updates the red, green, and blue scales within *Color value.*

Use the *Match* button beneath the list to find the closest match in *rgb.txt* to the current color.

Color value

This area contains three scales for adjusting the red / green / blue (RGB) content of the current color and a brightness control for increasing or decreasing the brightness of the current color. When you select an object from the *Set color* list, the RGB scales are set for the object's current color. The scales are also set when you choose a color from the *Named color* list or the palette.

Each scale refers to the amount of saturation for a color and ranges from *0%* to *100%*. As the percentage increases, the amount of the color increases in the current object's composite color.

You can increase the brightness by selecting the up arrow and decrease it by selecting the down arrow.

If you adjust the RGB value to a matching color in the *Named color* list, XIPD scrolls to that color and highlights it.

Palette

The palette area contains a set of colors for you to choose from. Clicking on an entry in the palette makes the selected color the current color. The content of the palette is controlled by a set of radio buttons beneath the palette. The meaning of the buttons are as follows:

Range	Modifies the palette to cover the entire spectrum range.
Wide	Modifies the palette to cover a wide range of colors around the current color.
Narrow	Modifies the palette to cover a narrow range of colors around the current color.

Pattern Names

This field contains a list of names for the various patterns you can use to draw an object. The list is disabled if XIPD cannot create a pattern for the current selection of *Set color for.* (for example, *General Background*). Selecting an item from this list sets the pattern stippling to be used for an object. Note that the *Solid* selection indicates that the object should be drawn solid without any stippling.

Available Patterns

This field contains various patterns you can use to draw an object. Some selections belonging to the *Set color for* option menu can be "stippled" with a pattern (for example, *Active Status*). If a pattern cannot be used for an object, the *Pattern* label is grayed out. Otherwise, you can choose the pattern for an object by selecting the pattern.

The upper-left corner pattern in this field represents no stippling: the object is drawn solid. The upper row continues to the right in patterns that draw less and less of the object on the screen. The lower row uses various straight lines for stippling an object.

Font Names

This area contains text fields for the different fonts used with XIPD. The text field entry should be a font name specification that can be obtained from X clients such as *xlsfonts* or *xfontsel*. A font can either be variable width or fixed width. It is strongly suggested that you use fixed width fonts for elements displaying source code or IPD output. The interface elements you can specify fonts for are the following:

General	Used for most XIPD interface text, such as labels and menu names.
Routine highlighted	Used for routine list entries that are highlighted.
Routine unhighlighted	Used for routine list entries that are not highlighted.
Code window	Uses in the code window. This should be a fixed width font.
Output	Used to display output from IPD, including the output from programs running, send / receive queues, stack frame traceback, system command output, and help text. This should be a fixed width font.

You can use the *Predefined fonts* option menu to load these fields.

Use symbols for node status

This on/off toggle controls whether the run-time status of nodes in the execution viewpoint panel are depicted with specific symbols or with filled circles. Symbols are useful for monochrome displays.

Node size

This sets the height and width of nodes.

Application height and width

This sets the size of the XIPD main panel.

Predefined colors

Predefined colors pops up an option menu containing the name of various color sets you can use with XIPD, as opposed to setting the colors manually. Selecting an item sets the color (and perhaps pattern) for entries under the *Set color for* option menu.

Predefined fonts

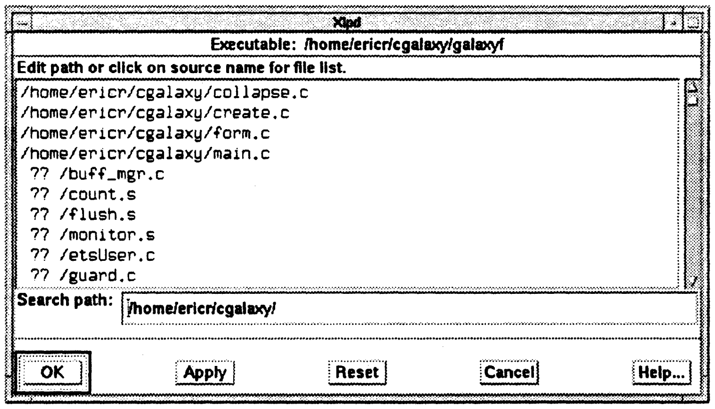
Predefined fonts pops up an option menu containing the names of various font sets. Selecting an item from this menu sets the fields in the *Font names* area.

Dialog Buttons

OK	Amends (or creates) your XIPD resource file, *XIpd*, stored under either the directory contained in the *XAPPLRESDIR* environment variable (if defined) or your home directory. If there are any problems in writing the resources (*XIpd* is write protected, for instance), XIPD displays an error dialog. Otherwise, XIPD displays an information dialog about the resource file changes.
Cancel	Dismisses the preferences dialog and makes no changes to your *XIpd* resource file.
Help	Displays help topic text about the preferences dialog is displayed.

Code Location Dialog

Use the code location dialog to find the source code files for an executable compiled with debug information. The dialog appears when you direct XIPD to look for source files automatically and it can't find them all or when you select *Locate Source Code* for a particular executable. Figure 3-12 shows the code location dialog.



Executable: /home/ericr/cgalaxy/galaxyf

Edit path or click on source name for file list.

```
/home/ericr/cgalaxy/collapse.c
/home/ericr/cgalaxy/create.c
/home/ericr/cgalaxy/form.c
/home/ericr/cgalaxy/main.c
 ?? /buff_mgr.c
 ?? /count.s
 ?? /flush.s
 ?? /monitor.s
 ?? /etsUser.c
 ?? /guard.c
```

Search path: /home/ericr/cgalaxy/

Figure 3-12. Code Location Dialog

Source File List

The code location dialog contains a list of all user source files that belong to the debug-compiled program. If XIPD cannot find a particular source file, it is preceded by two question marks (for example, *??/transform.c*).

When you select a file name, XIPD displays a file selection dialog. The file selection dialog filter is set so only those files that match the file name being searched for (like *transform.c*) are displayed. When the file is found, you can select it and dismiss the file selection dialog. XIPD notes that the selected file is within the given directory.

Search Path

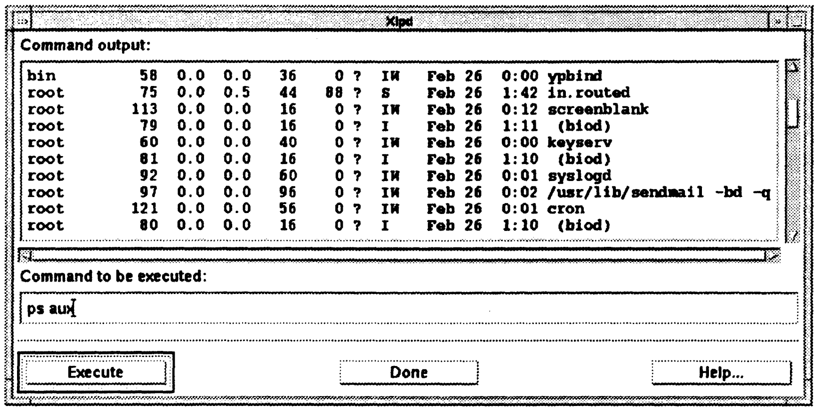
This field contains a blank-delimited list of all the directories to be searched for the debug-compiled program's sources. Order is important. If there are multiple directories in the search path, for example, and a *transform.c* exists in two of the directories, the first *transform.c found* is used to get source information about routines in *transform.c*.

Control Buttons

OK	Uses the *Search Path* value to locate all files that haven't been found yet and dismisses the source locator dialog. This allows multiple files to be found together instead of one at a time.
Apply	Uses the *Search Path* value to locate all files that haven't been found yet. This allows multiple files to be found together instead of one at a time.
Reset	Undoes any changes since the last *Apply*.
Cancel	Undoes any changes since the last *Apply* and dismisses the source locator dialog.
Help	Displays help topic text about the source locator.

System Command Dialog

The system command dialog executes a command that you type in and displays the text output of the command in a scrollable text window. If IPD is running, the command is executed via the IPD **system** command and IPD is blocked until the command has completed execution. Figure 3-13 shows the system command dialog.



```
bin         58  0.0  0.0   36    0 ?  IW   Feb 26  0:00 ypbind
root        75  0.0  0.5   44   88 ?  S    Feb 26  1:42 in.routed
root       113  0.0  0.0   16    0 ?  IW   Feb 26  0:12 screenblank
root        79  0.0  0.0   16    0 ?  I    Feb 26  1:11  (biod)
root        60  0.0  0.0   40    0 ?  IW   Feb 26  0:00 keyserv
root        81  0.0  0.0   16    0 ?  I    Feb 26  1:10  (biod)
root        92  0.0  0.0   60    0 ?  IW   Feb 26  0:01 syslogd
root        97  0.0  0.0   96    0 ?  IW   Feb 26  0:02 /usr/lib/sendmail -bd -q
root       121  0.0  0.0   56    0 ?  IW   Feb 26  0:01 cron
root        80  0.0  0.0   16    0 ?  I    Feb 26  1:10  (biod)
```

```
ps aux
```

Figure 3-13. System Command Dialog

Command output

This is a scrollable text region that contains the output of the last command you executed. You can not edit the text, but you can select it and paste it into another X client.

Command to be executed

Type the system command you want to execute into this text field.

Dialog Buttons

Execute	Sends the contents of the *Command to be executed* field to IPD, prefixed by the IPD **system** command. All output, up to the next IPD prompt, is put into *Command output.*
Done	Dismisses the system command dialog.
Help	Displays help topic text about the system command dialog.

Defaults Menu

The *Defaults* menu allows you to set certain options that control XIPD. The menu is enabled after you select a mesh. Some of the items are grouped together and represent exclusive choices. These grouped items are "radio-button" menu items, meaning that only one of the items in the group can be true. This is noted by a diamond next to the menu item. Groups are separated by a horizontal line. When one of the non-exclusive toggles is set, a small box (as opposed to a diamond) appears next to the item. Figure 3-14 shows the *Defaults* menu.

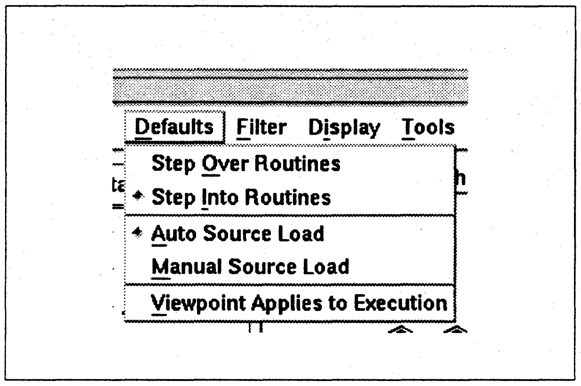


Figure 3-14. Defaults Menu

Step Over Routines

If set, all **step** commands issued to IPD include the **-call** option, meaning that calls to functions are stepped over instead of stepped into. When the function returns, the step is complete.

Step Into Routines

If set, all **step** commands to IPD are such that if stepping on a line with a call to a function compiled with debug information, the function is stepped into and its first executable line becomes the current line of the step.

Auto Source Load

If set, XIPD tries to automatically find a program's source files. When a program compiled for debug is loaded, XIPD obtains the name of the program's source files. If *Auto Source Load* is set, XIPD looks in the program's directory for the source files. If any files are not found, the source location dialog is displayed to ask you to find the rest. If all files are automatically found, however, XIPD assumes it has located the correct sources and doesn't ask you for confirmation.

You can select the *Locate Source Code* menu item to review or override any of the assumptions made by XIPD.

Manual Source Load

If set, XIPD does not try to automatically find a debug-compiled program's sources. If you load a program compiled for debugging, XIPD obtains the names of the source files, but then request that you find the location of the file.

This item is useful if you want to load debug-compiled programs but do not want to have XIPD obtain debug information about the programs.

Viewpoint Applies to Execution

If set, XIPD sends execution commands to only the nodes that are in the viewpoint. Typically, XIPD sends execution commands to all nodes in the mesh to which the command can apply. For example, when you select *Stepped Execution*, all nodes that can step are given a **step** command if *Viewpoint Applies to Execution* is not set. This option is intended for advanced users who want to have more control over the execution of their programs.

Filter Menu

The filter menu is associated with the routines list in the main panel. The settings in the filter menu control what is displayed in the routine list. For example, you can display only routines that have breakpoints set. The filter menu is enabled as long as a program is loaded.

The filter menu is divided into two groups. The first group contains two items that control whether or not filtering is in effect. The second group is used to turn specific filters on and off.

Figure 3-15 shows the filter menu.

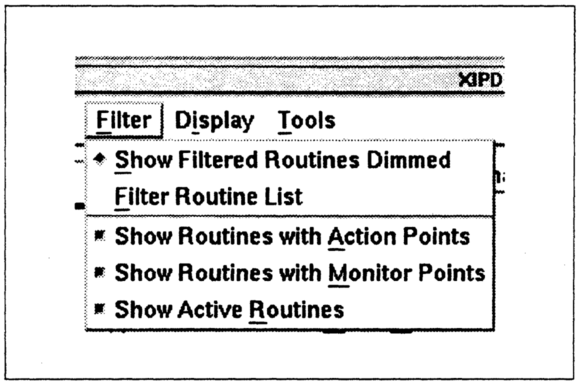

XIPD
Filter
Display
Tools
Show Filtered Routines Dimmed
Filter Routine List
Show Routines with Action Points
Show Routines with Monitor Points
Show Active Routines

Figure 3-15. Filter Menu

Show Filtered Routines Dimmed

When set, routines that would normally be filtered out of the routines list appear in a "dimmed" font that distinguishes them from routines that would not be filtered out.

Filter Routine List

When set, routines that don't meet the filtering criteria are not displayed in the routines list. If no routines meet the filtering criteria, the routines list is empty.

Show Routines with Action Points

If set, routines that have at least one breakpoint set are not filtered out.

Show Routines with Monitor Points

If set, routines with an instrumentation point set are not filtered out.

Show Active Routines

If set, routines containing an execution point are not filtered out.

Display Menu

The display menu displays certain dialogs containing information about the program and its execution. The display menu is enabled as long as a program is loaded into the mesh. Figure 3-16 shows the display menu.

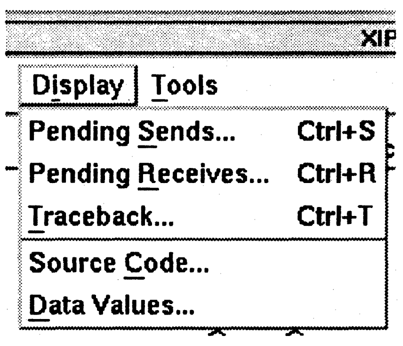


Figure 3-16. Display Menu

Pending Sends

Pending Sends displays the pending sends dialog. If IPD cannot provide pending messages being sent (for example, not all nodes are stopped), you are told that the information cannot be displayed (and are suggested to stop the nodes). The default keyboard accelerator for this item is *Control-S.*

Pending Receives

Pending Receives displays the pending receives dialog. If IPD cannot provide pending receives (for example, not all nodes are stopped), you are told that the information cannot be displayed (and are suggested to stop the nodes). The default keyboard accelerator for this item is *Control-R.*

Traceback

Traceback displays the traceback information dialog. The default keyboard accelerator for this item is *Control-T.*

Source Code

Source Code displays an information dialog that instructs you how to display a routine's source code.

Data Values

Data Values displays an information dialog that instructs you how to display or modify a variable's value.

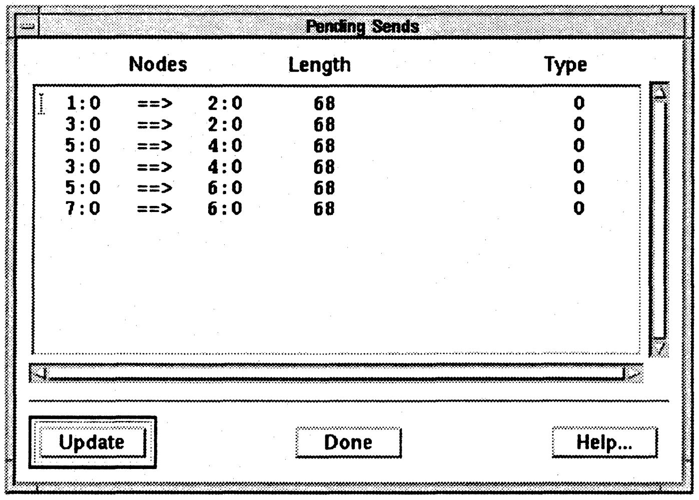
Pending Sends Dialog

The pending sends dialog is displayed when you select the *Pending Sends* menu item. This dialog contains the output of the **msgqueue** command. What it requests is filtered by the nodes that are part of the current viewpoint. If a node is part of the viewpoint, information is asked about the messages it has sent.

The dialog shows which node has sent a message, who that message was to, the length of the message, and the type of the message. XIPD includes process type information with the node identification.

You cannot edit the information, but you can select it and paste it into another client.

Figure 3-17 shows the pending sends dialog.



```
      Nodes             Length              Type
 1:0   ==>   2:0          68                  0
 3:0   ==>   2:0          68                  0
 5:0   ==>   4:0          68                  0
 3:0   ==>   4:0          68                  0
 5:0   ==>   6:0          68                  0
 7:0   ==>   6:0          68                  0
```

Figure 3-17. Pending Sends Dialog

Dialog Buttons

Update	Requests pending messages information again. The message information is updated only when you request so you have a chance to study it. The new request takes into account nodes that have been added to or removed from the viewpoint.
Done	Dismisses the dialog.
Help	Displays help topic text about the dialog.

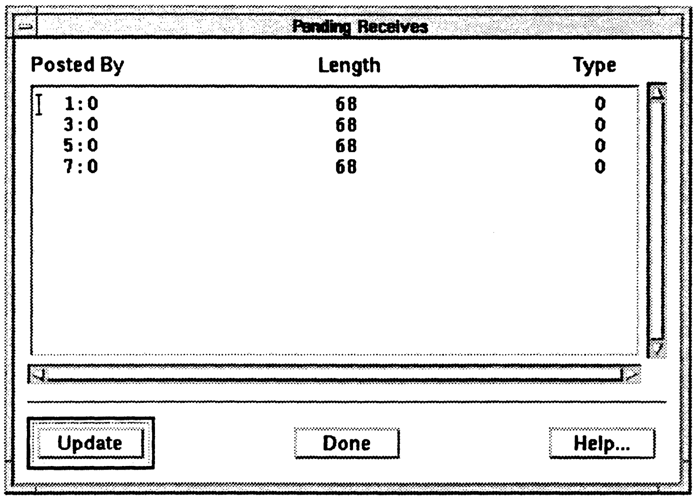
Pending Receives Dialog

The pending receives dialog is displayed when you select the *Pending Receives* menu item. This dialog contains the output of the **recvqueue** command. What it requests is filtered by the nodes that are part of the current viewpoint. If a node is part of the viewpoint, information is asked about the messages it is waiting to receive.

The dialog shows which node is waiting to receive a message, the length of the expected message, and the type of the expected message. XIPD includes process type information with the node identification.

You cannot edit the information, but you can select it and paste it into another client.

Figure 3-18 shows the pending receives dialog.



```
Posted By            Length          Type
 1:0                   68              0
 3:0                   68              0
 5:0                   68              0
 7:0                   68              0
```

Figure 3-18. Pending Receives Dialog

Dialog Buttons

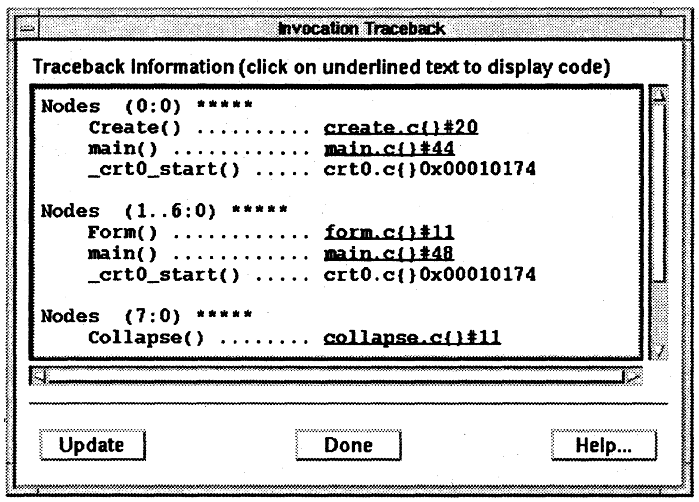
Update	Requests pending receive information again. The message information is updated only when you request so you have a chance to study it. The new request takes into account nodes that have been added to or removed from the viewpoint.
Done	Dismisses the dialog.
Help	Displays help topic text about the dialog.

Traceback Dialog

The traceback dialog is displayed when you select the *Traceback* menu item. This dialog contains the output of the **frame** command. It requests information for the nodes that are part of the current viewpoint. If a node is part of the viewpoint, information is asked about where it currently is executing and how it got there. Nodes that are executing the same point are grouped together. The node's current location is printed first, followed by the invoking routine's location, and so on.

You cannot edit the information, but you can select it and paste it into another client.

Figure 3-19 shows the traceback dialog.


Invocation Traceback
Traceback Information (click on underlined text to display code)
Update
Done
Help...

```
Nodes  (0:0) *****
    Create() .......... create.c{}#20
    main() ............ main.c{}#44
    _crt0_start() ..... crt0.c{}0x00010174

Nodes  (1..6:0) *****
    Form() ............ form.c{}#11
    main() ............ main.c{}#48
    _crt0_start() ..... crt0.c{}0x00010174

Nodes  (7:0) *****
    Collapse() ........ collapse.c{}#11
```

Figure 3-19. Traceback Dialog

Underlined Code Locations

Underlined code locations represent display links to code windows. Clicking on an underlined entry causes XIPD to display the appropriate source code window auto-scrolled to the line number in the underlined entry. XIPD underlines all code locations for which it can display a code window.

Dialog Buttons

Update	Requests frame traceback information again. The message information is updated only when you request so you have a chance to study it. The new request takes into account nodes that have been added to or removed from the viewpoint.
Done	Dismisses the dialog.
Help	Displays help topic text about the dialog.

Tools Menu

This menu contains the names of outside performance tools that you can execute with XIPD and a selection to display the code instrumentation dialog. This menu is not present if you invoked XIPD with the **-noanalysis** command line option. Figure 3-20 shows the tools menu.

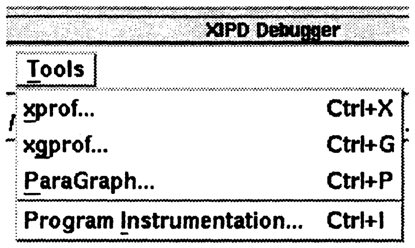


Figure 3-20. Tools Menu

xprof

xprof displays the XProf dialog. You can use this dialog to bring up XProf to analyze **prof** performance output. The default keyboard accelerator for this item is *Control-X.*

xgprof

xgprof displays the XGprof dialog. You can use this dialog to bring up XGprof to analyze **gprof** performance output. The default keyboard accelerator for this item is *Control-G.*

ParaGraph

ParaGraph displays the ParaGraph dialog. You can use this dialog to bring up ParaGraph to analyze ParaGraph performance and trace output. The default keyboard accelerator for this item is *Control-P.*

Instrument Code

Instrument Code displays the code instrumentation dialog which can be used to collect program performance information that can be analyzed for prof, gprof, or ParaGraph. The default keyboard accelerator for this item is *Control-I.*

XProf Dialog

Figure 3-21 shows the XProf dialog.

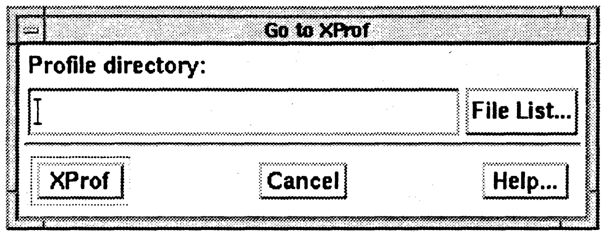


Figure 3-21. XProf Dialog

Profile directory

This field contains the name of the prof output directory (for example, *./mon.out/*). If you don't leave this field blank, it should contain the name of a directory that contains prof-format performance output files.

File List

File List displays the file selection dialog. If you select a directory from this dialog, the directory name is copied into the *Profile directory* field.

Dialog Buttons

XProf	XIPD first checks to be sure the given profile directory exists (if *Profile directory* isn't empty). If it does exist, XIPD dismisses the dialog and invokes XProf with the given options. If the profile directory doesn't exist, XIPD displays an error dialog.
Cancel	Dismisses the XProf dialog and discards any changes to the dialog.
Help	Displays help text for the XProf dialog.

XGprof Dialog

Figure 3-22 shows the XGprof dialog.

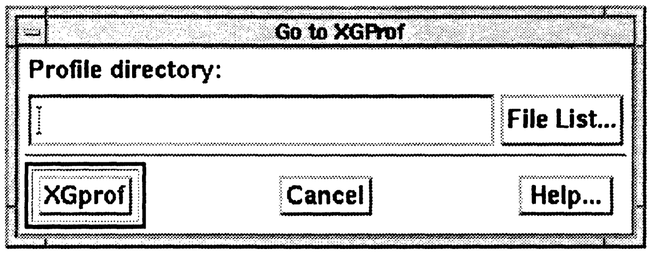


Figure 3-22. XGprof Dialog

Profile directory

This field contains the name of the gprof output directory (for example, *./mon.out/*). If you don't leave this field blank, it should contain the name of a directory that contains gprof-format performance output files.

File List

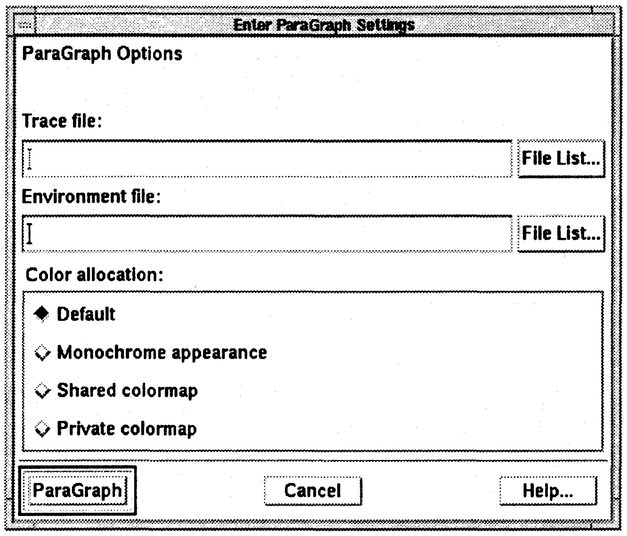
File List displays the file selection dialog. If you select a directory from this dialog, the directory name is copied into the *Profile directory* field.

Dialog Buttons

XGprof	XIPD first checks to be sure the given profile directory exists (if *Profile directory* isn't empty). If it does exist, XIPD dismisses the dialog and invokes XGprof with the given options. If the profile directory doesn't exist, XIPD displays an error dialog.
Cancel	Dismisses the XGprof dialog and discards any changes to the dialog.
Help	Displays help text for the XGprof dialog.

ParaGraph Dialog

Figure 3-23 shows the ParaGraph dialog.



Enter ParaGraph Settings

ParaGraph Options

Trace file:

Environment file:

Color allocation:

◆ Default

◇ Monochrome appearance

◇ Shared colormap

◇ Private colormap

Figure 3-23. Enter ParaGraph Settings Dialog

Trace file

This field contains the name of a ParaGraph trace file. You can leave this field blank.

Environment file

This field contains the name of the window layout file created with ParaGraph's *Save Layout*. You can leave this field blank.

File List

File List displays the file selection dialog. If you select a file from this dialog, the file name is copied into the text field next to the selected *File List* button. For example, if you select the *File List* button next to the *Trace File* field and choose a file with the file selection dialog, the file name is copied into the *Trace File* text field.

Color allocation

Default	ParaGraph defaults to its own color selection scheme.
Monochrome	ParaGraph appears in black and white.
Shared colormap	ParaGraph uses the shared colormap, which might result in ParaGraph not having enough colormap space for it to display all the colors it needs.
Private colormap	ParaGraph uses a private colormap, which gives it enough space for all the colors it needs, but might result in a colormap "flash" when moving in and out of ParaGraph windows.

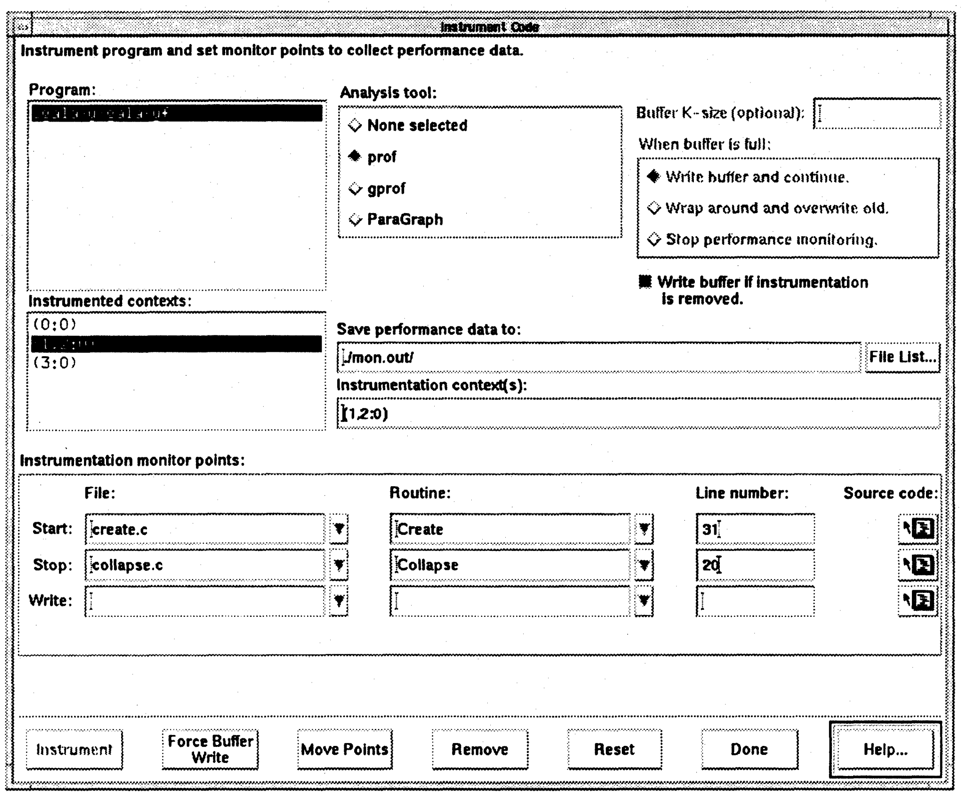
Dialog Buttons

ParaGraph	XIPD first checks to be sure any given file names are valid. If all file name fields are valid or blank, XIPD dismisses the dialog and invokes ParaGraph with the given options. If a given file name is not valid, XIPD displays an error dialog.
Cancel	Dismisses the ParaGraph dialog and discards any changes to the dialog.
Help	Displays help text for the ParaGraph dialog.

Program Instrumentation Dialog

Use the program instrumentation dialog to gather performance information for programs loaded into the mesh. Two things have to be done to the programs in order to collect performance data: their code has to be instrumented so that information can be collected during execution, and monitor points must be placed within the programs to denote where performance data collection should begin and end.

Figure 3-24 shows the program instrumentation dialog



Instrument Code

Instrument program and set monitor points to collect performance data.

Program: [illegible]

Analysis tool: None selected [✓ prof] gprof ParaGraph

Buffer K-size (optional):

When buffer is full: [✓ Write buffer and continue.] Wrap around and overwrite old. Stop performance monitoring.

[✓ Write buffer if instrumentation is removed.]

Instrumented contexts:
(0:0)
[illegible]
(3:0)

Save performance data to: ./mon.out/

Instrumentation context(s): (1,2:0)

Instrumentation monitor points:

	File:	Routine:	Line number:
Start:	create.c	Create	31
Stop:	collapse.c	Collapse	20
Write:			

Figure 3-24. Program Instrumentation Dialog

To instrument a program, you must first select an executable name from the *Program* list, which initially is the only part of the dialog enabled (except for *Done* and *Help*). When you select a program for the first time, the analysis tool radio box is enabled and *None selected* is selected. When you select a tool (such as ParaGraph), the rest of the dialog (except for *Move Points* and *Remove*) is enabled. If there is only one program loaded, it is auto-selected for you. You can set multiple instrumentations for a program, but none can overlap in context.

At a minimum, you must provide an output file name. This is initialized to the default file name for the tool [for example, *./mon.out/* for prof]). In this case, the instrumentation context is set to the current viewpoint, the start point is assumed as the current execution point of the program, and the stop and end points are at the program's exit.

The output file name must be unique for all the instrumentation being performed. This avoids two instrumented programs, one set up for prof and the other for gprof, writing to the same directory.

If you select a program that has already been instrumented from the *Program* list, all contexts that have been instrumented appear in the *Instrumented contexts* list. Selecting an entry from this list updates the interface to reflect the options used for instrumentation. At this point, you can remove instrumentation or revise it by changing the location of the monitor points. To remove instrumentation, you must select the *Remove* button (which is enabled only for program contexts that are successfully instrumented). To change the monitor points, you must select *Move Points*.

Program

The program list is always enabled and contains the names of all the known executables loaded into the mesh. When you select a program from the list, it becomes the selection of the dialog. If the program has not been instrumented yet, only the analysis tool radio box is enabled and *Nothing* is set as the current selection. If the program is instrumented, however, the *Instrumentation contexts* list is loaded with the contexts of the previous instrumentations.

If no programs are currently loaded, this list contains "[Empty]".

Instrumented contexts

This list contains the contexts of any instrumentation that has been performed for a program. Selecting an entry in the list sets the dialog contents to the options used to instrument the program. XIPD disables the *Instrument* button and enables the *Move Points* and *Remove* buttons. You can alter the location of the monitor points or remove the instrumentation.

Analysis tool

This radio box is enabled once you select a program. It contains the names of all the analysis tools.

Once you select a tool and a successful **instrumentation** command is processed, the analysis tool radio box is locked to the tool selected for instrumentation on the specified program context. You cannot select any other tool for this context until you select *Remove.*

None selected	Initial selection. Indicates that the selected program is not instrumented. Most of the features are disabled while this is selected.
prof	Selects instrumentation for prof. The dialog is enabled, the *Buffer size* entry is disabled and cleared, and *Save performance data to* is set to *./mon.out/.*
gprof	Selects instrumentation for gprof. The dialog is enabled, the *Buffer size* entry is disabled and cleared, and *Save performance data to* is set to *./gmon.out/.*
ParaGraph	Selects instrumentation for ParaGraph. The dialog is enabled, the *Buffer size* entry is disabled and cleared, and *Save performance data to* is set to *./pg.trf.*

Buffer K-size

This is a text entry area for setting the optional write buffer size. This field is enabled only when ParaGraph is selected. If the field is left blank, the default buffer size is used.

When buffer is full

This radio box is enabled when you choose ParaGraph as the analysis tool. The contents control what is done when the data collection buffer is full.

Write buffer and continue	When the buffer is full its contents are written to disk and the buffer is emptied to collect new performance data.
Wrap around and overwrite old	When the buffer is full the new information starts to overwrite the old information.
Stop performance monitoring	When the buffer is full performance monitoring stops and no new information is kept.

Write buffer if instrumentation is removed

If this option is set when you remove instrumentation, IPD saves the contents of any buffers that have not already been written to disk. If this option is not set, any unwritten buffer contents are lost.

Save performance data to

This is a required field that is enabled once an analysis tool is selected, at which time this field is initialized with the tool's default output directory and file name.

File List

File List displays a file selection dialog. The file name is placed into the *Save output to* text field. This button is enabled when you select an analysis tool.

Instrumentation context

This field contains the context used for instrumentation. When you select a program, this field is initialized to all nodes within the current main panel viewpoint. You can edit this field to reduce or add contexts. When you select a context from the *Instrumented contexts* list, this field is set to the context used for the previous instrumentation and you can no longer edit the field.

Instrumentation monitor points

Use this area to establishing where monitoring should start, stop, and end. You have the choice of typing in the information, using popup menus to choose information, or clicking on a code window's line to automatically fill in all the information. This area is enabled once you select an analysis tool.

Start The *start* row is used to turn on performance data collection. The allowed blank entries are the following: the row can be completely blank (monitoring begins at the current execution point), you can omit the line number entry (monitoring begins at the start of the supplied function), or you can omit the function name (monitoring begins at the line number in the given file).

Stop The *stop* row is used to turn off performance data collection. Entries in the *stop* row are not valid if the *start* row is empty. The allowed blank entries are the following: the row can be completely blank (monitoring stops at the end of the program), you can omit the line number entry (monitoring stops at the start of the supplied function), or you can omit the function name (monitoring stops at the line number in the given file). The *stop* row, if non-blank, must be different from the *start* row.

Write The *write* row is used to signal the writing of performance information. Entries in the *write* row are not valid if the *stop* row is empty. The allowed blank entries are the following: the row can be completely blank (monitor data is written at the stop point if it is non-blank or the exit of the program if stop is blank), you can omit the line number (monitor data is written at the start of the supplied function), or you can omit the function name (monitor data is written at the line number in the given file).

File The *file* column contains the file name location for a monitor point. You can type in a file name or select the button to the right of the field to display a popup menu of all known file names for the program. The file name popup menu is displayed only if the executable is compiled for debug. Selecting an entry from the popup menu copies the file name from the menu to the file text field.

Routine The *routine* column contains the function name for a monitor point to be set. The monitor point is currently set at the function's entry, not exit, if a line number is not provided. You can type in a function name or select the button to the right of the field to display a popup menu of all known function names for the file name entry for the row. The function name popup menu is displayed only if the executable is compiled for debug and function name information has been collected for the file. Selecting an entry from the popup menu copies the function name from the menu to the function text field. A function field is not valid without an entry in the file field.

Line Number	The line number field sets a monitor point at a specific line. The file name field must have an entry but you can leave the function field blank. If the program is compiled for debug and the function name is provided, XIPD verifies that the line number is executable.
File popup menu	The button for the file popup menu is enabled only when a program is selected for which XIPD has collected debug file information.
Function popup menu	The button for the function popup menu is enabled only when a program is selected for which XIPD has collected debug file information and when the file entry for the row is a valid file name for the program.
Source code window link	The code window link button is to the right of the line number column. Selecting this button enables you to make entries into *File*, *Function*, and *Line* by clicking on a source code line in the code window for a routine.

Dialog Buttons

Instrument	Sends an **instrument -on** command to IPD, based on the entries in the dialog. As much error checking as possible is done before the **instrument** command is issued. This button is enabled once you select an analysis tool.
Force Buffer Write	Sends an **instrument -write** command to IPD. You should use this button if a program abnormally terminates before writing its performance buffer.
Move Points	Sends an **instrument -on** command for a previously instrumented context. It moves the monitor points for a context.
Remove	Sends an **instrument -off** command to IPD for a previously instrumented context. If successful, the instrumented context is removed from the *Instrumented contexts* list. This button is enabled when you select an entry from the *Instrumented contexts* list.
Reset	Undoes all changes made to the dialog's fields.
Done	Dismisses the instrumentation dialog. *Done* is always enabled.
Help	Displays help topic text for the instrumentation dialog.

Help Menu

The *Help* menu, available from the main window, provides various levels of help. This menu is always enabled. Through the *Help* menu, you can obtain context-sensitive help and browse through all of the help topics. Figure 3-25 shows the help menu.

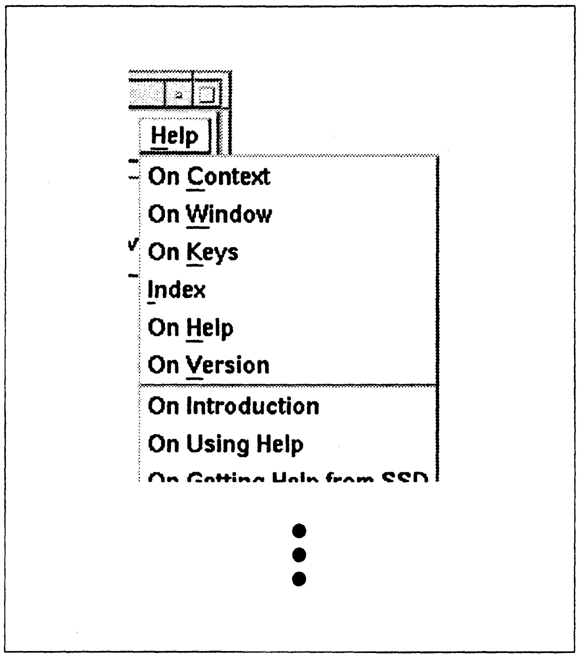


Figure 3-25. Help Menu

On Context

The *On Context* menu item enables context-sensitive help and changes the mouse pointer to a question mark. You can then click on areas of the XIPD interface. If there is a help entry for the selected area (such as the text entry field for the start-up dialog's password field), XIPD displays the help topic text for that feature. You can also obtain context help from the keyboard by pressing the *Help* key (typically mapped to F1) while the keyboard focus is directed to the desired interface feature.

On Window

Displays the help topic text for the main window.

On Keys

Displays help topic text about special key accelerators.

Index

Displays the *Index* dialog. All help topics for XIPD are listed in the *Index* dialog.

On Help

Displays the help topic text that explains how to use all of the aspects of help.

On Version

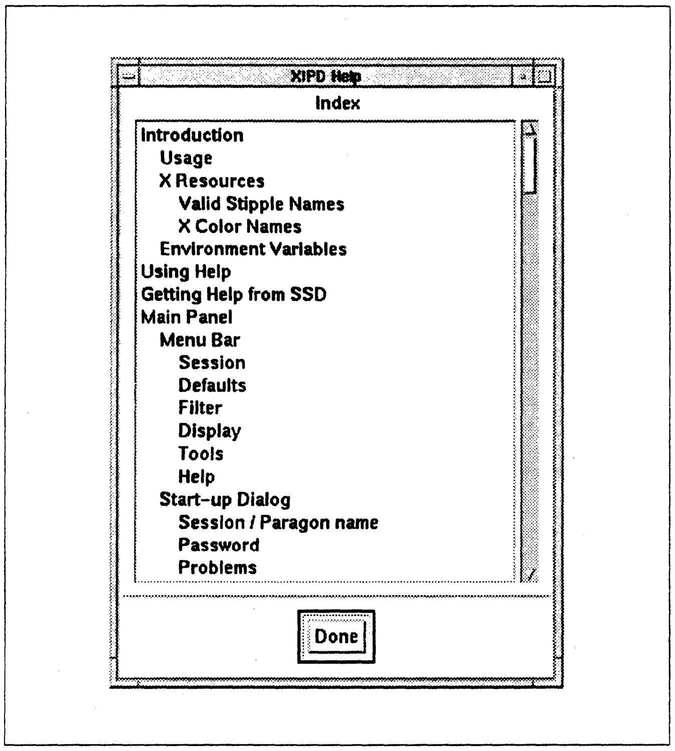
Displays a dialog containing XIPD version information.

Additional Topics

The names of all the major XIPD help topics follow the *On Version* entry on the *Help* menu. When you select a topic, XIPD displays help text for the selected topic.

Help Index

XIPD displays the *Index* dialog when you choose the *Index* menu item from the *Help* menu. The help index contains all the topics and sub-topics of help available for XIPD. Sub-topics are indented underneath major topics. Figure 3-26 shows the help index dialog.



Introduction
  Usage
  X Resources
    Valid Stipple Names
    X Color Names
  Environment Variables
Using Help
Getting Help from SSD
Main Panel
  Menu Bar
    Session
    Defaults
    Filter
    Display
    Tools
    Help
  Start-up Dialog
    Session / Paragon name
    Password
    Problems

Figure 3-26. Help Index Dialog

Help Topics

This field contains a scrollable list of all XIPD help topics. Select a topic from the list to display the help text for that topic.

Done

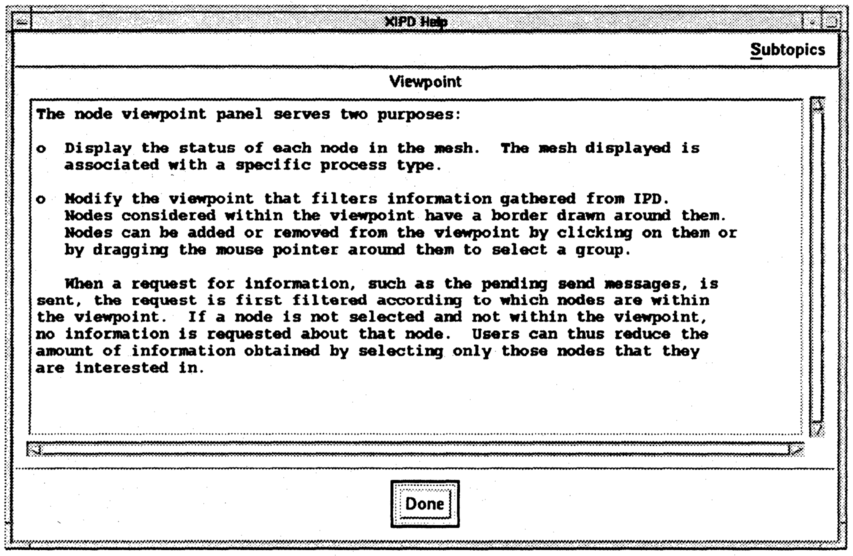
Select *Done* to dismiss the help index dialog.

Help Topic

The help topic dialog appears when you request help for a particular topic. You can select a topic by any of the following:

- selecting the topic in the help *Index* dialog
- selecting a dialog's *Help* button
- using context sensitive help
- selecting a major topic from the *Help* menu.

Figure 3-27 shows the help topic dialog.



```
The node viewpoint panel serves two purposes:

o  Display the status of each node in the mesh.  The mesh displayed is
   associated with a specific process type.

o  Modify the viewpoint that filters information gathered from IPD.
   Nodes considered within the viewpoint have a border drawn around them.
   Nodes can be added or removed from the viewpoint by clicking on them or
   by dragging the mouse pointer around them to select a group.

   When a request for information, such as the pending send messages, is
sent, the request is first filtered according to which nodes are within
the viewpoint.  If a node is not selected and not within the viewpoint,
no information is requested about that node.  Users can thus reduce the
amount of information obtained by selecting only those nodes that they
are interested in.
```

Figure 3-27. Help Topic Dialog

Subtopics

If a topic has subtopics, the help topic dialog contains a *Subtopics* pull-down menu. All of the subtopics are included on the menu. Select an item from the *Subtopics* menu to display help text for the subtopic.

Help Title

The title identifies which topic the help text describes.

Help Text

This field contains the help text for the topic. You can select the text and paste it into other clients.

Items that are underlined in the text represent links to other help topics. Clicking on an underlined text entry displays the help topic dialog for that entry.

Done

Use the *Done* button to dismiss the help topic dialog.

Legend

The main window's legend associates a node's current state with its appearance in the viewpoint panel. The nodes, when displayed, are typically associated with a process type as well. The node's status is therefore a combination of the node and the currently selected mesh process type. A node with multiple process types can have a number of different states, but XIPD displays one node state at a time, based on the currently selected process type. Figure 3-28 shows the main window legend.

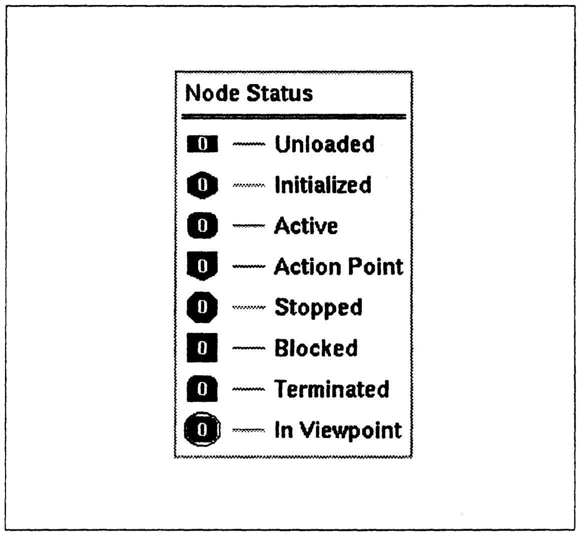


Figure 3-28. Main Window Legend

XIPD uses the status symbols unless the symbol option is turned off, in which case filled circles are used.

The various states are described as follows:

Unloaded	Nothing is loaded into the node.
Initialized	A program is loaded and stopped at its first executable statement, ready to begin execution.
Active	The node is executing.
Action Point	The node has stopped due to an action point, such as a user-set breakpoint or a data based watchpoint.
Stopped	The node is stopped. This could be due to executing a **step** command or selecting the *Stop Execution* button. The node might be stopped at an instruction address.
Blocked	The node is blocked, waiting for a message to be sent to it. It could also be blocked for another reason, such as a call to **gsync**().

Terminated	The program has finished execution.
In Viewpoint	The node is part of the viewpoint.

Mesh Selection

The *Meshes* option menu in the main window changes the viewpoint to a different process type. When you select *Meshes*, a menu containing all of the process types allocated during load or during execution is popped up. When you select a process type, XIPD updates the viewpoint to reflect the node status for the process type.

Execution Controls

The main window execution controls affect all nodes and all process types. You can start execution in either a "stepped" or "continuous" mode. You can also stop any executing nodes, restart execution of all nodes, and unload all of the nodes. Figure 3-29 show the execution control buttons.

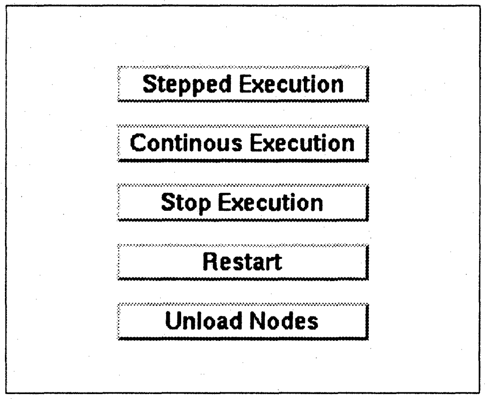


Figure 3-29. Execution Controls

If you want control over which nodes receive execution commands, you must first set *Viewpoint Applies to Execution* in the *Defaults* menu, and select which nodes in the viewpoint should receive execution commands. If *Viewpoint Applies to Execution* is set, only the nodes in the viewpoint receive execution-related commands.

XIPD considers the status of a node before issuing an execution command to the node. For example, XIPD does not issue a **step** command to a terminated or executing node.

XIPD issues its execution commands based on process types. For example, if process types zero, one, and two (0,1,2) are loaded and you choose *Stepped Execution*, XIPD issues step directives first for process type zero, then for process type one, and finally for process type two.

Stepped Execution

When selected, all nodes that can be stepped are given an IPD **step** command. If *Step Over Routines* is set, a **-call** option is issued as well. If a node is not stopped at a debug-compiled source line, a **-instruction** option is issued for the node.

Continuous Execution

When selected, all nodes are instructed to begin continuous execution. If the node state is *Initial*, a **run** command is issued to start execution. If the nodes have already begun execution, however, a **continue** command is issued instead.

Stop Execution

When selected, a **stop** command is issued to all nodes that are currently executing. This control is enabled after executing nodes are detected in the mesh.

Restart

When selected, all loaded nodes are reloaded with the application.

Unload Nodes

When selected, you are asked if you want to unload the nodes. If you do, all loaded nodes are unloaded with a **kill** command and XIPD displays the load dialog.

Partial unloading of the mesh is not supported.

Node Viewpoint Panel

The node viewpoint panel serves two purposes:

- Display the status of each node in the mesh. The mesh displayed is associated with a specific process type.
- Modify the viewpoint that filters information gathered from IPD. Nodes considered within the viewpoint have a border drawn around them. You can add or remove nodes from the viewpoint by clicking on them or by dragging the mouse pointer around them to select a group.

When a request for information, such as the pending send messages, is sent, the request is first filtered according to which nodes are within the viewpoint. If a node is not selected and not within the viewpoint, no information is requested about that node. Users can thus reduce the amount of information obtained by selecting only those nodes that they are interested in.

Adding a Node to the Viewpoint

Clicking on a node with the left mouse button toggles the node in and out of the viewpoint. A border is drawn around a node if it is within the viewpoint. Figure 3-30 shows how a node appears in and out of viewpoint

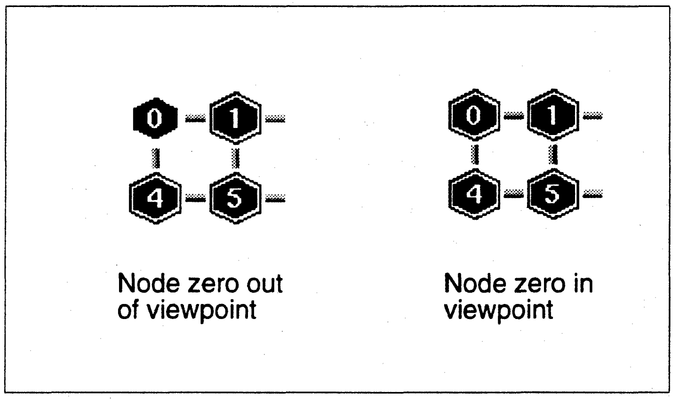
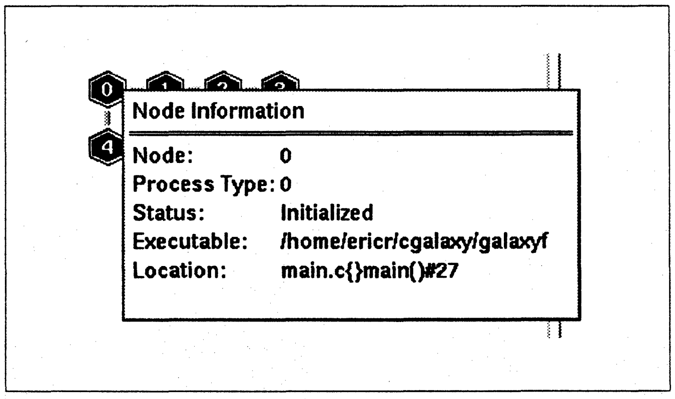


Figure 3-30. Node Viewpoint

Displaying a Node's Status

You can display a pop-up information panel about an individual node by holding down the right mouse button and moving over the node. The information displayed includes the node number, process type, status, what program it is executing, and where it is executing. Figure 3-31 shows a node information panel.



```
Node Information
Node:         0
Process Type: 0
Status:       Initialized
Executable:   /home/ericr/cgalaxy/galaxyf
Location:     main.c{}main()#27
```

Figure 3-31. Node Information Panel

Displaying the Source Code a Node is Executing

Clicking on a node with the middle mouse button displays the code that the node is currently executing. XIPD retrieves the source code for the routine that the node is within and auto-scrolls the source code window to the line the node is executing. In order for XIPD to display this information, the node must be stopped at a debug-compiled source line, and the source code the node is executing must have been located.

Routine List

The routine list displays the routine names for which you can obtain a source code window. The routine's source code window is displayed when you select a routine name. Only the routines that belong to debug-compiled programs and that reside in located source files are displayed in the routine list. For example, a routine won't be listed if a program compiled for debug is loaded but the source file containing the routine is not found.

To the left of each routine name are graphical indications as to whether a break point or monitor point has been set for a routine or if the routine is currently executing. Figure 3-32 shows a routine list.

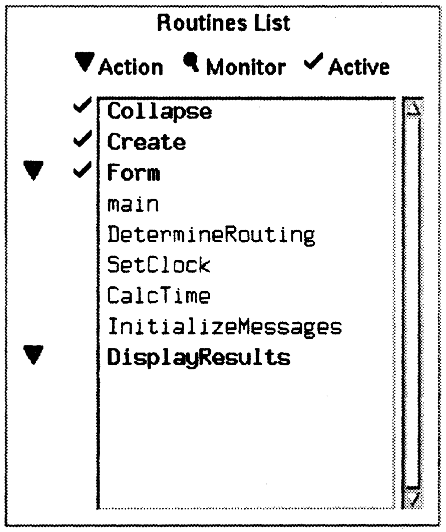
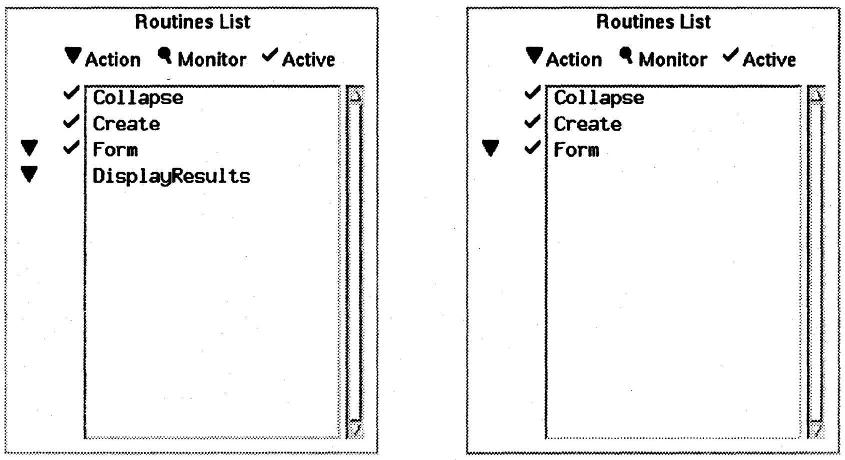


Figure 3-32. Routine List

The routine list in Figure 3-32 shows a number of routine names, four of which (**Collapse**, **Create**, **Form**, and **DisplayResults**) are highlighted. Next to **Form** there's an action point symbol, meaning that at least one action point (such as a breakpoint) has been placed within **Form**. Next to **Collapse**, **Create** and **Form** are active symbols, meaning that lines within these routines are currently being executed.

You can filter the routine list with the settings in the *Filter* menu. All routines are listed if no filtering is in effect (as in Figure 3-32). Routines that are filtered out appear in a dim, normal font, while routines that aren't filtered out appear in a bold font. If filtering is turned on, the routine list shown in Figure 3-32 appears as shown in Figure 3-33.



Routine list filtered to show routines that are active and with action points

Routine list filtered to show only the active routines

Figure 3-33. Filtered Routine List

If you load multiple programs at the same time and some of the routines in the programs have the same name (for example, **main(**)), the name of the file the routine resides in is prefixed to the routine name to help distinguish it from the other routines.

Application Output

The application output area of the main window is provided for programs that write to standard output.

Program output is added to the scrollable text region as it occurs. You can select this text and paste it into other clients.

Control Buttons

The control buttons along the bottom of the main window are used with various dialogs that appear within the main window. These dialogs include:

- Start-up
- Mesh Selection
- Program Load

When XIPD displays a dialog within the main window, it enables the buttons it supports. Not all buttons are supported by every dialog. The buttons have the following functions:

OK

OK implements any changes to the contents of the dialog (after checking for possible errors) and dismisses the dialog. If there is an error condition, the dialog is not dismissed until the error is corrected or you select *Cancel.*

Apply

Apply implements any changes to the contents of the dialog, and the dialog remains open.

Reset

Reset discards all changes since the last *Apply,* and the dialog contents revert to what they were before you made changes. The dialog is not dismissed.

Cancel

Cancel discards all changes since the last *Apply* and dismisses the dialog.

Help

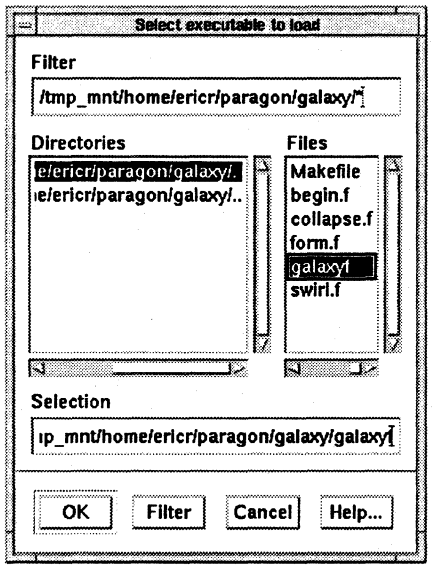
Help displays help topic text about the current dialog. Help about the main window is displayed by default when no dialogs are displayed within the main window. *Help* is always enabled.

Message Area

One-line status messages not critical enough to warrant their own alert dialog are displayed in this field. When you dismiss a working dialog, the dialog text is copied here.

File Selection

You can display the file selection dialog in situations where you are asked to type in a file name or from the source locator. Figure 3-34 shows the file selection dialog.



```
/tmp_mnt/home/ericr/paragon/galaxy/*
```

Directories:

```
e/ericr/paragon/galaxy/.
ie/ericr/paragon/galaxy/..
```

Files:

```
Makefile
begin.f
collapse.f
form.f
galaxy.f
swirl.f
```

Selection:

```
ıp_mnt/home/ericr/paragon/galaxy/galaxy
```

Figure 3-34. File Selection Dialog

Filter

This text field contains the filter for displaying files. All directories are displayed, but you can filter out files to control the length of the list. You can use standard UNIX shell expressions in the filter field. For example, to display all C-language source code files in a directory, you could enter the following in the filter field:

*/home/username/src/**.c

The *.c expression causes XIPD to match all files that end in .c. Therefore, all files that end with the extension *.c* are displayed.

If you want to list all the files in a directory, make sure the filter ends with an asterisk.

Directories

All directories in the current directory (specified by the filter) are listed in this field. Selecting a directory makes it the current directory. Directories are never filtered out.

Files

All files in the current directory that pass through the filter are listed in this field. It is possible for this list to be empty (represented by a [ ] in the file list) if no files match the filter, or if the current directory is empty. Make sure the filter ends with an asterisk if you want to list all the files.

If you select a file, the file is listed in the file selection dialog and XIPD automatically invokes *OK*, dismissing the dialog.

Selection

The *Selection* field contains the current file selection. If you select *OK*, this file is returned by the file selection dialog to the dialog from which it was displayed. For example, if the file selection dialog was displayed from the load program dialog, pressing *OK* copies the name of the file in the *Selection* field to the load program dialog's text field as the name of the executable to load.

Dialog Buttons

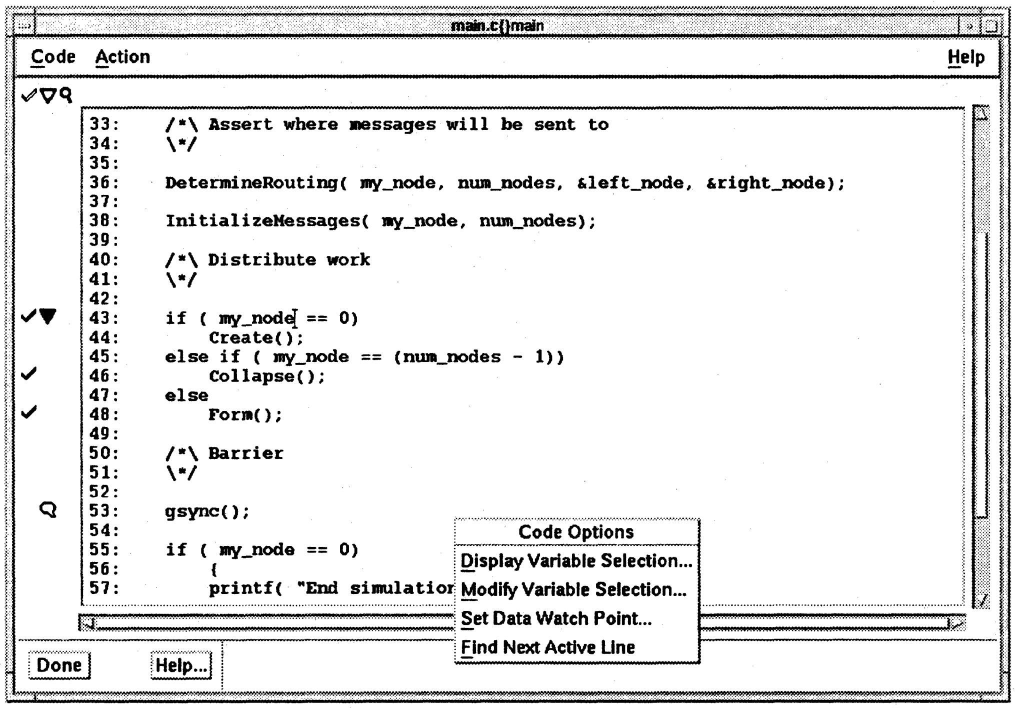
OK	Passes the entry in *Selection* to the originating dialog and dismisses the file selection dialog.
Filter	Obtains a new file list, and possibly a new directory list.
Cancel	Dismisses the file selection dialog and passes nothing to the originating dialog.
Help	Displays help topic text about the file selection dialog.

Code Window

The code window contains the source code for a specified routine. XIPD displays a code window when you select a routine name in the main window's routine list, when you click on an underlined entry in the traceback dialog, or when you click on a specific node with the middle mouse button. XIPD obtains this source code from IPD, with line numbers. XIPD has stored information about which lines are executable to be used to verify user set breakpoints.

The code window contains a menu bar, the text of the routine, and an icon strip that identifies executing lines, lines with breakpoints, and lines with monitor points. There is also a pop-up menu within the text region of the code window, displayed by holding down the third mouse button.

Figure 3-35 shows the code window.



```
/*\ Assert where messages will be sent to
\*/

DetermineRouting( my_node, num_nodes, &left_node, &right_node);

InitializeMessages( my_node, num_nodes);

/*\ Distribute work
\*/

if ( my_node == 0)
    Create();
else if ( my_node == (num_nodes - 1))
    Collapse();
else
    Form();

/*\ Barrier
\*/

gsync();

if ( my_node == 0)
    {
    printf( "End simulatio
```

Figure 3-35. Code Window

The contents of the code window change if you invoke XIPD with the **-nodebug** or the **-noanalysis** command line options.

Title

The title of the code window contains the name of the routine, preceded by the name of the file that contains the routine.

Code Menu

Figure 3-36 shows the code menu.

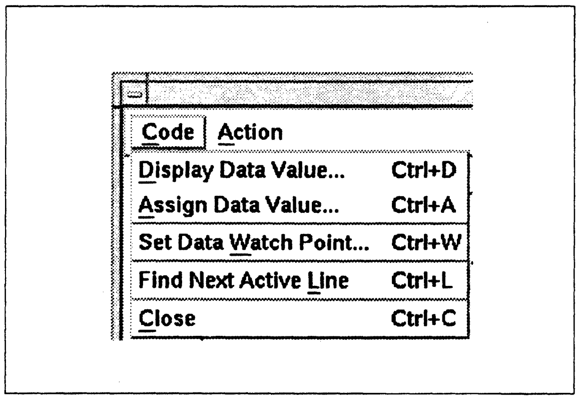

Code
Action
Display Data Value... Ctrl+D
Assign Data Value... Ctrl+A
Set Data Watch Point... Ctrl+W
Find Next Active Line Ctrl+L
Close Ctrl+C

Figure 3-36. Code Menu

Display Data Value	Displays the data display dialog for the current code window.
Assign Data Value	Displays the data modification dialog for the current code window.
Set Data Watch Point	Displays the data watchpoint dialog for the current routine's program.
Find Active Lines	Scrolls the code window to the next executing line, if there is one. When an active line is found, the line is highlighted and a message is displayed about which nodes are executing the line. More than one line can be executing, and subsequent clicks on *Find Active Lines* advance to the next one. If the last executing line is being displayed, the next click on *Find Active Lines* scrolls back to the first executing line.
Close	Dismisses the code window.

Action Menu

The action menu selects the kind of action point to be set for a line. Figure 3-37 shows the action menu.

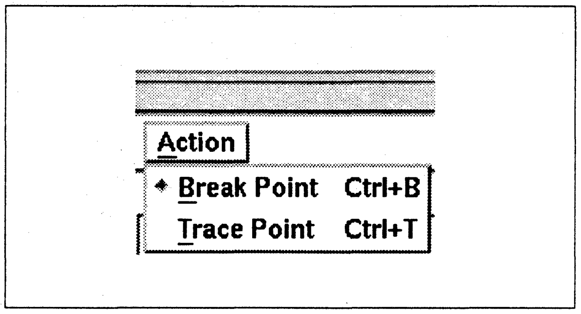


Figure 3-37. Action Menu

You can only select one item at a time from this menu. The action points currently supported are:

Break Point	Stops execution when line is about to be executed.
Trace Point	Prints message when line is executed.

Help Menu

The code window help menu is an abbreviated form of the main window help menu. Figure 3-38 shows the code window help menu.

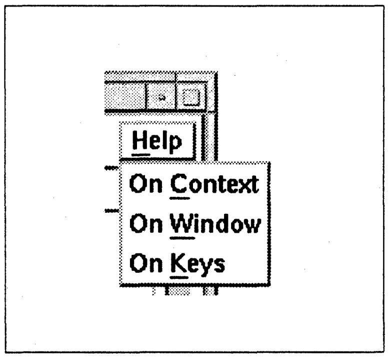


Figure 3-38. Code Window Help Menu

On Context	The mouse pointer turns into a question mark and help topic text, if it exists, is displayed about the portion of the feature you click on.
On Window	Displays help topic text about the code window.
On Keys	Displays help topic text about special accelerator keys.

Code Lines

The code lines are contained in a scrollable text region. You can not edit this text, but you can select it and paste it into another X client. If the data display, modification, or watch point dialog is brought up, any selected text is scanned for the first possible variable name and this is put into the variable name field for the dialog.

Code that is compiled with both debug information and optimization can cause some code reorganization, and not all normally executable lines are noted as executable.

Icons

Next to each line is an area for three icons: executing (a check mark), action point (depends on the type of action point), and monitor point (a magnifying glass). When the mouse pointer moves into the column for one of the icons, the pointer changes into a "ghost" version of the icons that appear in the column.

The executing icon appears next to a line when one or more nodes are executing the line. You can search for executable lines by selecting the *Find Active Lines* button.

Clicking within the action point column either sets or removes an action point. If no action point is set for the line next to where you click, a new action point is set. If the pointer is over an existing action point and you click, the action point is removed. Action points can only be set for executable lines. XIPD searches forward for the next executable line if you try to set an action point for an unexecutable line.

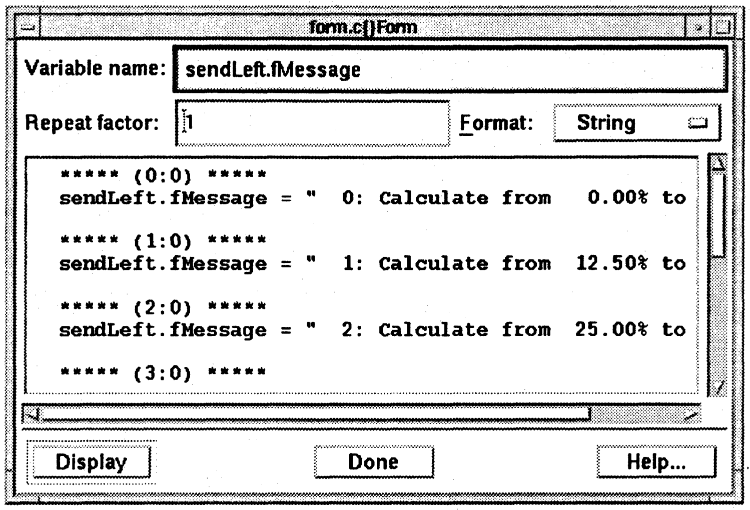
When you set a monitor point for a source code line (with the *Instrument Code* dialog), a monitor icon appears next to the line.

Dialog Buttons

Done	Dismisses the code window.
Help	Displays help topic text for the code window.

Data Display

The data display dialog obtains the value of a variable, which might be active on more than one node. A data display dialog is associated with a particular code window. Figure 3-39 shows the data display dialog.



Variable name: sendLeft.fMessage

Repeat factor: 1 Format: String

```
***** (0:0) *****
sendLeft.fMessage = "  0: Calculate from   0.00% to
***** (1:0) *****
sendLeft.fMessage = "  1: Calculate from  12.50% to
***** (2:0) *****
sendLeft.fMessage = "  2: Calculate from  25.00% to
***** (3:0) *****
```

Figure 3-39. Data Display Dialog

Title

The dialog title is the name of the code window with which it is associated. This name is composed of the routine name prefixed with the file in which the routine resides.

Variable Name

The name of the variable to be displayed. Expressions are not allowed.

Repeat Factor

The repeat factor is for displaying arrays. For scalar variables (such as integers), the repeat factor should be one. If the repeat factor is greater than one, and the variable is an array, XIPD displays multiple entries of the array starting at the index specified in *Variable Name*. The number of array members displayed is equal to the repeat factor.

For example, using the repeat factor, you could display the first five elements of an array. If *Variable Name* contains *Output[0]* and the repeat factor is *5*, the first five values of the C-language array *Output* would be displayed.

Format

This is a pop-up option menu containing the various display formats. The first entry is *Default*, which should be sufficient for most variables. An exception is C-language character strings. The default display prints each array member individually. If *Format* is set to *String*, C-language character strings are printed as quoted strings.

Value Display

The value of the variable is displayed in this field. The value is printed for each node for which the variable is active. If some of the nodes have the same value, the value is printed once, and all the nodes having the same value are listed together.

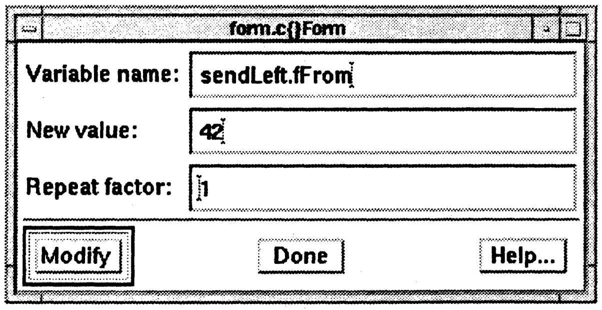
You can select the displayed text and copy it into another client.

Dialog Buttons

Display	XIPD asks IPD for the value of the variable. The request is only for those nodes in the viewpoint. The dialog is disabled until IPD finishes responding.
Done	Dismisses the data display dialog.
Help	Displays help topic text about the dialog.

Data Modification

The data modification dialog changes the value of a variable. You provide a variable name and a new value, and all nodes in the viewpoint for which the variable is active have their variable changed to the given value. The data modification dialog is associated with the code window from which it was displayed. Figure 3-40 shows the data modification dialog.



Variable name: sendLeft.fFrom
New value: 42
Repeat factor: 1

Figure 3-40. Data Modification Dialog

Title

The dialog title is the name of the code window with which it is associated. This name is composed of the routine name prefixed with the file in which the routine resides.

Variable Name

The name of the variable to be modified.

New Value

The new value for the variable.

Repeat Factor

The repeat factor is for modifying arrays. For scalar variables (such as integers), the repeat factor should be one. If the repeat factor is greater than one, and the variable is an array, XIPD modifies entries of the array starting at the index specified in *Variable Name.* The number of array members modified is equal to the repeat factor. All of the array members are set to the given value.

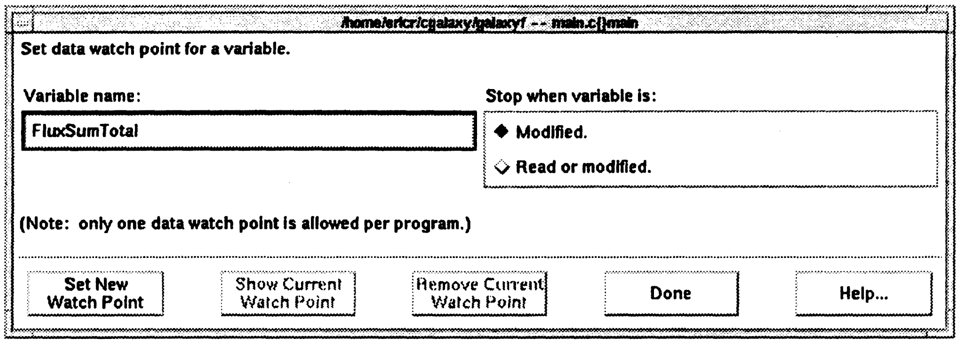
For example, using the repeat factor, you could modify the first five elements of an array. If *Variable Name* contains *Output[0]*, *New Value* contains *6*, and the repeat factor is *5*, the first five values of the C-language array *Output* would be modified to six.

Dialog Buttons

Modify	IPD modifies the given variable to the value specified in *New Value*. The dialog is disabled until IPD completes modifying the variable.
Done	Dismisses the data modification dialog.
Help	Displays help topic text about the dialog.

Data Watch Point

The data watch point dialog sets a watchpoint for a particular variable. You can instruct the debugger to halt a program's node when a program reads from and/or writes to a particular variable. Only one watchpoint can be in effect at a time for each node/process type combination. XIPD further restricts this by allowing just one data watchpoint per program. When a watch point is set for a variable, it is set for all appropriate programs loaded in the mesh. Figure 3-41 shows the data watch point dialog.


/home/ericr/cgalaxy/galaxyf -- main.c{}main

Set data watch point for a variable.

Variable name: FluxSumTotal

Stop when variable is: [✓ Modified.] Read or modified.

(Note: only one data watch point is allowed per program.)

Figure 3-41. Data Watchpoint Dialog

Variable Name

The name of the variable that a watchpoint should be set for or that a watch point has already been set for.

Stop when variable is

This radio-button box defines the stop condition. There are two exclusive choices:

Modified	A node is halted whenever it performs an operation that modifies the contents of the variable.
Read or Modified	A node is halted whenever it performs an operation that reads from the variable or modifies the contents of the variable.

Dialog Buttons

Set New Watch Point	Sets a new watchpoint for the program associated with the code window. This button is disabled if the program currently has a watchpoint set for it.
Show Current Watch Point	Sets *Variable name* and *Stop when variable is* to the current watchpoint settings. This button is disabled if no watchpoint is currently in effect for the program associated with the code window.
Remove Current Watch Point	Removes the current watchpoint set for the program associated with the code window. This button is enabled only if the program has a watchpoint in effect.

Done	Dismisses the data watchpoint dialog.
Help	Displays help topic text about the dialog.

Configuring XIPD

You can configure XIPD by using an X resource file. You can make the resource entries in your *.Xdefaults* file (which resides in your home directory) or in a file named *XIpd* (located either in your home directory or in a directory specified by the *XAPPLRESDIR* environment variable). The entries in the *.Xdefaults* file take precedence over the *XIpd* file entries.

Along with the resources corresponding to the standard X toolkit command line options, the following XIPD application resources are provided to configure XIPD:

XIpd.appForeground	A color name that defines the default foreground color for interface elements. This resource and *XIpd*foreground* should be the same.
XIpd.appBackground	A color name that defines the default background color for interface elements. This resource and *XIpd*background* should be the same.
XIpd*foreground	A color name that defines the default foreground color for all XIPD interface elements.
XIpd*background	A color name that defines the default background color for all XIPD interface elements.
XIpd.unloadedStatusForeground	A color name that defines the foreground color for unloaded nodes in the mesh.
XIpd.initializedStatusForeground	A color name that defines the foreground color for loaded but not yet executing nodes in the mesh.
XIpd.activeStatusForeground	A color name that defines the foreground color for nodes selected for load or for executing nodes.
XIpd.breakpointStatusForeground	A color name that defines the foreground color for nodes in the mesh at a break point.
XIpd.stoppedStatusForeground	A color name that defines the foreground color for nodes in the mesh that are stopped.

XIpd.blockedStatusForeground	A color name that defines the foreground color for nodes in the mesh that are blocked waiting for a message.
XIpd.terminatedStatusForeground	A color name that defines the foreground color for nodes in the mesh that have terminated execution.
XIpd.unloadedStatusStipple	A stipple name that defines the stippling pattern used when drawing an unloaded status node.
XIpd.initializedStatusStipple	A stipple name that defines the stippling pattern used when drawing an initialized status node.
XIpd.activeStatusStipple	A stipple name that defines the stippling pattern used when drawing an active status node.
XIpd.breakpointStatusStipple	A stipple name that defines the stippling pattern used when drawing a break point status node.
XIpd.stoppedStatusStipple	A stipple name that defines the stippling pattern used when drawing a stopped status node.
XIpd.blockedStatusStipple	A stipple name that defines the stippling pattern used when drawing a blocked status node.
XIpd.terminatedStatusStipple	A stipple name that defines the stippling pattern used when drawing a terminated status node.
XIpd.brightColor	A color name that defines the color for the bright part of the mesh lines drawn behind the nodes. The mesh is drawn with *XIpd.normColor* and *XIpd.darkColor* as well.
XIpd.normColor	A color name that defines the normal color for the mesh lines drawn behind the nodes. The mesh is drawn with *XIpd.brightColor* and *XIpd.darkColor* as well.
XIpd.darkColor	A color name that defines the color for the dark (shadow) part of the mesh drawn lines behind the nodes. The mesh is drawn with *XIpd.brightColor* and *XIpd.normColor* as well.
XIpd.sourceForeground	A color name that defines the color used for rendering the program text in the source code window.
XIpd.sourceBackground	A color name that defines the color used for the background of the program text in the source code window.

XIpd.breakpointForeground	A color name that defines the color used for the break point icons. The *Preferences* dialog sets this the same color as break point status.
XIpd.monitorpointForeground	A color name that defines the color used for the monitor point icons. The *Preferences* dialog sets this the same color as unloaded status.
XIpd.activeForeground	A color name that defines the color used for active code icons. The *Preferences* dialog sets this the same color as active status.
XIpd.generalFontName	A font name that defines the font used to render most of XIPD's text.
XIpd.codeWindowFontName	A font name that defines the font used to render the text in a source code window. *This should be a fixed width font.*
XIpd.outputFontName	A font name that defines the font used to render the text in windows that display output either IPD (like the pending send queue) or within the console input/output panel. *This should be a fixed width font.*
XIpd.monospaceFontName	A font name that defines the font for displaying highlighted routine names in the main panel's routine list.
XIpd.monospaceDimFontName	A font name that defines the font for displaying non-highlighted (dimmed) routine names in the main window's routine list.
XIpd.showSymbols	A boolean (True / False) value. If this resource is *True*, XIPD uses symbols as well as color to display the status of nodes in the viewpoint panel. If *False*, XIPD uses filled circles.
XIpd.nodeHeight	An integer that defines the height, in pixels, that XIPD draws nodes (this and *XIpd.nodeWidth* should be set to the same value).
XIpd.nodeWidth	An integer that defines the width, in pixels, that XIPD draws nodes (this and *XIpd.nodeHeight* should be set to the same value).
XIpd.geometry	A geometry string that controls the initial size and placement of XIPD.

You can use an optional stippling pattern for drawing the nodes. If you use this pattern, it must be one of the following strings (or else XIPD ignores the resource setting):

dotsVeryDark	Almost all the pixels drawn.
dotsDark	Most of the pixels drawn.
dotsNorm	Half the pixels drawn.
dotsLight	Some of the pixels drawn.
dotsVeryLight	Very few of the pixels drawn.
lineDiagHatch	Hatched diagonal lines.
lineDiagLeft	Diagonal lines slanted to the left.
lineDiagRight	Diagonal lines slanted to the right.
lineHatch	Hatched horizontal and vertical lines.
lneHoriz	Horizontal lines.
lineVert	Vertical lines.

Default Configuration

The following are the XIPD default resources settings:

XIpd.appForeground	white (color) black (grayscale / monochrome)
XIpd.appBackground	#2f689e (color) #b5b5b5 (grayscale) white (monochrome)
XIpd*foreground	white (color) black (grayscale / monochrome)
XIpd*background	#2f689e (color) #b5b5b5 (grayscale) white (monochrome)
XIpd.unloadedStatusForeground	#acacac (color) #cccccc (grayscale) black (monochrome)
XIpd.initializedStatusForeground	#40e9f8 (color) #ffffff (grayscale) black (monochrome)
XIpd.activeStatusForeground	#00da00 (color) #e0e0e0 (grayscale) black (monochrome)

XIpd.breakpointStatusForeground	#e6f637 (color) #494949 (grayscale) black (monochrome)
XIpd.stoppedStatusForeground	#e94723 (color) #595959 (grayscale) black (monochrome)
XIpd.blockedStatusForeground	#f1b000 (color) black (grayscale) black (monochrome)
XIpd.terminatedStatusForeground	black (color) #898989 (grayscale) black (monochrome)
XIpd.brightColor	white (color) white (grayscale) black (monochrome)
XIpd.normColor	#bfbfbf (color / grayscale) black (monochrome)
XIpd.darkColor	grey50 (color / grayscale) black (monochrome)
XIpd.sourceForeground	white (color) black (grayscale / monochrome)
XIpd.sourceBackground	#2f689e (color) #b5b5b5 (grayscale) white (monochrome)
XIpd.unloadedStatusStipple	*Null-String* (color / grayscale) "dotsNorm" (monochrome)
XIpd.initializedStatusStipple	*Null-String* (color / grayscale) "dotsVeryDark" (monochrome)
XIpd.activeStatusStipple	*Null-String* (color / grayscale) "lineDiagLeft" (monochrome)
XIpd.breakpointStatusStipple	*Null-String* (color / grayscale) "lineHatch" (monochrome)
XIpd.stoppedStatusStipple	*Null-String* (color / grayscale / monochrome)
XIpd.blockedStatusStipple	*Null-String* (color / grayscale / monochrome)
XIpd.terminatedStatusStipple	*Null-String* (color / grayscale) "lineHoriz" (monochrome)
XIpd.generalFontName	variable
XIpd.codeWindowFontName	fixed
XIpd.outputFontName	7x13

XIpd.monospaceFontName	-*-fixed-bold-*-semicondensed-*-13-*-*-*-*-*-*-*
XIpd.monospaceDimFontName	-*-fixed-medium-*-semicondensed-*-13-*-*-*-*-*-*-*
XIpd.showSymbols	True
XIpd.nodeHeight	32
XIpd.nodeWidth	32
XIpd.geometry	900x600+20+20

Null-String means that the stipple resource is not defined, and by default, an empty, null string is used to indicate that no pattern should be used to stipple the node's status.

Environment Variables

The following environment variables, if defined, are used directly by XIPD:

NX_DFLT_PART	Name of the default partition that XIPD selects automatically within the mesh configuration dialog. If not defined (or invalid), XIPD selects the *.compute* partition.
XAPPLRESDIR	XIPD looks in this directory for the resource file *XIpd* when you modify XIPD resource settings through the *Preferences* dialog. If not defined, XIPD looks in your home directory.
XIPDHOME	Alternative home directory for XIPD session files. If not defined, XIPD stores the session files in your home directory.

Program Profiling: prof and gprof 4

This chapter describe **prof** and **gprof**, the execution profilers for the Paragon system. Both **prof** and **gprof** analyze profile files produced with the Interactive Parallel Debugger (IPD) **instrument** command. **prof** produces a simple execution profile, and **gprof** produces an execution profile, call-graph profile, and cycle listing.

A program creates a profile file if it has been loaded under IPD and processed with the **instrument** command. Only programs that execute the *write_location* point of the **instrument** command cause a data file to be written. Without this data file, you can not use **prof** or **gprof** to profile an application. For a complete description of the IPD **instrument** command, refer to the *Paragon*™ *Interactive Parallel Debugger Reference Manual*

prof

When you process an application with the IPD **instrument** command, IPD produces a profile data file. **prof** creates an execution profile by correlating the symbol table in the executable file with the profile file created for that application by IPD.

For each external text symbol, **prof** lists the percentage of time spent executing between the address of that symbol and the address of the next symbol, together with the number of times the function was called and the average number of milliseconds per call.

Invoking prof

To invoke **prof**, use the **prof** command as follows:

prof [*sort_option*] [*address_option*] [*display_options*] [**-m** *profile_file*] [*executable_file*] *sort_option* can be one of the following:

-t	Sort output lines by decreasing percentage of total time (default).
-c	Sort output lines by decreasing number of calls.
-a	Sort output lines by increasing symbol address.
-n	Sort output lines by symbol name.

address_option can be one of the following

-o	Display each symbol address in octal.
-x	Display each symbol address in hexadecimal.

display_options can be any of the following:

-g	Include non-global symbols (static functions).
-z	Include all symbols in the profile range, even if associated with zero number of calls and zero time.
-h	Suppress the heading normally displayed on the report. (This is useful if the report is to be processed further.)
-s	Display a summary of several of the monitoring parameters and statistics on the standard error output.

The other **prof** command line parameters are defined as follows:

-m *profile_file* Use the file specified by *profile_file* as the input profile file. By default, the file with the lowest *node:ptype* pair from the directory *mon.out* is used. The data files in *mon.out* are named with the following form:

executable_name.pid.node.ptype

where *pid* is the process id, *node* is the number of the node on which the process is running, and *ptype* is the last process type the process had before the performance data was written.

executable_file Correlate the symbol table in the file specified by *executable_file* with the *profile_file* specified by the **-m** argument. If you omit this argument, **prof** uses the symbol table in the executable file *a.out*.

Sample prof Output

This section shows **prof** output for the following simple program named *hello*.

```
int
main()
{
    printf("hello\n");
}
```

The **prof** command line and the sample output are as follows:

```
paragon> prof hello
Executable: /home/auld/hello
prof data: /home/auld/mon.out/hello.459166.0.-459166
 %Time Seconds Cumsecs  #Calls  msec/call  Name
 100.0    0.01    0.01                     _memcpy
   0.0    0.00    0.01       1        0.   _main
   0.0    0.00    0.01       6        0.   _NCisshift
   0.0    0.00    0.01       6        0.   _NLchrlen
   0.0    0.00    0.01       1        0.   __doprnt
   0.0    0.00    0.01       1        0.   _exit
   0.0    0.00    0.01       1        0.   __xflsbuf
   0.0    0.00    0.01       1        0.   _fwrite
   0.0    0.00    0.01       1        0.   _memchr
   0.0    0.00    0.01       1        0.   _printf
   0.0    0.00    0.01       2        0.   _profil
   0.0    0.00    0.01       1        0.   _write
```

The columns in the output represent the following:

%Time	The percentage of the total running time of the program used by the function.
Seconds	The number of seconds accounted for by the function.
Cumsecs	A running total of the number of seconds accounted for by the function and the functions listed above it in the output.
#Calls	The number of times the function was invoked.
msec/call	The average number of milliseconds spent in the function per call.
Name	The name of the function.

gprof

When you process an application with the IPD **instrument** command, IPD produces a profile data file. **gprof** creates a profile by correlating the symbol table in the executable file with the profile file created for that application by IPD. **gprof** produces a flat profile, a call graph profile, and a cycle listing. The flat profile is similar to the profile provided by **prof**. This profile provides total execution times and call counts for each function in the program, sorted by decreasing time.

To develop the call graph profile, **gprof** builds a call graph and discovers cycles in the call graph. Execution times are propagated along the edges of the call graph, and calls into a cycle are made to share the time of the cycle. The call graph profile provides a listing of the functions sorted according to the time they represent. This includes the time of their call graph descendents. The call graph profile lists the direct call graph children of each function below the entry for that function and lists how their times are propagated to the function. Above each function entry is a display that shows how the time of the function and its descendents are propagated to the function's direct call graph parents.

Finally, **gprof** provides a listing of the cycles, with an entry for each cycle as a whole and a listing of the members of the cycle and their contribution to the time and call counts of the cycle.

Invoking gprof

To invoke **gprof**, use the **gprof** command as follows:

gprof [*display_options*] [*routine_options*] ... [*object_file* [*profile_file*] ...]

display_options can be any of the following:

-a	Suppresses printing statically declared functions. If this option is given, all relevant information about the static function (for example, time samples, calls to other functions, and calls from other functions) belongs to the function loaded just before the static function in the *a.out* file.
-b	Provides brief output. Suppresses descriptions of each field in the profile.
-s	Produces a profile file *gmon.sum* which represents the sum of the profile information in all the specified profile files. This summary profile file may be given to subsequent executions of **gprof** (probably also with a **-s** option to accumulate profile data across several runs of an application).
-z	Displays routines which have zero usage (as indicated by call counts and accumulated time).

routine_options can be any of the following:

-e *function_name* Suppresses printing the graph profile entry for routine *function_name* and all its descendants, unless they have other ancestors that are not suppressed. More than one **-e** option can be given. Only one *function_name* can be given with each option.

-E *function_name* Suppresses printing the graph profile entry for routine *function_name* and its descendants (unless they have other ancestors that are not suppressed), and also excludes the time spent in *function_name* and its descendants from the total and percentage time computations. More than one **-E** option can be given.

-f *function_name* Prints the graph profile entry only for routine *function_name* and its descendants. More than one **-f** option can be given. Only one *function_name* can be given with each **-f** option. The **-f** option overrides the **-e** option.

-F *function_name* Prints the graph profile entry only for routine *function_name* and its descendants, and also uses only the times of the printed routines in total time and percentage computations. More than one **-F** option can be given. Only one *function_name* can be given with each **-F** option. The **-F** option overrides the **-E** option.

The *function_name* specified for the **-e**, **-f**, **-E** and **-F** options must be a valid COFF symbol (without the leading underscore generated by the compiler). For C symbols, this is identical to the name of the function. Since the Paragon Fortran compilers generate a trailing underscore for Fortran routines, it is necessary to add this underscore to the routine name. For example, **-e** f_ , for the Fortran routine f().

The other **gprof** *command line parameters are defined as follows:*

object_file Specifies the name of the executable used by **gprof** to extract symbol table information. The object file should match the executable that produced the profile file being analyzed. The default is *a.out*.

profile_file Specifies the name of a directory containing multiple profile files generated by a parallel application run, or the name of a single profile file. The default is *gmon.out*. **gprof** checks whether this argument specifies a directory or a file. In the case of a directory, **gprof** expects to find an *INFO* file that contains information about the application that generated the profile directory. The *INFO* file has the following format:

```
Controlling process: executable_name pid_value
    pid          node          ptype        Executable
    xxxxxxx      xxxxx   xxxx   full_path_of_executable
    xxxxxxx      xxxxx   xxxx   full_path_of_executable
    .....
    .....
```

gprof expects the directory to contain one profile file for every process listed in the *INFO* file. The individual data files are named *executable_name.pid.node.ptype* where pid is the process id, ptype is the process type and node is the node number as given in the *INFO* file. By default, **gprof** chooses the lowest (node:ptype) pair data file for the specified object file as the profile file to use. To view **gprof** output on other (node:ptype) pairs, the specific *executable_name.pid.node.ptype* data file must be specified as the profile file.

Multiple profile files may be given but only the first profile file specified is assumed to be a directory containing multiple files. For example, to produce a summary profile of all the data files for the application binary *tst*, you could use the following command:

```
gprof -s tst gmon.out/tst*
```

Sample gprof Output

This section shows gprof output for the following simple program named hello.

```
int
main()
{
   printf("hello\n");
}
```

The gprof command line and the sample output follow. In the output, each section of the analysis is preceded by a description of the terms used in the analysis.

```
paragon> gprof hello
```

call graph profile:

The sum of self and descendents is the major sort for this listing.

function entries:

index	the index of the function in the call graph listing, as an aid to locating it (see below).
%time	the percentage of the total time of the program accounted for by this function and its descendents.

self	the number of seconds spent in this function itself.
descendents	the number of seconds spent in the descendents of this function on behalf of this function.
called	the number of times this function is called (other than recursive calls).
self	the number of times this function calls itself recursively.
name	the name of the function, with an indication of its membership in a cycle, if any.
index	the index of the function in the call graph listing, as an aid to locating it. parent listings:
self*	the number of seconds of this function's self time which is due to calls from this parent.
descendents*	the number of seconds of this function's descendent time which is due to calls from this parent.
called**	the number of times this function is called by this parent. This is the numerator of the fraction which divides up the function's time to its parents.
total*	the number of times this function was called by all of its parents. This is the denominator of the propagation fraction.
parents	the name of this parent, with an indication of the parent's membership in a cycle, if any.

index	the index of this parent in the call graph listing, as an aid in locating it.
	children listings:
self*	the number of seconds of this child's self time which is due to being called by this function.
descendent*	the number of seconds of this child's descendent's time which is due to being called by this function.
called**	the number of times this child is called by this function. This is the numerator of the propagation fraction for this child.
total*	the number of times this child is called by all functions. This is the denominator of the propagation fraction.
children	the name of this child, and an indication of its membership in a cycle, if any.
index	the index of this child in the call graph listing, as an aid to locating it.

* these fields are omitted for parents (or children) in the same cycle as the function. If the function (or child) is a member of a cycle, the propagated times and propagation denominator represent the self time and descendent time of the cycle as a whole.

** static-only parents and children are indicated by a call count of 0.

cycle listings:
the cycle as a whole is listed with the same fields as a function entry. Below it are listed the members of the cycle, and their contributions to the time and call counts of the cycle.

```
granularity: each sample hit covers 4 byte(s) no time propagated

                                  called/total     parents
index  %time     self descendents  called+self    name        index
                                  called/total     children
```

```
                 0.00    0.00    6/6        _NLchrlen [116]
[115]   0.0      0.00    0.00    6      _NCisshift [115]
-----------------------------------------------
                 0.00    0.00    6/6        __doprnt [118]
[116]   0.0      0.00    0.00    6      _NLchrlen [116]
                 0.00    0.00    6/6        _NCisshift [115]
-----------------------------------------------
                 0.00    0.00    1/2        __PMMStart [159]
                 0.00    0.00    1/2        __PMMStop [160]
[117]   0.0      0.00    0.00    2      _profil [117]
-----------------------------------------------
                 0.00    0.00    1/1        _printf [124]
[118]   0.0      0.00    0.00    1      __doprnt [118]
                 0.00    0.00    6/6        _NLchrlen [116]
                 0.00    0.00    1/1        _fwrite [121]
-----------------------------------------------
                 0.00    0.00    1/1        _fwrite [121]
[119]   0.0      0.00    0.00    1      __xflsbuf [119]
                 0.00    0.00    1/1        _write [125]
-----------------------------------------------
                 0.00    0.00    1/1        __crt0_start [193]
[120]   0.0      0.00    0.00    1      _exit [120]
-----------------------------------------------
                 0.00    0.00    1/1        __doprnt [118]
[121]   0.0      0.00    0.00    1      _fwrite [121]
                 0.00    0.00    1/1        _memchr [123]
                 0.00    0.00    1/1        __xflsbuf [119]
-----------------------------------------------
                 0.00    0.00    1/1        __crt0_start [193]
[122]   0.0      0.00    0.00    1      _main [122]
                 0.00    0.00    1/1        _printf [124]
-----------------------------------------------
                 0.00    0.00    1/1        _fwrite [121]
[123]   0.0      0.00    0.00    1      _memchr [123]
-----------------------------------------------
                 0.00    0.00    1/1        _main [122]
[124]   0.0      0.00    0.00    1      _printf [124]
                 0.00    0.00    1/1        __doprnt [118]
-----------------------------------------------
                 0.00    0.00    1/1        __xflsbuf [119]
[125]   0.0      0.00    0.00    1      _write [125]
-----------------------------------------------

flat profile:

 %          the percentage of the total running time of the
time        program used by this function.
```

```
cumulative     a running sum of the number of seconds accounted
 seconds       for by this function and those listed above it.

 self          the number of seconds accounted for by this
seconds        function alone. This is the major sort for this
               listing.

calls          the number of times this function was invoked, if
               this function is profiled, else blank.

 self          the average number of milliseconds spent in this
ms/call        function per call, if this function is profiled,
               else blank.

 total         the average number of milliseconds spent in this
ms/call        function and its descendents per call, if this
               function is profiled, else blank.

name           the name of the function. This is the minor sort
               for this listing. The index shows the location of
               the function in the gprof listing. If the index is
               in parenthesis it shows where it would appear in
               the gprof listing if it were to be printed.

granularity: each sample hit covers 4 byte(s) no time accumulated

  %   cumulative  self              self    total
 time  seconds   seconds    calls  ms/call  ms/call  name
  0.0    0.00     0.00        6     0.00     0.00    _NCisshift [115]
  0.0    0.00     0.00        6     0.00     0.00    _NLchrlen [116]
  0.0    0.00     0.00        2     0.00     0.00    _profil [117]
  0.0    0.00     0.00        1     0.00     0.00    __doprnt [118]
  0.0    0.00     0.00        1     0.00     0.00    __xflsbuf [119]
  0.0    0.00     0.00        1     0.00     0.00    _exit [120]
  0.0    0.00     0.00        1     0.00     0.00    _fwrite [121]
  0.0    0.00     0.00        1     0.00     0.00    _main [122]
  0.0    0.00     0.00        1     0.00     0.00    _memchr [123]
  0.0    0.00     0.00        1     0.00     0.00    _printf [124]
  0.0    0.00     0.00        1     0.00     0.00    _write [125]

Index by function name

[115] _NCisshift      [120] _exit         [124] _printf
[116] _NLchrlen       [121] _fwrite       [117] _profil
[118] __doprnt        [122] _main         [125] _write
[119] __xflsbuf       [123] _memchr
```

XProf 5

This chapter describes XProf, a graphical front end to **prof**. For a description of **prof**, refer to Chapter 4, *Performance Monitoring*. Using **prof** from the command line can be complicated in the Paragon environment, since a directory of files (instead of just a single file) can be created when you profile an application. For parallel applications, each process running on the mesh that is prepared for profiling (through instrumentation with IPD) generates a profile output file. XProf helps you select files by displaying the following information for each file in the profile directory:

- Node number
- Process type
- Process ID
- Executable name

When you select a file entry, XProf executes **prof** on that file and displays the output in a separate, scrollable window.

XProf provides dialogs to assist you in selecting a profile output directory, saving the output of **prof** to a text file, and setting the runtime options for **prof**.

XProf also provides online, context-sensitive help.

You can use XProf as a stand-alone graphical interface or from ParAide, the graphical front end to the Paragon toolset. The XIPD graphical front end to the Interactive Parallel Debugger also provides a connection to XProf. XIPD users can select the source code of a program to instrument for profiling, run the program, and invoke XProf to examine the profile results. Chapter 1 describes ParAide, and Chapter 3 describes XIPD.

Using XProf

This section describes how to use XProf to examine profile output of parallel applications on your Paragon system. Detailed information about the individual windows, menus, dialogs, and commands can be found in the section *Windows, Menus, and Commands.*

Invoking XProf

To invoke XProf use the **xprof** command as follows:

xprof [*sort_option*] [*address_option*] [*display_option*] [-m *profdir*] [*X Toolkit parameters*]

The parameters to the **xprof** command are described as follows:

sort_option can be one of the following:

-t	Sort report entries by decreasing percentage of total time. This is the default.
-c	Sort report entries by decreasing call count.
-a	Sort report entries by increasing symbol address.
-n	Sort report entries alphabetically by symbol name.

The *sort_option* options have the following precedence, from highest to lowest: **-n**, **-a**, **-c**, **-t**. If you specify more than one *sort_option* on the XProf command line, XProf uses the one with the highest precedence.

address_option can be one of the following:

-o	Display symbol addresses in octal base.
-x	Display symbol addresses in hexadecimal base. This is the default.

The *address_option* options have the following precedence, from highest to lowest: **-x**, **-o**. If you specify more than one *address_option* on the XProf command line, XProf uses the one with the highest precedence.

display_option can be any of the following:

-g	Include local (static-declared) symbols.
-z	Include all symbols, even those associated with zero number of calls and zero amount of time.
-h	Suppress the report header.
-s	Display a summary.
-rows *numrows*	Display the profile output directory's files within *numrows* rows. Ten rows is the default.

The other parameters are described as follows:

-m *profdir*	Specify *profdir* as the path name of the profile output directory to be initially read. This path name can either be an absolute path or a path relative to the current directory. *profdir* defaults to *mon.out.*
X Toolkit parameters	The standard parameters supported by the X Toolkit (refer to the *X Toolkit Intrinsics Programming Manual*). You can specify these parameters on the XProf command line.

Choosing a Profile Output Directory

You can choose a profile output directory by any of the following:

- Having the default directory name (typically *mon.out*) in your current directory
- Passing the directory name to XProf as a command line argument.
- Selecting the directory is selected via the *Open...* dialog of the *File* menu.

Once you choose a profile output directory, XProf reads a special descriptive file in the directory and displays the profile file names.

Setting prof Run-Time Options

Before you execute **prof**, you can set preferences for the format of the output with the *prof Settings* dialog of the *Options* menu. These runtime settings include the following:

- Sorting methods (for example, sort by symbol name)
- Address notation (hexadecimal or octal)
- Miscellaneous display options (for example, include local routines in the report)

You can also specify format settings in an *XProf* resource file. This allows you to automatically set the output format each time you invoke XProf. For a description of how to use a resource file to configure XProf, refer to the section *Configuring XProf.*

Selecting a Profile Output File

When you choose a profile output directory, the directory contents are displayed within a scrolling list, sorted by node number, process type, process ID, and executable name. When you select one of the entries in the list, XProf creates a **prof** output window and executes **prof**, using the specified runtime settings. The output of **prof** (standard error and standard output) appears in this output window.

A **prof** output window is available for each list entry. If you select a list entry while its prof output window is somewhere on the screen, the window is brought to the front of the screen.

Examining prof Output

The **prof** output window consists of two pull-down menus and a scrollable text region containing the output of **prof**. You can not edit the text, but you can select it and paste it into another client.

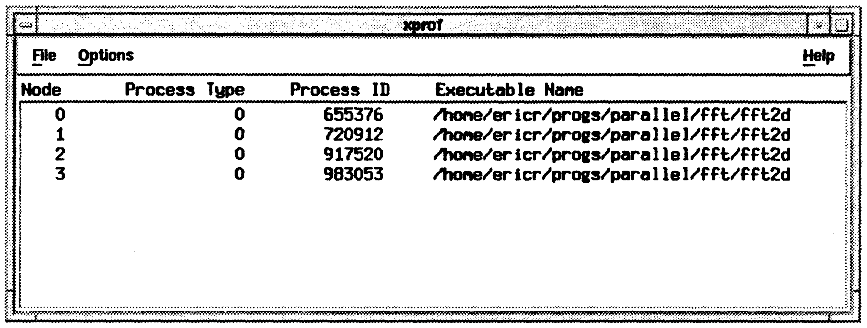
If you change the runtime settings for **prof** and want to generate new output for the **prof** output window, you can choose the *Re-execute prof* menu item. You can also save the **prof** output into a text file with the file dialog brought up when you select the *Save As...* menu item.

Windows, Menus, and Commands

XProf has two windows: the main window and the output window. The following sections describe these windows and the menus and commands you can invoke from the windows.

Main Window

The main window of XProf contains menus and a list of the profile directory contents. Figure 5-1 shows the XProf main window.



```
Node   Process Type   Process ID   Executable Name
  0              0       655376    /home/ericr/progs/parallel/fft/fft2d
  1              0       720912    /home/ericr/progs/parallel/fft/fft2d
  2              0       917520    /home/ericr/progs/parallel/fft/fft2d
  3              0       983053    /home/ericr/progs/parallel/fft/fft2d
```

Figure 5-1. XProf Main Window

The menu bar across the top of the main window contains the menus: *File, Options,* and *Help*. The inner region of the main window contains a list of the profile information contained within the chosen profile output directory. The list is empty if you have not selected a directory.

File Menu

The *File* menu contains items to open a profile output directory and to exit XProf. You can pull down the *File* menu by selecting the menu with the pointer or by pressing the menu's mnemonic key **F**. Figure 5-2 shows the *File* menu

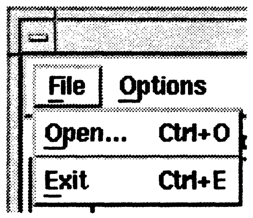


Figure 5-2. File Menu

Open...

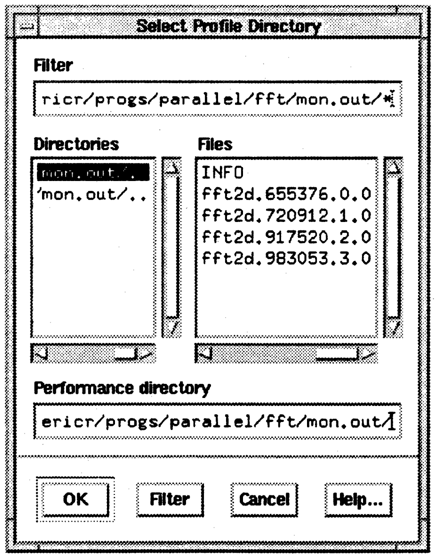
The *Open...* menu item displays the *Select Profile Directory* dialog. This dialog is always enabled. If the open is successful, the contents of the main window's list are replaced with the contents of the selected directory. The default keyboard accelerator for this item is *Control-O*.

Exit

The *Exit* menu item quits XProf and closes all open windows. XProf displays a question dialog to ask you if you really want to quit. The default keyboard accelerator for *Exit* is *Control-E*.

Select Profile Directory

XProf displays the *Select Profile Directory* dialog when you select the *Open...* menu item from the main window. Figure 5-3 shows the *Select Profile Directory* dialog.



```
ricr/progs/parallel/fft/mon.out/*
```

Directories:

```
mon.out/.
mon.out/..
```

Files:

```
INFO
fft2d.655376.0.0
fft2d.720912.1.0
fft2d.917520.2.0
fft2d.983053.3.0
```

Performance directory:

```
ericr/progs/parallel/fft/mon.out/
```

Figure 5-3. Select Profile Directory Dialog

To select a profile directory, select the directory name and then select *OK*. You can also select a file within the directory and XProf selects the profile directory containing the file.

The following sections describe the features of the *Select Profile Directory* dialog.

Filter

This text field contains the filter for displaying files. All profile directories are displayed, but you can filter out files to control the length of the file list. If you want all of the file names to be displayed, the filter should end with an asterisk ('*'), as in the following example:

/home/*username*/src/mon.out/*

If you specify an invalid directory, the dialog issues a beep to the console and does not change the file or directory list.

Directories

All directories in the current directory (specified by the filter) are listed in this field. Selecting a directory or pressing *Filter* makes the directory the current directory. Directories are never filtered out of the directory list.

Files

All files in the current directory that pass through the filter are listed here. It is possible for this list to be empty (represented by a [ ] in the file list) if no files match the filter or if the directory is empty. Make the filter end with a '/*' to see all files in the directory.

If you select a file, it becomes the selection of the file dialog and *OK* is automatically invoked, dismissing the file selection dialog.

Performance directory

This field shows the current file selection. If you select *OK*, XProf uses the file name contained in this field as the profile output directory name.

Dialog Buttons

OK	When you select *OK*, XProf checks the entry in the *Performance directory* field to be sure it is a valid profile output directory (or a file within a valid directory). If the directory is valid, the contents appear in the main window's contents list.
Filter	XProf uses the entry in the *Filter* field to obtain a new file list and a new directory list if the filter has been changed to a different directory.
Cancel	When selected, XProf dismisses the dialog.
Help...	Displays help topic text about using the *Select Profile Directory* dialog.

Options Menu

The *Options* menu, available from the main window, is used to select which command arguments XProf uses when it invokes **prof**. Select the *prof Settings...* menu item to set your preferences. Figure 5-4 shows the *Options* menu.

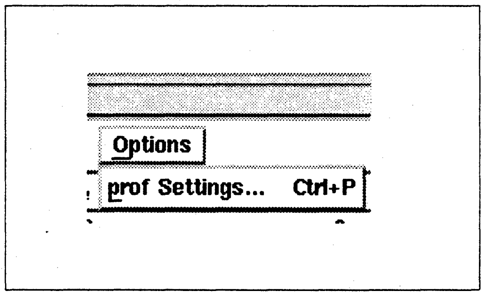


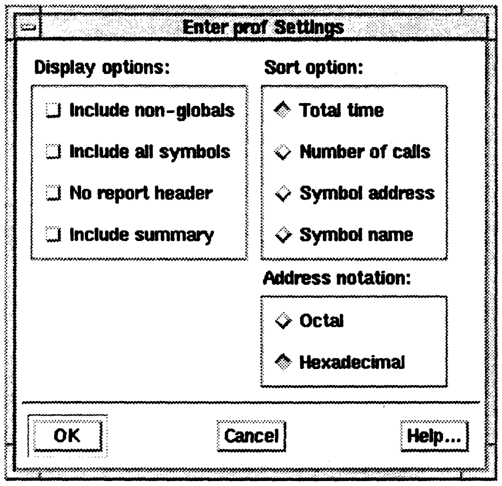
Figure 5-4. Options Menu

prof Settings...

This selection displays the *prof Settings* dialog. Changes to this dialog affect subsequent **prof** generated reports. The default keyboard accelerator for this item is *Control-P*.

Enter prof Settings

The **prof** utility has a number of command-line options. You can set these options with the *Enter prof Settings* dialog. Figure 5-5 shows the *Enter prof Settings* dialog.



Enter prof Settings

Display options:

- Include non-globals
- Include all symbols
- No report header
- Include summary

Sort option:

[✓ Total time] Number of calls | Symbol address | Symbol name

Address notation:

Octal [✓ Hexadecimal]

Figure 5-5. Enter prof Settings Dialog

The *Enter prof Settings* dialog has its own defaults. You can change these defaults with command line options to XProf (as described in the section *Invoking XProf*) or with entries in the XProf resource file. The XProf resource file is described in the section *Configuring XProf.*

The following sections describe the features of the *Enter prof Settings* dialog.

Display Options

You can select more than one display option. The display options are described as follows:

Include non-globals	Include local (static declared) routines in the **prof** output.
Include all symbols	Include all symbols, including symbols with zero-use, in the **prof** output.
No report header	Omit the **prof** output header.
Include summary	Include a summary at the end of the **prof** output.

Sort Option

You can only select one sort option. The sort options control how the **prof** output is sorted. The sort options are described as follows:

Total time	Sort by decreasing amount of time spent in a routine.
Number of calls	Sort by decreasing number of calls to a routine.
Symbol address	Sort by ascending symbol address.
Symbol name	Sort alphabetically by symbol names.

Address Notation

You can only select one address notation option. The address notation options control the display format for symbol addresses. The address notation options are described as follows:

Octal	Print addresses in base eight.
Hexadecimal	Print addresses in base sixteen.

Dialog Buttons

OK	Dismisses the settings dialog. Changes to the dialog are noted and are used the next time **prof** is executed. To make changes persistent to the next execution of XProf, you must set the options in your XProf resource file (refer to the section *Configuring XProf*).
Cancel	Dismisses the settings dialog. Any changes to the dialog are discarded.
Help...	Displays help text for the *Enter prof Settings* dialog.

Help Menu

The *Help* menu, available from the main window, provides various levels of help. This menu is always enabled. Through the *Help* menu, you can obtain context sensitive help and browse through all of the help topics. Figure 5-6 shows the *Help* menu.

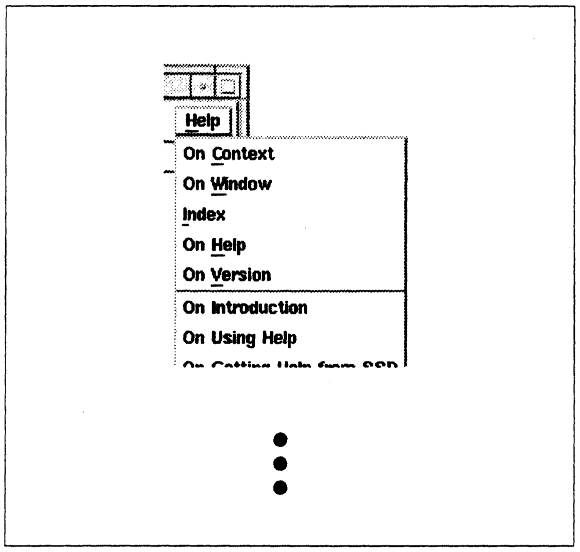


Figure 5-6. Help Menu

On Context

The *On Context* menu item enables context-sensitive help and changes the mouse pointer to a question mark. You can then click on areas of the XProf interface. If there is a help entry for the selected area (such as the list area of the main window), XProf displays the help topic text for that feature. You can also obtain context help from the keyboard by pressing the *Help* key (typically mapped to F1) while the keyboard focus is directed to the desired interface feature.

On Window

Displays the help topic text for the main window.

Index

Displays the *Index* dialog. All help topics for XProf are listed in the *Index* dialog.

On Help

Displays the help topic text that explains how to use all of the aspects of help.

On Version

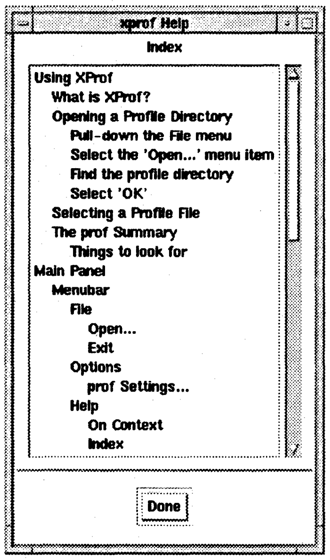
Displays a dialog containing XProf version information.

Additional Topics

The names of all the major XProf help topics follow the *On Version* entry on the *Help* menu. When you select a topic, XProf displays help text for the selected topic.

Help Index

XProf displays the *Index* dialog when you choose the *Index* menu item from the *Help* menu. The help index contains all the topics and sub-topics of help available for XProf. Sub-topics are indented underneath major topics. Figure 5-7 shows the *Index* dialog.



```
Using XProf
   What is XProf?
   Opening a Profile Directory
      Pull-down the File menu
      Select the 'Open...' menu item
      Find the profile directory
      Select 'OK'
   Selecting a Profile File
   The prof Summary
      Things to look for
Main Panel
   Menubar
      File
         Open...
         Exit
      Options
         prof Settings...
      Help
         On Context
         Index
```

Figure 5-7. Help Index Dialog

Help Topics

This field contains a scrollable list of all XProf help topics. Select a topic from the list to display the help text for that topic.

Done

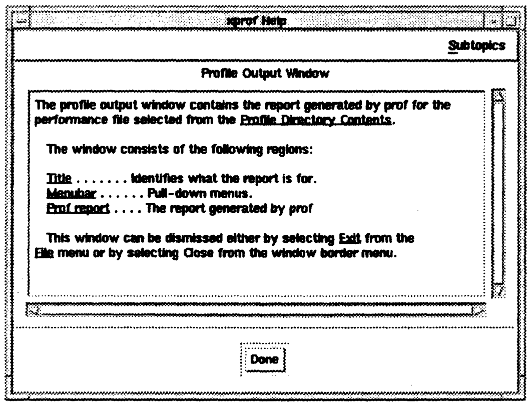
Select *Done* to dismiss the help index dialog.

Help Topic

The help topic dialog appears when you request help for a particular topic. You can select a topic by any of the following:

- selecting the topic in the help *Index* dialog
- selecting a dialog's *Help* button
- using context sensitive help
- selecting a major topic from the *Help* menu.

Figure 5-8 shows a help topic dialog.



Profile Output Window

The profile output window contains the report generated by prof for the performance file selected from the Profile Directory Contents.

The window consists of the following regions:

```
Title ....... Identifies what the report is for.
Menubar ...... Pull-down menus.
Prof report .... The report generated by prof
```

This window can be dismissed either by selecting Exit from the File menu or by selecting Close from the window border menu.

Figure 5-8. Help Topic Dialog

Subtopics

If a topic has subtopics, the help topic dialog contains a *Subtopics* pull-down menu. All of the subtopics are included on the menu. Select an item from the *Subtopics* menu to display help text for the subtopic.

Help Title

The title identifies which topic the help text describes.

Help Text

This field contains the help text for the topic. You can select the text and paste it into other clients.

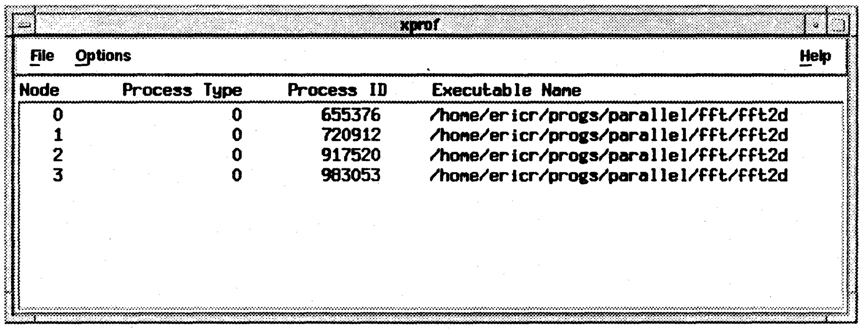
Items that are underlined in the text represent links to other help topics. Clicking on an underlined text entry displays the help topic dialog for that entry.

Done

Use the *Done* button to dismiss the help topic dialog.

Profile Directory Contents List

The main window list entries identify profile output files that exist for specific processes. Figure 5-9 shows an example of the main window profile output file list.



```
Node       Process Type    Process ID    Executable Name
  0                   0        655376    /home/ericr/progs/parallel/fft/fft2d
  1                   0        720912    /home/ericr/progs/parallel/fft/fft2d
  2                   0        917520    /home/ericr/progs/parallel/fft/fft2d
  3                   0        983053    /home/ericr/progs/parallel/fft/fft2d
```

Figure 5-9. Profile Output File List

For each entry, the following information is shown:

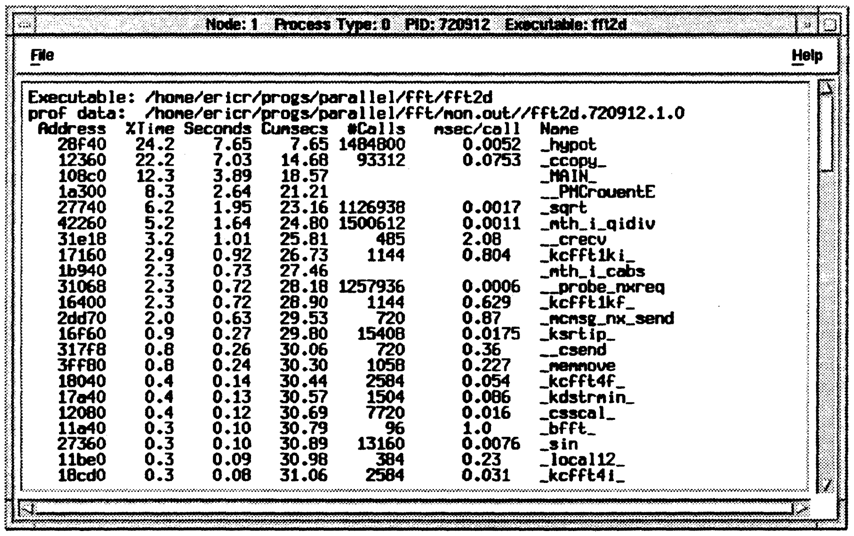
- the node number
- process type

- process ID
- executable name

The **prof** output window is displayed when you select an entry from the list. This window contains the output of **prof**, controlled by the selections in the prof settings dialog, for the selected entry. The following section describes the **prof** output window.

Prof Output Window

The **prof** output window is displayed when you select an entry from the main window file list. You can display multiple **prof** output windows at one time. You can display one for each list entry. Figure 5-10 show the **prof** output window.



```
Executable: /home/ericr/progs/parallel/fft/fft2d
prof data:  /home/ericr/progs/parallel/fft/mon.out//fft2d.720912.1.0
 Address  %Time Seconds Cumsecs  #Calls   msec/call  Name
   28f40   24.2    7.65    7.65 1484800     0.0052  _hypot
   12360   22.2    7.03   14.68   93312     0.0753  _ccopy_
   108c0   12.3    3.89   18.57                     _MAIN_
   1a300    8.3    2.64   21.21                     __PMCrouentE
   27740    6.2    1.95   23.16 1126938     0.0017  _sqrt
   42260    5.2    1.64   24.80 1500612     0.0011  _mth_i_qidiv
   31e18    3.2    1.01   25.81     485     2.08    __crecv
   17160    2.9    0.92   26.73    1144     0.804   _kcfft1ki_
   1b940    2.3    0.73   27.46                     _mth_i_cabs
   31068    2.3    0.72   28.18 1257936     0.0006  __probe_nxreq
   16400    2.3    0.72   28.90    1144     0.629   _kcfft1kf_
   2dd70    2.0    0.63   29.53     720     0.87    _mcmsg_nx_send
   16f60    0.9    0.27   29.80   15408     0.0175  _ksrtip_
   317f8    0.8    0.26   30.06     720     0.36    __csend
   3ff80    0.8    0.24   30.30    1058     0.227   _memmove
   18040    0.4    0.14   30.44    2584     0.054   _kcfft4f_
   17a40    0.4    0.13   30.57    1504     0.086   _kdstrmin_
   12080    0.4    0.12   30.69    7720     0.016   _csscal_
   11a40    0.3    0.10   30.79      96     1.0     _bfft_
   27360    0.3    0.10   30.89   13160     0.0076  _sin
   11be0    0.3    0.09   30.98     384     0.23    _local12_
   18cd0    0.3    0.08   31.06    2584     0.031   _kcfft4i_
```

Figure 5-10. Output Window

The menu bar across the top contains the menus *File* and *Help*. The inner region of the window contains the text output of **prof** for the selected list entry. You can not edit the text, but you can select it and paste it into another client. You can also save the text in a file by selecting the *Save As...* menu item from the *File* menu.

The title bar at the top of the window identifies the profile output file by displaying node number, process type, process ID, and executable name for the file.

Output Window File Menu

The output window *File* menu contains items for saving the text of **prof**'s output, for reexecuting **prof**, and for removing the window from the display. Figure 5-11 shows the output window *File* menu.

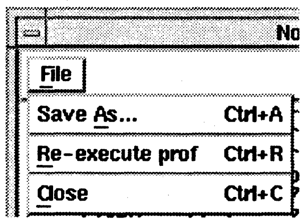


Figure 5-11. Output Window File Menu

Save As...

Save As... displays the output window file save dialog appears.You can then save the output of **prof** into a text file for later use. The default keyboard accelerator for *Save As...* is *Control-A*.

Re-execute prof

This menu item reexecutes **prof** and updates the contents of the window's output text. You should select this menu item if you changed the settings for **prof**'s options (for example, sorting by names instead of address) and you want to execute **prof** again with the new settings. The default keyboard accelerator for this item is *Control-R*.

Close

Use *Close* to dismiss the **prof** output window. The main window and any other **prof** output windows remain on the screen. To exit XProf, you must select *Exit* from the main window's *File* menu. The default keyboard accelerator for *Close* is *Control-C*.

Save prof Output Dialog

The *Save prof Output* dialog appears when you select the *Save As...* menu item from the output window *File* menu. Figure 5-12 shows the *Save prof Output* dialog.

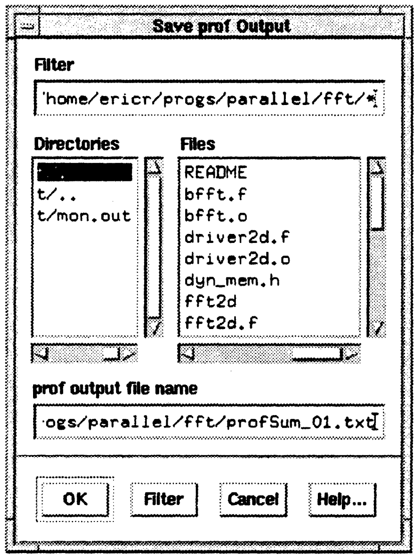


Figure 5-12. Save prof Output Dialog

This dialog is similar to the profile directory selection dialog, except that instead of selecting a file, you should first select a directory and then type in the name of a file at the end of the *prof output file name* field. You should only select a file from the *Files* list if you want to save the **prof** output over the contents of the file.

Files

When you select a file, it becomes the selection of the file dialog and XProf automatically selects *OK*. Since the file already exists, a question dialog asks you if you really want to overwrite the file.

prof output file name

This is the full path name of the file to which XProf saves the output of **prof**. Typically, you should select a directory and type in a file name at the end of this field.

Dialog Buttons

OK	XProf checks to see if the entry in the *prof output file name* field is a valid file name. If the file already exists, a question dialog asks you if you want to overwrite the file. If the file does not exist, XProf checks to be sure the file can be written to. If the file is not valid, XProf displays an error dialog informing you that the file cannot be written to. If the file is valid, the **prof** output is saved to the file and the save dialog dismisses itself.
Filter	Displays a new file list (and possibly a new directory list) based on the entry in the *Filter* field.
Cancel	Dismisses the *Save prof Output* dialog.
Help...	Displays help topic text about using the *Save prof Output* dialog.

Help Menu

The **prof** output window *Help* menu is an abbreviated version of the main window's help menu. Figure 5-13 shows the output window *Help* menu.

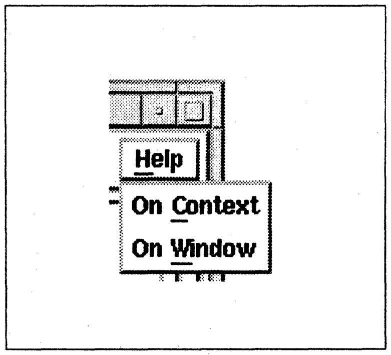


Figure 5-13. Output Window Help Menu

On Context

On Context turns the mouse pointer into a question mark and displays help topic text for any selected item.

On Window

On Window displays help topic text for the **prof** output window.

Prof Output

Xprof displays the output of **prof** in a scrollable text field. While **prof** is executing, a working dialog is displayed over the text. The working dialog dismisses itself when **prof** is complete, or you can force it to go away by selecting the *OK* button of the dialog.

You can not edit the **prof** output text, but you can select it and paste it into another client. You can also save the text in a file by using the *Save As...* menu item in the **prof** output window *File* menu.

Configuring XProf

You can configure XProf by using an X resource file. You can make the resource entries in your *.Xdefaults* file (which resides in your home directory) or in a file named *XProf* (located either in your home directory or in a directory specified by the *XAPPLRESDIR* environment variable). The entries in the *.Xdefaults* file take precedence over the *XProf* file entries.

Along with the resources corresponding to the standard X toolkit command line options, the following XProf application resources are provided to configure XProf:

XProf.listRows	An integer that controls the number of entries displayed at one time within the scrollable profile selection list.
XProf.infoPathName	The default name of the profile directory that XProf searches for when it begins execution.
XProf.includeNonGlobals	A boolean (True / False) value that chooses the *Include non-globals* display option in the settings dialog.
XProf.includeAllSymbols	A boolean (True / False) value that chooses the *Include all symbols* display option in the settings dialog.
XProf.noReportHeader	A boolean (True / False) value that chooses the *No report header* display option in the settings dialog.
XProf.includeSummary	A boolean (True / False) value that chooses the *Include summary* display option in the settings dialog.
XProf.sortByTime	A boolean (True / False) value that chooses the *Total time* sort option in the settings dialog.
XProf.sortByCall	A boolean (True / False) value that chooses the *Number of calls* sort option in the settings dialog.

XProf.sortByAddress	A boolean (True / False) value that chooses the *Symbol address* sort option in the settings dialog.
XProf.sortByName	A boolean (True / False) value that chooses the *Symbol name* sort option in the settings dialog.
XProf.addressOctal	A boolean (True / False) value that chooses the *Octal* address option in the settings dialog.
XProf.addressHex	A boolean (True / False) value that chooses the *Hexadecimal* address option in the settings dialog.

Most of these options are for the settings dialog. If you have some options that you commonly use, you can use the *XProf* resource file to initialize the settings dialog. This allows you to go directly to generating **prof** files rather than bringing up the settings dialog each time.

The command line options that affect the values of XProf's application resources take precedence over the values in the resource file.

Default Configuration

XProf has the following default resource settings:

XProf.listRows	10
XProf.infoPathName	mon.out
XProf.includeNonGlobals	False
XProf.includeAllSymbols	False
XProf.noReportHeader	False
XProf.includeSummary	False
XProf.sortByTime	True
XProf.sortByCall	False
XProf.sortByAddress	False
XProf.sortByName	False
XProf.addressOctal	False
XProf.addressHex	True

XGprof 6

This chapter describes XGprof, a graphical front end to **gprof**. For a description of **gprof**, refer to Chapter 4, *Performance Monitoring*. Using **gprof** from the command line can be complicated in the Paragon environment, since a directory of files (instead of just a single file) can be created when you profile an application. For parallel applications, each process running on the mesh that is prepared for profiling (through instrumentation with IPD) generates a profile output file. XGprof helps you select files by displaying the following information for each file in the profile directory:

- Node number
- Process type
- Process ID
- Executable name

When you select one or more file entries and click on the **gprof** button, XGprof executes **gprof** on the files and displays the output in a separate, scrollable window.

XGprof provides dialogs to assist you in selecting a profile output directory, saving the output of **gprof** to a text file, and setting the runtime options for **gprof**.

XGprof also provides online, context-sensitive help.

You can use XGprof as a stand-alone graphical interface or from ParAide, the graphical front end to the Paragon toolset. The XIPD graphical front end to the Interactive Parallel Debugger also provides a connection to XGprof. XIPD users can select the source code of a program to instrument for profiling, run the program, and invoke XGprof to examine the profile results. Chapter 1 describes ParAide, and Chapter 3 describes XIPD.

Using XGprof

This section describes how to use XGprof to examine profile output of parallel applications on your Paragon system. Detailed information about the individual windows, menus, dialogs, and commands can be found in the section *Windows, Menus, and Commands.*

Invoking XGprof

To invoke XGprof use the **xgprof** command as follows:

xgprof [*display_options*] [*routine_options*]... [-m *gprofdir*] [*X Toolkit parameters*]

display_options can be any of the following:

-a	Suppress printing statically-declared functions.
-b	Provide brief output.
-s	Produce a *gmon.sum* file.
-z	Display routines that have zero usage (time / number of calls).
-rows *numrows*	Display *numrows* rows in XGprof's profile list.

routine_options can be any of the following:

-e *routine*	Suppress printing the graph profile entry for *routine* and all of its descendants. More than one **-e** option may be given.
-f *routine*	Print only the graph profile entry for *routine* and its descendants. More than one **-f** option may be given. **-f** overrides **-e** if both are included on the command line.
-E *routine*	Suppress printing the graph profile entry for *routine* and exclude the time spent in the routine (and its descendants) from the total. More than one **-E** option may be given.
-F *routine*	Print only the graph profile entry for *routine* and its descendants and also use only the times of the routines in total computations. More than one **-F** option may be given. **-F** overrides **-E** if both are included on the command line.

The other parameters are described as follows:

-m *gprofdir*	Specify *gprofdir* as the path name of the profile output directory to be initially read. This path name can either be an absolute path or a path relative to the current directory. *gprofdir* defaults to *mon.out.*
X Toolkit parameters	The standard parameters supported by the X Toolkit (see command-line options in the *X Toolkit Intrinsics Programming Manual*). You can specify these parameters on the XGprof command line.

Choosing a Profile Output Directory

You can choose a profile output directory by any of the following:

- Having the default directory name (typically *gmon.out*) in your current directory
- Passing the directory name to XGprof as a command line argument.
- Selecting the directory is selected via the *Open...* dialog of the *File* menu.

Once you choose a profile output directory, XGprof reads a special descriptive file in the directory and displays the profile file names.

Setting gprof Run-Time Options

Before you execute **gprof**, you can set preferences for the format of the output with the *gprof Settings* dialog of the *Options* menu. These runtime settings include the following:

- Sorting methods (for example, sort by symbol name)
- Address notation (hexadecimal or octal)
- Miscellaneous display options (for example, include local routines in the report)

You can also specify format settings in an XGprof resource file. This allows you to automatically set the output format each time you invoke XGprof. For a description of how to use a resource file to configure XGprof, refer to the section *Configuring XGprof.*

Selecting a Profile Output File

When you choose a profile output directory, the directory contents are displayed within a scrolling list, sorted by node number, process type, process ID, and executable name. When you select one or more of the entries in the list and click on the **gprof** button, XGprof creates a **gprof** output window and executes **gprof**, using the specified runtime settings. The output of **gprof** (standard error and standard output) appears in this output window.

A **gprof** output window is available for each list entry. If you select a list entry while its gprof output window is somewhere on the screen, the window is brought to the front of the screen.

Examining gprof Output

The **gprof** output window consists of two pull-down menus and a scrollable text region containing the output of **gprof**. You can not edit the text, but you can select it and paste it into another client.

If you change the runtime settings for **gprof** and want to generate new output for the **gprof** output window, you can choose the *Re-execute gprof* menu item. You can also save the **gprof** output into a text file with the file dialog brought up when you select the *Save As...* menu item.

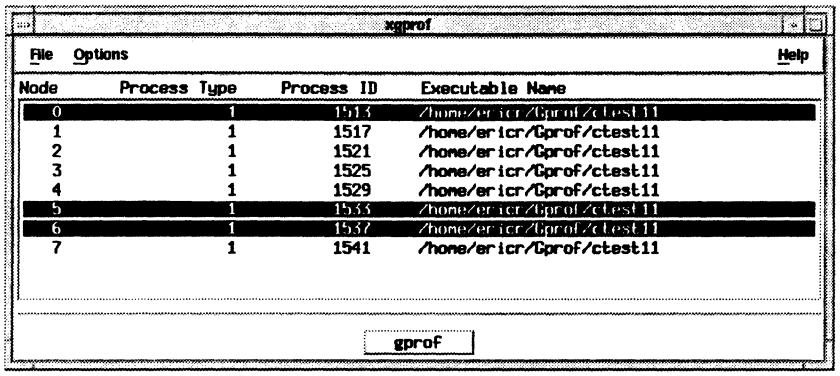
Windows, Menus, and Commands

XGprof has two windows: the main window and the output window. The following sections describe these windows and the menus and commands you can invoke from the windows.

Main Window

The main window of XGprof contains menus and a list of the profile directory contents. Figure 6-1 shows the XGprof main window.

The menu bar across the top of the main window contains the menus: *File, Options,* and *Help*. The inner region of the main window contains a list of the profile information contained within the chosen profile output directory and a **gprof** button that will launch **gprof** to analyze the selected files. The list is empty if no directory has been selected.



Node	Process Type	Process ID	Executable Name
0	1	1513	/home/ericr/Gprof/ctest11
1	1	1517	/home/ericr/Gprof/ctest11
2	1	1521	/home/ericr/Gprof/ctest11
3	1	1525	/home/ericr/Gprof/ctest11
4	1	1529	/home/ericr/Gprof/ctest11
5	1	1533	/home/ericr/Gprof/ctest11
6	1	1537	/home/ericr/Gprof/ctest11
7	1	1541	/home/ericr/Gprof/ctest11

Figure 6-1. Main Window

File Menu

The *File* menu contains items to open a profile output directory and to exit XGprof. You can pull down the *File* menu by selecting the menu with the pointer or by pressing the menu's mnemonic key **F**. Figure 6-2 shows the *File* menu

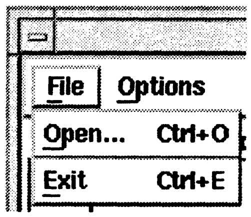


Figure 6-2. File Menu

Open...

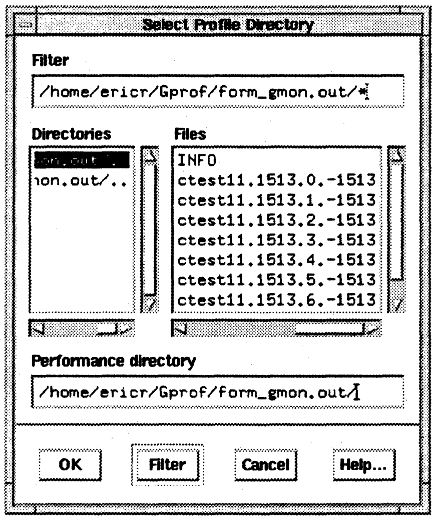
The *Open...* menu item displays the *Select Profile Directory* dialog. This dialog is always enabled. If the open is successful, the contents of the main window's list are replaced with the contents of the selected directory. The default keyboard accelerator for this item is *Control-O*.

Exit

The *Exit* menu item quits XGprof and closes all open windows. XGprof displays a question dialog to ask you if you really want to quit. The default keyboard accelerator for *Exit* is *Control-E*.

Select Profile Directory

XGprof displays the *Select Profile Directory* dialog when you select the *Open...* menu item from the main window. Figure 6-3 shows the *Select Profile Directory* dialog.



Filter

```
/home/ericr/Gprof/form_gmon.out/*
```

Directories

```
on.out/.
non.out/..
```

Files

```
INFO
ctest11.1513.0.-1513
ctest11.1513.1.-1513
ctest11.1513.2.-1513
ctest11.1513.3.-1513
ctest11.1513.4.-1513
ctest11.1513.5.-1513
ctest11.1513.6.-1513
```

Performance directory

```
/home/ericr/Gprof/form_gmon.out/
```

Figure 6-3. Select Profile Directory Dialog

To select a profile directory, select the directory name and then select *OK*. You can also select a file within the directory and XGprof selects the profile directory containing the file.

The following sections describe the features of the *Select Profile Directory* dialog.

Filter

This text field contains the filter for displaying files. All profile directories are displayed, but you can filter out files to control the length of the file list. If you want all of the file names to be displayed, the filter should end with an asterisk ('*'), as in the following example:

/home/*username*/src/mon.out/*

If you specify an invalid directory, the dialog issues a beep to the console and does not change the file or directory list.

Directories

All directories in the current directory (specified by the filter) are listed in this field. Selecting a directory or pressing *Filter* makes the directory the current directory. Directories are never filtered out of the directory list.

Files

All files in the current directory that pass through the filter are listed here. It is possible for this list to be empty (represented by a [ ] in the file list) if no files match the filter or if the directory is empty. Make the filter end with a '/*' to see all files in the directory.

If you select a file, it becomes the selection of the file dialog and *OK* is automatically invoked, dismissing the file selection dialog.

Performance directory

This field shows the current file selection. If you select *OK*, XGprof uses the file name contained in this field as the profile output directory name.

Dialog Buttons

OK	When you select *OK*, XGprof checks the entry in the *Performance directory* field to be sure it is a valid profile output directory (or a file within a valid directory). If the directory is valid, the contents appear in the main window's contents list.
Filter	XGprof uses the entry in the *Filter* field to obtain a new file list and a new directory list if the filter has been changed to a different directory.
Cancel	When selected, XGprof dismisses the dialog.
Help...	Displays help topic text about using the *Select Profile Directory* dialog.

Options Menu

The *Options* menu, available from the main window, is used to select which command arguments XGprof uses when it invokes **gprof**. Select the *gprof Settings...* menu item to set your preferences. Figure 6-4 shows the *Options* menu.

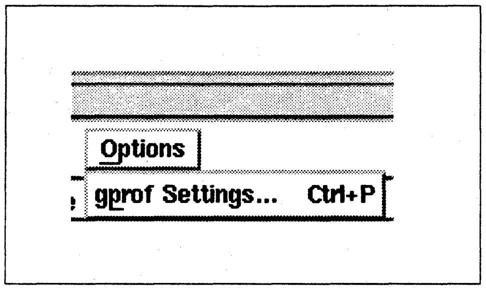


Figure 6-4. Options Menu

gprof Settings...

This selection displays the *gprof Settings* dialog. Changes to this dialog affect subsequent **gprof** generated reports. The default keyboard accelerator for this item is *Control-P*.

gprof Settings

The **gprof** utility has a number of command line options. You can set these options with the *Enter gprof Settings* dialog. Figure 6-5 shows the *Enter gprof Settings* dialog.

The *Enter gprof Settings* dialog has its own defaults. You can change these defaults with command line options to XGprof (as described in the section *Invoking XGprof*) or with entries in the *XGprof* resource file. The *XGprof* resource file is described in the section *Configuring XGprof*.

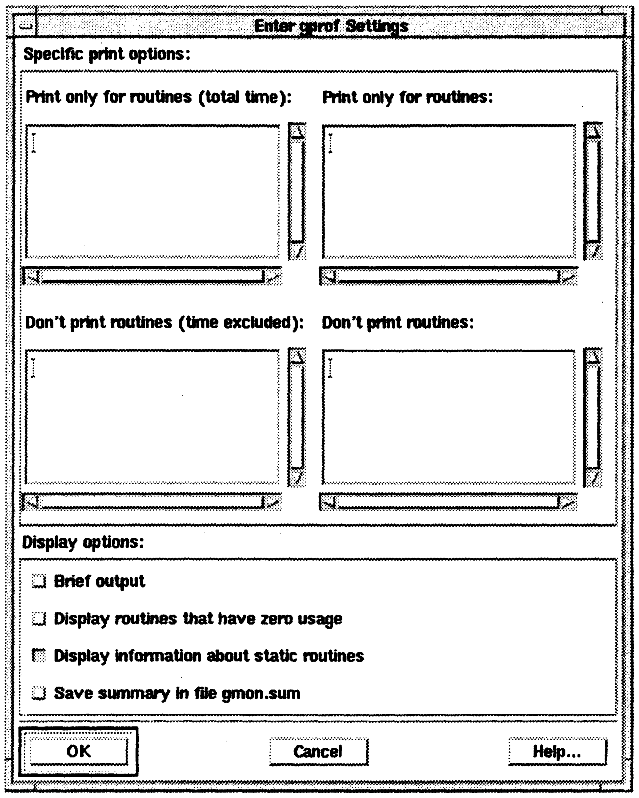

Enter gprof Settings
Specific print options:
Print only for routines (total time):
Print only for routines:
Don't print routines (time excluded):
Don't print routines:
Display options:
Brief output
Display routines that have zero usage
Display information about static routines
Save summary in file gmon.sum
OK
Cancel
Help...

Figure 6-5. Enter gprof Settings Dialog

The following sections describe the features of the *Enter gprof Settings* dialog.

Print only for routines (total time)

This item sets the option to display only the graph profile entry for the given routine names and the routines they invoke. These routines are used to calculate the total time and percentage time. You can leave this entry blank.

Print only for routines

This item sets the option to only display the graph profile entry for the given routine names and the routines they invoke. You can leave this entry blank.

Don't print routines (time excluded)

This item sets the option to suppress displaying the graph profile entry for the given routine names and the routines they invoke, excluding the time spent in the function and its descendants from the total time and percentage time computations. You can leave this entry blank.

Don't print routines

This item sets the option to suppress displaying the graph profile entry for the given routine names and the routines they invoke. You can leave this entry blank.

Display options

Brief output	Suppresses display of descriptions of each field in the profile.
Display routines that have zero usage	Displays routines which have zero call counts and zero accumulated time.
Display information about static routines	Displays information about statically-declared functions.
Save summary in file gmon.sum	Produces a *gmon.sum* file that represents the sum of the profile information in all the specified profile files. This file can be passed to **gprof** (not XGprof) to produce a report.

Dialog Buttons

OK	Dismisses the settings dialog. Changes to the dialog are noted and are used the next time **gprof** is executed. To make changes persistent to the next execution of XGprof, you must set the options in your XGprof resource file (refer to the section *Configuring XGprof*).
Cancel	Dismisses the settings dialog. Any changes to the dialog are discarded.
Help...	Displays help text for the *Enter gprof Settings* dialog.

Help Menu

The *Help* menu, available from the main window, provides various levels of help. This menu is always enabled. Through the *Help* menu, you can obtain context sensitive help and browse through all of the help topics. Figure 6-6 shows the *Help* menu.

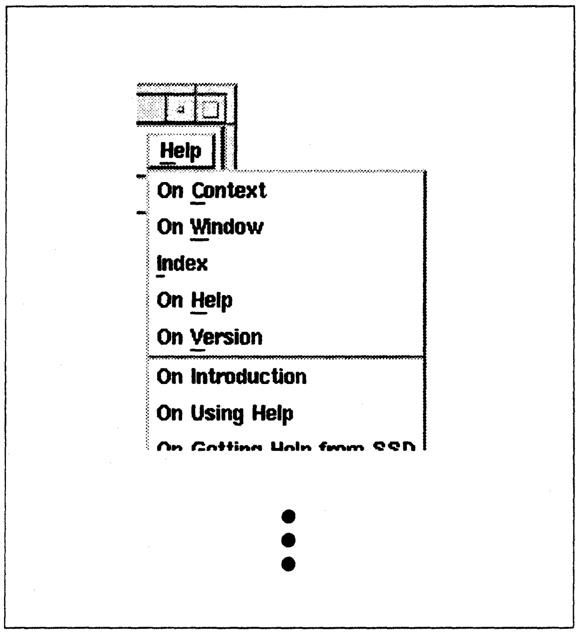


Figure 6-6. Help Menu

On Context

The *On Context* menu item enables context-sensitive help and changes the mouse pointer to a question mark. You can then click on areas of the XGprof interface. If there is a help entry for the selected area (such as the list area of the main window), XGprof displays the help topic text for that feature. You can also obtain context help from the keyboard by pressing the *Help* key (typically mapped to F1) while the keyboard focus is directed to the desired interface feature.

On Window

Displays the help topic text for the main window.

Index

Displays the *Index* dialog. All help topics for XGprof are listed in the *Index* dialog.

On Help

Displays the help topic text that explains how to use all of the aspects of help.

On Version

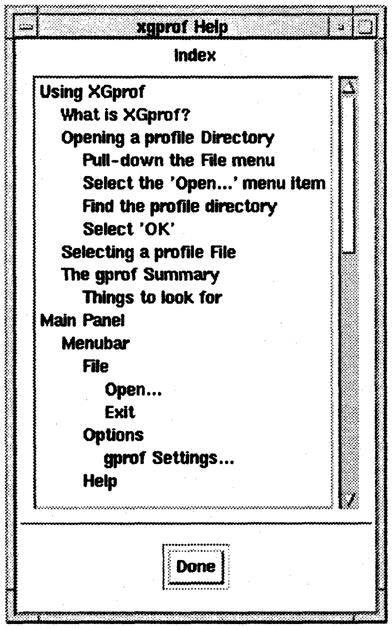
Displays a dialog containing XGprof version information.

Additional Topics

The names of all the major XGprof help topics follow the *On Version* entry on the *Help* menu. When you select a topic, XGprof displays help text for the selected topic.

Help Index

XGprof displays the *Index* dialog when you choose the *Index* menu item from the *Help* menu. The help index contains all the topics and sub-topics of help available for XGprof. Sub-topics are indented underneath major topics. Figure 6-7 shows the *Index* dialog.



```
Using XGprof
  What is XGprof?
  Opening a profile Directory
    Pull-down the File menu
    Select the 'Open...' menu item
    Find the profile directory
    Select 'OK'
  Selecting a profile File
  The gprof Summary
    Things to look for
Main Panel
  Menubar
    File
      Open...
      Exit
    Options
      gprof Settings...
    Help
```

Figure 6-7. Help Index Dialog

Help Topics

This field contains a scrollable list of all XGprof help topics. Select a topic from the list to display the help text for that topic.

Done

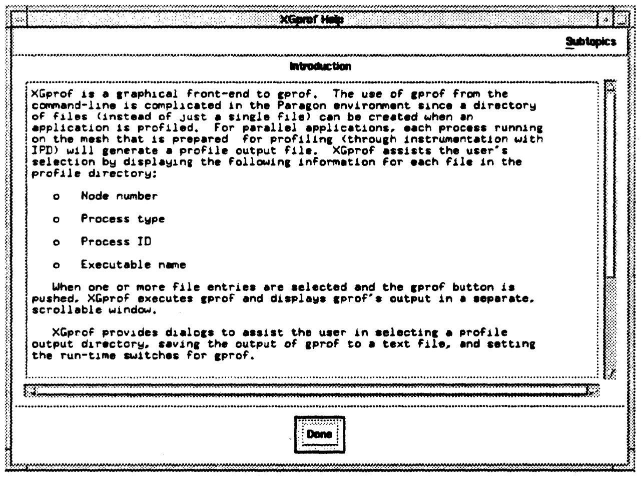
Select *Done* to dismiss the help index dialog.

Help Topic

The help topic dialog appears when you request help for a particular topic. You can select a topic by any of the following:

- selecting the topic in the help *Index* dialog
- selecting a dialog's *Help* button
- using context sensitive help
- selecting a major topic from the *Help* menu.

Figure 6-8 shows a help topic dialog.



```
XGprof is a graphical front-end to gprof.  The use of gprof from the
command-line is complicated in the Paragon environment since a directory
of files (instead of just a single file) can be created when an
application is profiled.  For parallel applications, each process running
on the mesh that is prepared  for profiling (through instrumentation with
IPD) will generate a profile output file.  XGprof assists the user's
selection by displaying the following information for each file in the
profile directory:

   o   Node number

   o   Process type

   o   Process ID

   o   Executable name

   When one or more file entries are selected and the gprof button is
pushed, XGprof executes gprof and displays gprof's output in a separate,
scrollable window.

   XGprof provides dialogs to assist the user in selecting a profile
output directory, saving the output of gprof to a text file, and setting
the run-time switches for gprof.
```

Figure 6-8. Help Topic Dialog

Subtopics

If a topic has subtopics, the help topic dialog contains a *Subtopics* pull-down menu. All of the subtopics are included on the menu. Select an item from the *Subtopics* menu to display help text for the subtopic.

Help Title

The title identifies which topic the help text describes.

Help Text

This field contains the help text for the topic. You can select the text and paste it into other clients.

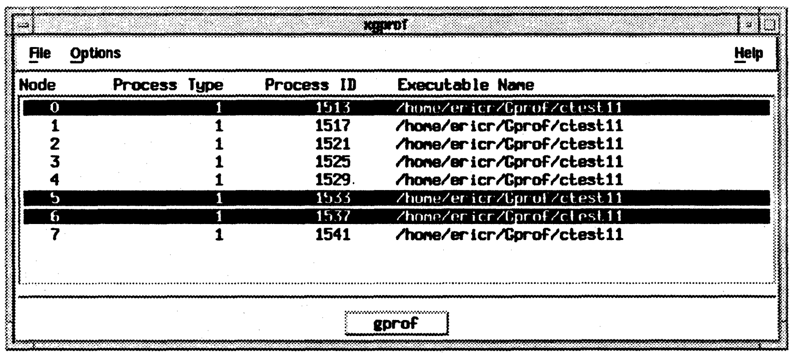
Items that are underlined in the text represent links to other help topics. Clicking on an underlined text entry displays the help topic dialog for that entry.

Done

Use the *Done* button to dismiss the help topic dialog.

Profile Directory Contents List

The main window list entries identify profile output files that exist for specific processes. Figure 6-9 shows an example of the main window profile output file list.



```
Node      Process Type    Process ID    Executable Name
   0                 1          1513    /home/ericr/Gprof/ctest11
   1                 1          1517    /home/ericr/Gprof/ctest11
   2                 1          1521    /home/ericr/Gprof/ctest11
   3                 1          1525    /home/ericr/Gprof/ctest11
   4                 1          1529    /home/ericr/Gprof/ctest11
   5                 1          1533    /home/ericr/Gprof/ctest11
   6                 1          1537    /home/ericr/Gprof/ctest11
   7                 1          1541    /home/ericr/Gprof/ctest11
```

Figure 6-9. Profile Output File List

For each entry, the following information is shown:

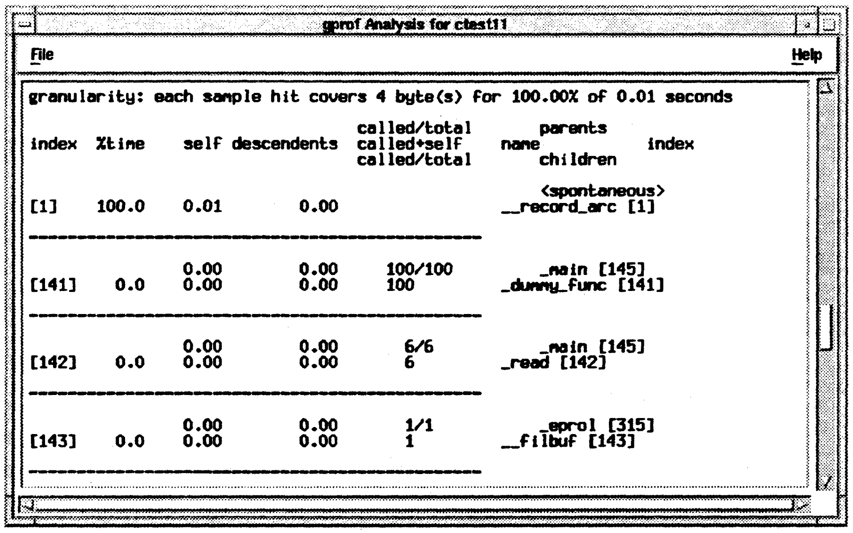
- the node number
- process type
- process ID
- executable name

The **gprof** output window is displayed when you select one or more entries from the list and click on the **gprof** button. This window contains the output of **gprof**, controlled by the selections in the gprof settings dialog, for the selected entries. You can select multiple entries by holding down the <SHIFT> or <CTRL> key while selecting elements.

The following section describes the **gprof** output window.

Gprof Output Window

The **gprof** output window is displayed when you select an entry from the main window file list. You can display multiple **gprof** output windows at one time. You can display one for each set of files for which **gprof** is executed. Figure 6-10 show the **gprof** output window.



```
granularity: each sample hit covers 4 byte(s) for 100.00% of 0.01 seconds

                                     called/total       parents
index  %time     self descendents   called+self    name              index
                                     called/total       children

                                                        <spontaneous>
[1]    100.0     0.01        0.00                  __record_arc [1]
-----------------------------------------------

                 0.00        0.00     100/100           _main [145]
[141]     0.0    0.00        0.00     100          _dummy_func [141]
-----------------------------------------------

                 0.00        0.00       6/6             _main [145]
[142]     0.0    0.00        0.00       6          _read [142]
-----------------------------------------------

                 0.00        0.00       1/1             _eprol [315]
[143]     0.0    0.00        0.00       1          __filbuf [143]
-----------------------------------------------
```

Figure 6-10. Output Window

The menu bar across the top contains the menus *File* and *Help*. The inner region of the window contains the text output of **gprof** for the selected list entries. You can not edit the text, but you can select it and paste it into another client. You can also save the text in a file by selecting the *Save As...* menu item from the *File* menu.

The window's title identifies the executable for which the report is executed.

Output Window File Menu

The output window *File* menu contains items for saving the **gprof** output text, for reexecuting **gprof**, and for removing the window from the display. Figure 6-11 shows the output window *File* menu.

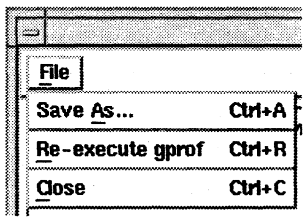


Figure 6-11. Output Window File Menu

Save As...

Save As... displays the output window file save dialog appears.You can then save the output of **gprof** into a text file for later use. The default keyboard accelerator for *Save As...* is *Control-A*.

Re-execute gprof

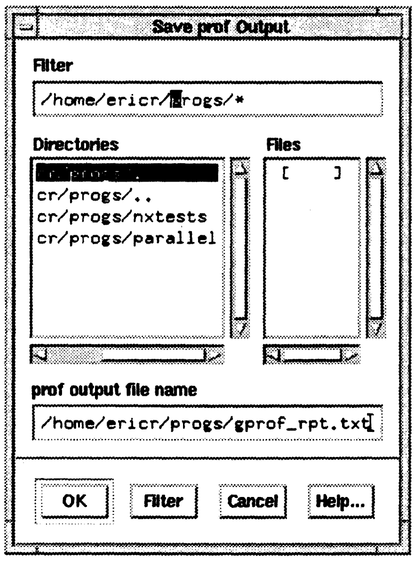
This menu item reexecutes **gprof** and updates the contents of the window's output text. You should select this menu item if you changed the settings for **gprof**'s options (for example, sorting by names instead of address) and you want to execute **gprof** again with the new settings. The default keyboard accelerator for this item is *Control-R*.

Close

Use *Close* to dismiss the **gprof** output window. The main window and any other **gprof** output windows remain on the screen. To exit XGprof, you must select *Exit* from the main window's *File* menu. The default keyboard accelerator for *Close* is *Control-C*.

Save gprof Output Dialog

The *Save gprof Output* dialog appears when you select the *Save As...* menu item from the output window *File* menu. Figure 6-12 shows the *Save gprof Output* dialog.



Filter: /home/ericr/progs/*

Directories:

```
cr/progs/.
cr/progs/..
cr/progs/nxtests
cr/progs/parallel
```

Files: [ ]

prof output file name: /home/ericr/progs/gprof_rpt.txt

Figure 6-12. Save gprof Output Dialog

This dialog is similar to the profile directory selection dialog, except that instead of selecting a file, you should first select a directory and then type in the name of a file at the end of the *prof output file name* field. You should only select a file from the *Files* list if you want to save the **gprof** output over the contents of the file.

Files

When you select a file, it becomes the selection of the file dialog and XGprof automatically selects *OK*. Since the file already exists, a question dialog asks you if you really want to overwrite the file.

prof output file name

This is the full path name of the file to which XGprof saves the output of **gprof**. Typically, you should select a directory and type in a file name at the end of this field.

Dialog Buttons

OK	XGprof checks to see if the entry in the *prof output file name* field is a valid file name. If the file already exists, a question dialog asks you if you want to overwrite the file. If the file does not exist, XGprof checks to be sure the file can be written to. If the file is not valid, XGprof displays an error dialog informing you that the file cannot be written to. If the file is valid, the **gprof** output is saved to the file and the save dialog dismisses itself.
Filter	Displays a new file list (and possibly a new directory list) based on the entry in the *Filter* field.
Cancel	Dismisses the *Save gprof Output* dialog.
Help...	Displays help topic text about using the *Save gprof Output* dialog.

Help Menu

The **gprof** output window *Help* menu is an abbreviated version of the main window's help menu. Figure 6-13 shows the output window *Help* menu.

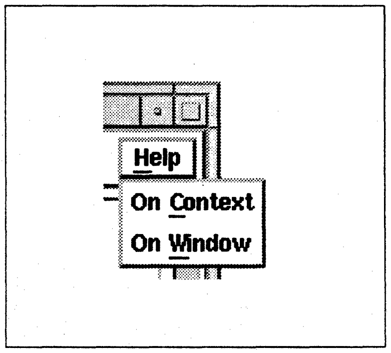


Figure 6-13. Output Window Help Menu

On Context

On Context turns the mouse pointer into a question mark and displays help topic text for any selected item.

On Window

On Window displays help topic text for the **gprof** output window.

Prof Output

XGprof displays the output of **gprof** in a scrollable text field. While **gprof** is executing, a working dialog is displayed over the text. The working dialog dismisses itself when **gprof** is complete, or you can force it to go away by selecting the *OK* button of the dialog.

You can not edit the **gprof** output text, but you can select it and paste it into another client. You can also save the text in a file by using the *Save As...* menu item in the **gprof** output window *File* menu.

Configuring XGprof

You can configure XGprof by using an X resource file. You can make the resource entries in your *.Xdefaults* file (which resides in your home directory) or in a file named *XGprof* (located either in your home directory or in a directory specified by the *XAPPLRESDIR* environment variable). The entries in the *.Xdefaults* file take precedence over the *XGprof* file entries.

Along with the resources corresponding to the standard X toolkit command line options, the following XGprof application resources are provided to configure XGprof:

XGprof.listRows	An integer that controls the number of entries displayed at one time within the scrollable profile selection list.
XGprof.infoPathName	The default name of the profile directory that XGprof searches for when it begins execution.
XGprof.briefOutput	A boolean (True / False) value that chooses the *Brief output* display option in the settings dialog.
XGprof.zeroUsage	A boolean (True / False) value that chooses the *Display routines that have zero usage* display option in the settings dialog.

XGprof.staticRoutines	A boolean (True / False) value that chooses the *Display information about static routines* display option in the settings dialog.
XGprof.saveSummary	A boolean (True / False) value that chooses the *Save summary if file gmon.sum* display option in the settings dialog.

Most of these options are for the settings dialog. If you have some options that you commonly use, you can use the *XGprof* resource file to initialize the settings dialog. This allows you to go directly to generating **gprof** files rather than bringing up the settings dialog each time.

The command line options that affect the values of XGprof's application resources take precedence over the values in the resource file.

Default Configuration

The following are XGprof's default resource settings:

XGprof.listRows	10
XGprof.infoPathName	gmon.out
XGprof.briefOutput	False
XGprof.zeroUsage	False
XGprof.staticRoutines	True
XGprof.saveSummary	False

ParaGraph 7

This chapter describes ParaGraph, a performance visualization tool for the Paragon system. ParaGraph is one of the most widely-used tools for analyzing the performance of parallel applications. Its main purpose is to visualize the communication performance of parallel programs using a variety of displays. Paragraph was originally developed by M. Heath and J. Finger under a research grant from D.O.E.

ParaGraph displays the performance behavior of a Paragon application on your workstation using a trace file generated by the performance monitoring subsystem.

A program creates a trace file if it has been loaded under IPD and processed with the **instrument** command. Without this trace file, you can not use ParaGraph to analyze an application. For a complete description of the IPD **instrument** command, refer to the *Paragon*™ *Interactive Parallel Debugger Reference Manual*

Overview

ParaGraph is a graphical display system for visualizing the behavior and performance of Paragon applications. The visual animation is based on execution trace information monitored during an actual run of an application. ParaGraph replays the resulting trace data pictorially to provide a dynamic depiction of the behavior of the parallel program, as well as graphical summaries of its overall performance.

ParaGraph is used in a post-mortem fashion to analyze event traces generated by the performance monitoring subsystem. It provides a variety of displays to visualize the performance of a parallel application.You can choose as many displays as will fit on the screen from the four different types of displays (*utilization*, *communication*, *task*, and *other*).

After selecting the desired displays, you press the start button to begin the graphical simulation of the parallel program based on the tracefile specified. The animation then proceeds to the end of the tracefile. You can, however, interrupt it for detailed study by using the pause/resume button. For even more detailed study, the step button provides a single-step mode that processes the tracefile one (or a few) event(s) at a time.You can select a particular time interval for study by specifying starting and stopping times.

You can restart the entire animation at any time by pressing the start button. Most of the displays show program behavior dynamically as individual events occur, but some show only overall summary information at the end of the run. Most of the displays show information on a per-processor basis. ParaGraph allows the visualization of traces that contain only subsets of the nodes used by a parallel application. In addition, you can focus on nodes by selecting only a subset of the nodes in the trace for visualization.

Invoking ParaGraph

To invoke ParaGraph, use the **paragraph** command as follows:

paragraph [**-m** | **-s** | **-p**] [**-f** *filename*] [**-e** *environment_file*] [*X Toolkit parameters*]

The command line parameters are defined as follows:

-m	Forces monochrome display mode. This is useful for making black-and-white hardcopies from a color screen.
-s	Forces ParaGraph to allocate read-only colorcells from the default colormap. By default, ParaGraph attempts to allocate read/write colorcells. This allows you to change the colors used within the displays interactively. However, read/write colorcells can not be shared by different applications and are thus a limited resource. If you use the **-s** option, ParaGraph's colors can not be edited and the Colors entry in the options menu is disabled.
-p	Forces ParaGraph to allocate read/write colorcells from a private colormap. Use this option if not enough colorcells can be allocated from the default colormap because they have been used up by other applications.
-f *filename*	Specifies the name of a trace file that contains previously-saved performance data in the Paragon SDDF trace format.
-e *environment_file*	Specifies the name of a file containing a layout environment produced through the Save Layout command.
X Toolkit parameters	The standard parameters supported by the X Toolkit (refer to the *X Toolkit Intrinsics Programming Manual*).

Display Overview

When you invoke ParaGraph, the main window is displayed. From this window you can select from the four types of displays provided by ParaGraph. This section describes the four types of displays. The pull-down menus, commands, buttons and dialog boxes are described in the section *Windows, Menus, and Commands*.

Utilization Displays

The utilization displays are concerned primarily with processor utilization. You can use them to determine the effectiveness with which the processors are used and how evenly the computational work is distributed across the processors. There are six utilization displays:

- Utilization Count
- Gantt Chart
- Utilization Summary
- Utilization Meter
- Concurrency Profile
- Kiviat Diagram

ParaGraph uses five different states to determine processor utilization.

idle	The processor has suspended execution (it is awaiting a message that has not yet arrived using a blocking message passing call) or it has ceased execution at the end of the run.
overhead	The processor is executing in the communication subsystem.
I/O	The processor is executing I/O statements.
busy	The processor is executing some portion of the program other than the communication or I/O subsystem.
flush	The time spent flushing the event buffers from the performance monitoring library to the event trace server.

The percentage of the time each of the processors is in the different states and the development over time is depicted using a user-configurable color scheme for each of the states.

Communication Displays

The communication displays are concerned primarily with depicting interprocessor communication.You can use them to determine the frequency, volume, and overall pattern of communication, and whether there is congestion in the message queues. There are ten communication displays:

- Communication Traffic
- Spacetime Diagram
- Message Queues
- Communication Matrix
- Communication Meter
- Animation
- Topology
- Node Info
- Network
- Color Code

You can use these displays to plot the total communication traffic in the interconnection network, the communications occurring between processors as a function of time, the sizes of message queues, and detailed communication statistics for user-selected processors. Some of the displays in this category don't scale beyond a certain number of processors (for example, 16 for the Topology display). Refer to the section titled *Restrictions* on page 7-57.

Task Displays

You can use the task displays to relate the information in the other displays to locations in the parallel program. They use information you provide to depict the portion of the parallel program executing at any given time. Specifically, you define "tasks" within the program by using special routines (linked into the application by default) to mark the beginning and ending of each task and assigning the task a task number.

ParaGraph provides four different displays to visualize task behavior:

- Task Count
- Task Gantt
- Task Status
- Task Summary

You can use these displays to show execution of tasks across the nodes and to measure the duration of each task as a percentage of the overall execution time of the parallel application.

Other Displays

ParaGraph provides the following additional displays:

Phase Portrait	Illustrates the relationship over time between communication and processor utilization.
Processor Status display	Captures detailed information about processor utilization, communication, and tasks in a compact format that scales up to large numbers of processors.
Trace display	Prints an annotated version of each trace event as it is read from the tracefile.
Clock display	Provides both digital and analog clock readings during the graphical simulation of the parallel program.
Statistics display	Gives numerical values for various statistics summarizing processor utilization and communication.
Coordinate Info display	Writes information produced by mouse clicks on the other displays.

Windows, Menus & Commands

The ParaGraph main window is displayed when you invoke ParaGraph. Figure 7-1 shows the ParaGraph main window.

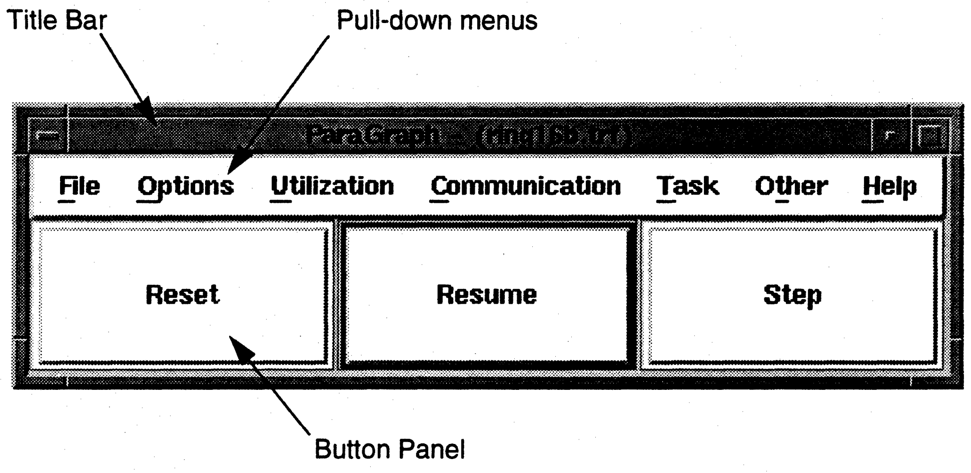


Figure 7-1. ParaGraph Main Window

Pull-down menus

The pull-down menu bar is at the top of the window. The menu bar contains the following menus:

- File
- Options
- Utilization
- Communication
- Task
- Other
- Help

Title bar

The title bar contains the tool name (ParaGraph) and the name of the trace file being visualized. If no trace file has been selected, this is indicated in the title bar.

Button panel

The button panel contains three push buttons that control the visualization process.

Start	Starts the visualization process. When you push the *Start* button, the button label changes to *Pause*. You can stop the visualization by pressing *Pause*, which changes the button label to *Resume*.
Reset	Returns to the beginning of the trace file.
Step	Steps through the trace events one (or a few) at a time.

All the buttons can be activated by clicking with the mouse. Repeated activations are possible by hitting the `osfActivate` key while the mouse is within the button panel. Thus, single-stepping is possible by repeatedly hitting the `osfActivate` key. The keyboard events for Motif applications can be configured on a system and application basis. Under Motif 1.2, the default for the `osfActivate` key is the space bar. Refer to chapter 2 of the Motif Programming Manual for details.

If no tracefile has been selected for visualization, the buttons are disabled. When the visualization reaches the end of the tracefile, the *Step* and *Resume* buttons are disabled and can be re-enabled by pressing the *Reset* button

File Menu

The *File* menu provides commands to manipulate files and quit ParaGraph. You can pull down the *File* menu by selecting the menu with the pointer or by pressing the menu's mnemonic key **F**. Figure 7-2 shows the *File* menu.

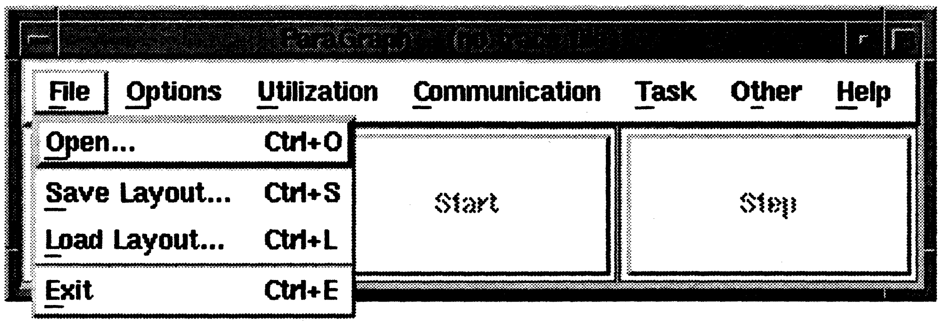
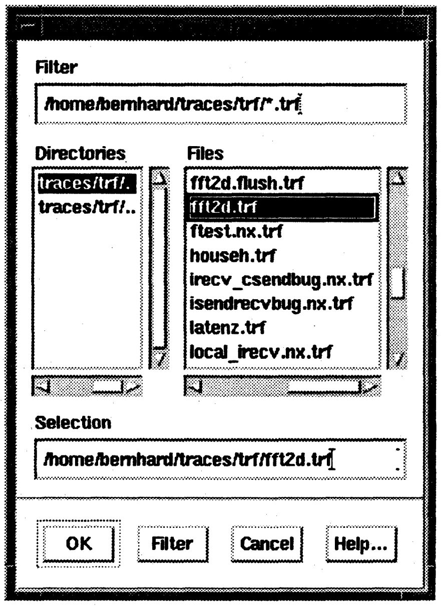


Figure 7-2. File Menu

Open...

The Open command allows you to select a trace file for replay. The command displays a file selection dialog to allow you to specify the trace file. When you choose a file name, the name of the file appears in the ParaGraph title bar. The default keyboard accelerator for *Open* is *Control-O*. Figure 7-3 shows the *Open Tracefile* dialog.


Filter

/home/bernhard/traces/trf/*.trf

Directories

traces/trf/.
traces/trf/..

Files

fft2d.flush.trf
fft2d.trf
ftest.nx.trf
househ.trf
irecv_csendbug.nx.trf
isendrecvbug.nx.trf
latenz.trf
local_irecv.nx.trf

Selection

/home/bernhard/traces/trf/fft2d.trf

Figure 7-3. Open Tracefile Dialog

Filter

This text field contains the filter for displaying files. All directories are displayed, but you can filter out files to control the length of the list. You can use standard UNIX shell expressions in the filter field. By default, the filter is set to display all trace files, as indicated by the **.trf* at the end of the filter.

Directories

All directories in the current directory (specified by the filter) are listed in this field. Selecting a directory makes it the current directory. Directories are never filtered out.

Files

All files in the current directory that pass through the filter are listed in this field. It is possible for this list to be empty (represented by a [ ] in the file list) if no files match the filter, or if the current directory is empty. Make sure the filter ends with **.trf* to list all trace files.

Selection

The *Selection* field contains the current file selection. If you select *OK*, this file is selected as the trace file for replay, and the file name is displayed in the title bar.

Dialog Buttons

OK	Selects the entry in *Selection* as the trace file for replay and dismisses the file selection dialog.
Filter	Obtains a new file list, and possibly a new directory list.
Cancel	Dismisses the file selection dialog and no trace file is selected.
Help	Displays help topic text about the dialog.

Save Layout...

The Save Layout command allows you to save the current layout of opened ParaGraph displays (including their position on the screen) to a file. The Save Layout command displays the *Save Layout* dialog. This dialog is a standard file selection dialog, similar to the *Open Tracefile* dialog described in the previous section. The default keyboard accelerator for *Save Layout* is *Control-S*.

The default name for the layout file is*.pgrc*. When you invoke ParaGraph, it checks for a file called *.pgrc* in your working directory or your home directory (in that order). If the file exists, the layout stored in the file is used to re-create the saved state. You can also use the **-e** command line option to specify a layout file.

The file produced by the Save Layout command is in ASCII format. It stores the status, size, and position of the ParaGraph displays, and configuration parameters such as the scale width or simulation speed. Options selected within the ParaGraph displays (such as the display type in the Animation display) are also stored. User-defined colors (set using the Colors command) are stored in a format conforming to X-resource specifications. The first lines of the layout file contain comments that explain the format of the file. These comments are marked by an exclamation mark at the beginning of the line.

Figure 7-4 shows an example layout file.

```
! Geometry and status for all displays. NOT conforming to Xresource format
! Format: display# opened width height xpos ypos +additional info
0 0 285 285 0 140 1 # Animation +anim_type
1 0 300 325 0 140 12 # Hypercube +hype_type_val
2 1 285 285 3 141 # Comm Matrix
3 0 286 286 0 140 # Kiviat
4 0 300 340 0 140 # Task Status
5 0 572 325 0 140 # Task Gantt
6 0 552 285 0 140 # Spacetime
8 0 572 325 0 140 # Util Gantt
9 0 572 325 0 140 # Util Count
10 0 552 620 0 140 0 0 # Node Info +stat_node +stat_type_val
11 0 592 295 0 140 1 # Traffic +traf_type_val
12 0 680 738 0 140 # Proc Status
13 0 630 295 0 140 1 # Msg Queues +queu_type_val
14 0 660 382 0 140 # Task Count
15 0 630 336 0 140 # Util Summary
16 0 360 375 0 140 # Phase Portrait
17 0 300 364 0 140 7 0 # Network +ntwk_type_val +opt_val
18 0 660 340 0 140 # Task Summary
19 0 260 322 0 140 0 # Profile +prof_type_val
20 0 50 280 0 140 # Util Meter
21 0 80 300 0 140 1 # Comm Meter +meter_type_val
22 0 536 285 0 140 # Trace Records
23 0 112 50 0 140 # Clock
24 0 408 273 0 140 # Statistics
25 0 200 208 0 140 # Buttonpress Info
26 0 87 250 0 140 0 # Color Legend +code_val
27 0 1 1 25 0 # scroll_val, backing-store, step increment, smoothing, speed
28 1 # stop on error flag

! Paragraph color definitions, conforming to Xresource format
Paragraph.fgColor:          #FF0000
Paragraph.bgColor:          #00FF00
```

Figure 7-4. Example Layout File

In this example, the only open display is the Communication Matrix display (as indicated by the 1 in the second column). Its width and height is 285 pixels, and it is positioned at x-position 3 and y position 140 on the screen.The display type selected for the animation display is a mesh (anim_type = 1). Backing store is selected and the smoothing interval is set to 25. The last two lines indicate a red foreground and a green background color for the ParaGraph displays.

Load Layout...

The Load Layout command allows you to load a saved ParaGraph layout from a file. The command displays the *Load Layout* dialog. You can use this dialog to specify the name of the environment file. The *Load Layout* dialog is a standard file selection dialog, similar to the *Open Tracefile* dialog described previously. The default keyboard accelerator for *Load Layout* is *Control-L.*

Loading a layout changes the size, state and position of the ParaGraph displays, the configuration parameters and the colors to the state they were in when the environment file was created.

Exit

The Exit command allows you to exit ParaGraph. ParaGraph displays a confirmation dialog to ask you if you really want to exit. Select *Yes* to exit ParaGraph.

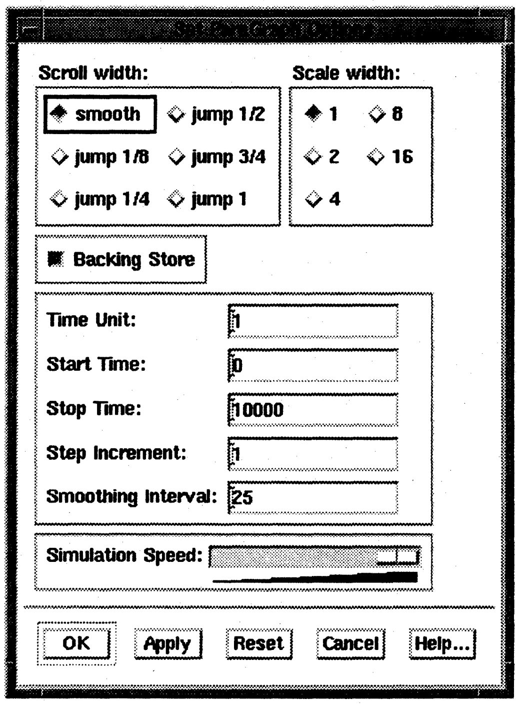
Options Menu

The Options menu provides commands that allow you to set configuration parameters and save the screen layout. You can pull down the *Options* menu by selecting the menu with the pointer or by pressing the menu's mnemonic key **O**. The following commands are provided:

- Configure
- Select Nodes
- Colors
- Message Log
- Trace Filter
- Close All

Configure

The Configure command displays the *Set ParaGraph Options* dialog. The default keyboard accelerator for *Configure* is *Control-F*. Figure 7-5 shows the *Set ParaGraph Options* dialog.



Scroll width:

[✓ ◆ smooth] ◇ jump 1/2
◇ jump 1/8 ◇ jump 3/4
◇ jump 1/4 ◇ jump 1

Scale width:

[✓ ◆ 1] ◇ 8
◇ 2 ◇ 16
◇ 4

[✓ ■ Backing Store]

Time Unit:	1
Start Time:	0
Stop Time:	10000
Step Increment:	1
Smoothing Interval:	25

Simulation Speed:

Figure 7-5. Set ParaGraph Options Dialog

Scroll width

ParaGraph displays that represent time along the horizontal dimension of the screen can smoothly scroll or jump scroll by a user-specified amount as simulation time advances. Smooth scrolling provides visual continuity, but results in a slower drawing speed. The options are:

smooth	Smooth scroll.
jump1/8	Displays are moved to the left by one eighth of the display width when the right edge of the display is reached.
jump 1/4	Displays are moved to the left by one quarter of the display width when the right edge of the display is reached.

jump1/2	Displays are moved to the left by one half of the display width when the right edge of the display is reached.
jump3/4	Displays are moved to the left by three quarters of the display width when the right edge of the display is reached
jump1	Displays are moved to the left by the full display width when the right edge of the display is reached.

Scale Width

The scale width determines the number of pixels on the screen that represent each unit of simulation time. A larger number of pixels per time unit magnifies the horizontal dimension of the scrolling displays to bring out more detail, but with less of the overall behavior of the program visible at once. The options are 1,2,4,8, or 16 pixels per time unit. The default value is 1.

Backing Store

ParaGraph distinguishes between stationary displays and displays that scroll with time. The displays that scroll with time are:

- Utilization Gantt
- Utilization Count
- Communication Traffic
- Communication Spacetime
- Node Info
- Task Gantt

For displays that scroll with time, there is no upper bound on the amount of information shown in the display. When one of these displays is resized or exposed, the information is not re-drawn. Instead, an expose event for scrolling displays is handled by scrolling the display all the way to the left edge of the display.

By turning backing store on and off, you can control whether information in the scrolling ParaGraph displays is retained if the window is displayed but not visible. Turning on backing store requires a larger amount of memory on the workstation and reduces execution speed. If the X-window server does not provide backing store, this menu is disabled.

Time Unit

The relationship between simulation time and the timestamps of the trace events is determined by the time unit chosen. By convention, event timestamps are provided in microseconds. For example, a value of 100 for the time unit in ParaGraph means that each tick of the simulation clock corresponds to 100 microseconds in the original execution of the parallel program (the timestamps in the tracefile are divided by 100).

During preprocessing, ParaGraph scans the timestamp information contained in the header of the tracefile and attempts to determine a reasonable value for the time unit. The value chosen is such that the entire length of the simulation fits into the default size of the scrolling displays. You can override this automatic choice, however, by entering a different value in the time unit subwindow. Once the time unit is set, all displays are expressed in terms of this time unit rather than the units of the original raw timestamps in the tracefile

For performance reasons, ParaGraph uses integer arithmetic to perform most calculations. The accuracy of the calculations is in part determined by the time unit chosen, with a small time unit leading to high accuracy. Changing the time unit in the middle of the simulation is possible, but can lead to quantization errors. It is best to set the time unit at the beginning of the simulation and keep it unchanged until the end.

Start Time and Stop Time

By default, ParaGraph starts the simulation at the beginning of the tracefile and proceeds to the end of the tracefile. By choosing other starting and stopping times, you can isolate any particular time period for examination. Once the specified stopping time is reached, the simulation pauses and can be resumed by typing a new stopping time or by clicking on the *Resume* or *Step* button.

The start and stop times determined by ParaGraph are normalized with respect to the minimum and maximum time stamp contained in the tracefile. Thus, if the minimum time stamp is 0.3000s and the maximum is 0.5127s, the start time determined by ParaGraph will be 0 and the stop time 2127 for a time unit of 100.

Step Increment

This parameter determines how many consecutive records from the records tracefile are processed each time you press the *Step* button. The default value is 1.

Smoothing Interval

The smoothing interval is the time interval used to calculate the average communication and utilization in the Kiviat and Phase Portrait diagrams. You can select the amount of smoothing used to avoid an excessively noisy or jumpy appearance. The amount of smoothing is determined by the width of a moving interval, with a larger value giving more smoothing and a smaller value giving less smoothing. This parameter is expressed in simulation time units and can be changed by entering a new value.

Simulation Speed

This slider controls the animation speed. By default, ParaGraph draws as fast as the workstation permits. This control provides "slow motion" replay. The slider control can be changed dynamically to change speeds during the run. Moving the slider to the left slows the simulation down, moving it to the right speeds the simulation up.

Dialog Buttons

Apply	Applies changes to the dialog values.
OK	Applies changes and dismisses the dialog.
Reset	Cancels any changes and resets the status of the dialog to the state at the last *Apply*.
Cancel	Cancels changes and dismisses the dialog.
Help	Displays help text for the dialog.

Select Nodes

The Select Nodes command displays the *Select Nodes* dialog. This dialog allows you to select a subset of the nodes in the trace for visualization. The default keyboard accelerator for *Select Nodes* is *Control-M*. Figure 7-6 shows the Select Nodes dialog.

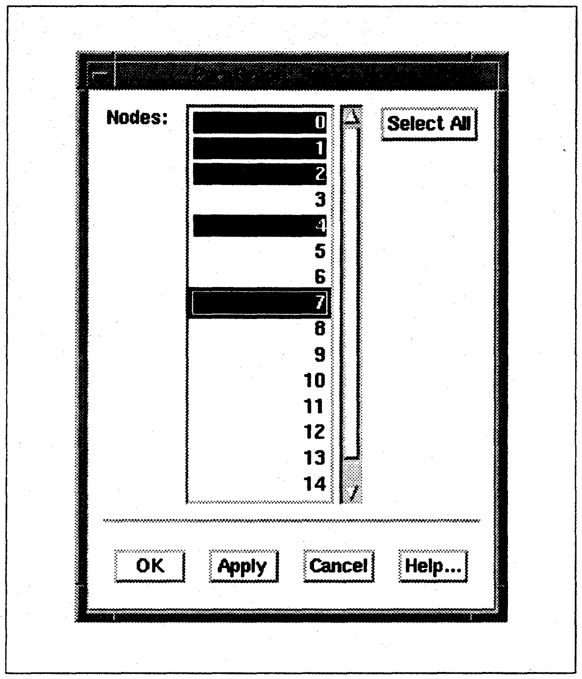


Figure 7-6. Select Nodes Dialog

You select nodes by clicking and dragging with the mouse. You can modify the selection by using the <SHIFT> key together with the left mouse button. Pressing the <CONTROL> key together with the left mouse button toggles items on and off. Selected items are highlighted. In Figure 7-6, nodes 0, 1, 2, 4 and 7 are selected for visualization. The *Select All* button selects all nodes.

This dialog allows you to focus on certain nodes of the application and provides a primitive zooming mechanism. When ParaGraph is run for a subset of the nodes in the trace, trace entries that come from nodes that are not being visualized are ignored. The effect is the same as when tracing for the nodes not selected is turned off. Selecting a subset of nodes is only possible at the beginning of a ParaGraph run. Thus, the window is disabled when you press the *Start* or *Step* button and can only be re-enabled by pressing the *Reset* button.

Dialog Buttons

Apply	Applies changes to the dialog.
OK	Applies changes and dismisses the dialog.
Cancel	Cancels changes and dismisses the dialog.
Help	Displays help text for the dialog.

Colors

You can customize the colors used in the ParaGraph displays interactively or permanently. The Colors command allows you to choose display colors interactively. The Colors command displays a dialog that contains a palette of colors from which you can choose. The facility is disabled when using a monochrome display or if you have specified the **-s** command line option. The default keyboard accelerator for *Colors* is *Control-R*. Figure 7-7 shows the *Select Colors* dialog.

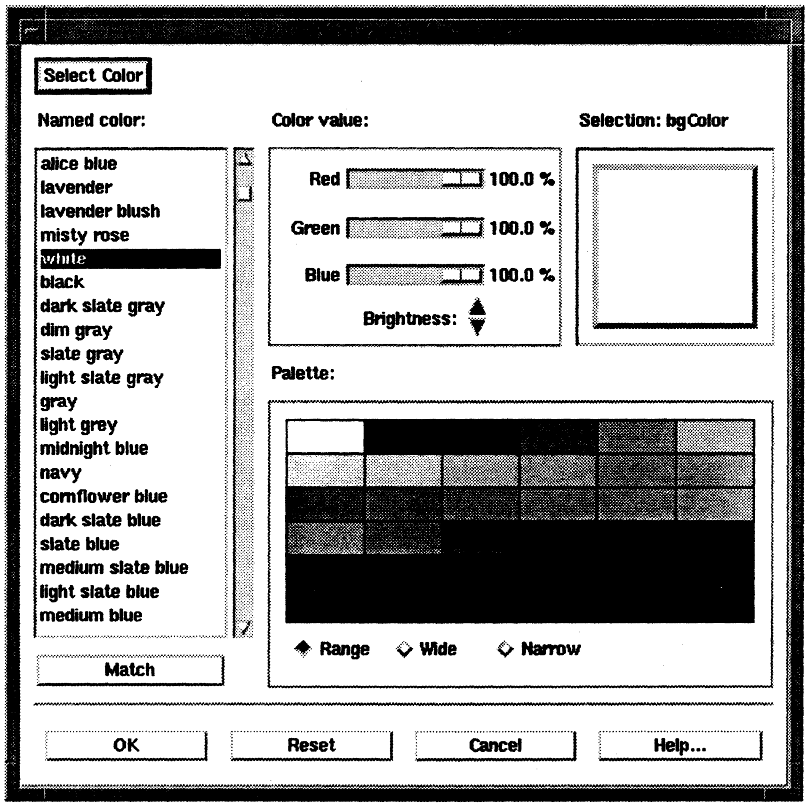


Figure 7-7. Select Colors Dialog

If no color has been selected, the *Selection* box at the top right of the window shows the label "None" and most of the window is insensitive.

To select ParaGraph colors, click on the *Select Color* button to change the cursor to a crosshair. Position the cursor in one of the ParaGraph displays over the color you want to change. Clicking with the mouse selects that color as the current color. This is indicated by the color field at the top of the dialog changing to the selected color and by the *Selection* label changing to the name of the X-resource for that color. Clicking the mouse in an area that does not belong to one of the ParaGraph displays results in an error message.

To adjust the current color you can use one of the three sliders, each controlling the red, green or blue values for the current color. You can also click on a cell displayed in the palette to use its color. The palette radio box situated below the color palette switches between the three palettes: range, narrow, and wide.

range	Covers the entire spectrum.
narrow	Covers a narrow range around the current color.
wide	Covers a wide range around the current color.

You can also select a color from the list of named colors at the left of the window. This list contains the color names from the file */usr/lib/X11/rgb.txt*. Selecting the *Match* button below the list of named colors causes the color from the list that most closely matches the current color to be used as the current color.

Changes to the color selection take effect immediately in all ParaGraph displays. However, changes are only made permanent if the window is closed using the *OK* button.

Dialog Buttons

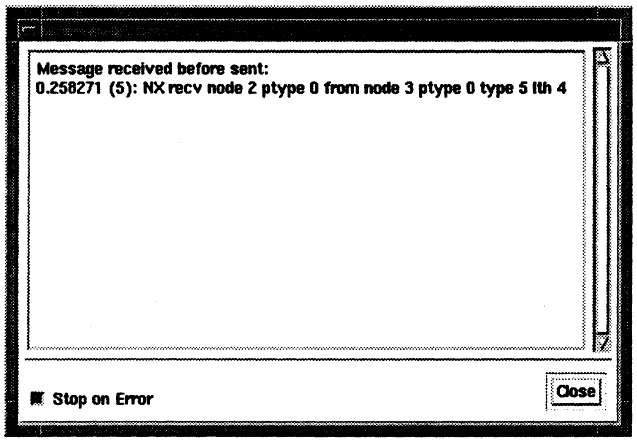
OK	Applies changes and dismisses the dialog.
Reset	Cancels any changes not yet made permanent.
Cancel	Cancels changes and dismisses the dialog.
Help	Displays help text for the dialog.

You can save the color configuration to a file using the File menu Save Layout command. The color definitions stored in the resulting environment file take the form of X resource definitions that you can use to customize ParaGraph's colors permanently.

Apart from enabling you to customize ParaGraph's colors, you can also use the color configuration facility to highlight interesting parts of ParaGraph's displays during the simulation. For example, if you wish to focus on the I/O activity of an application, it is possible to change the busy/overhead/flush and idle colors to the background color of the displays, leaving only the I/O activity for close examination.

Message Log

The Message Log command displays the *Message Log* window. This window is used by ParaGraph to display error and diagnostic messages during the course of the simulation. The default keyboard accelerator for *Message Log* is *Control-M*. Figure 7-8 shows the message log window.



```
Message received before sent:
0.258271 (5): NX recv node 2 ptype 0 from node 3 ptype 0 type 5 lth 4
```

Figure 7-8. Message Log Window

You can configure ParaGraph to stop whenever an error is encountered or to continue with the simulation. This is done using the check button at the bottom of the error log. The *Close* button removes the message log from the screen.

Error Conditions

The following descriptions outline error conditions and the diagnostic and error messages that can appear in the message log.

If the simulation state for a process in the trace has changed illegally, the following message displays in the message log:

```
Unexpected state change
```

This can happen if monitoring is started outside of user code, for example, if the function exit to a message passing function or message passing handler is chosen as the monitor start location.

This can also occur due to incorrect instrumentation of the program that generated the trace. As long as the records that generate this message are "end" records, the message can be safely ignored. If not, ParaGraph displays the following warning at the end of the simulation:

```
"WARNING: Inconsistent state detected -  Visualization results
  may be invalid!"
```

In this case, the visualization results should be treated with caution.

If **traceblockbegin**() and **traceblockend**() statements inserted into your code are not bracketed properly, the following message displays:

```
Incorrectly nested blocks
```

The following message indicates that a message receive was detected before a corresponding send was found. Since event traces on the Paragon system are based on global clock values, this shouldn't happen. However, this condition can occur if instrumentation is incorrect or if messages are flushed after being sent because only part of an application run is traced.

```
Message received before sent
```

The following message displays if the simulation is stopped after a trace record that matches the specification in the trace filter is detected.

```
Record matches trace filter - Simulation stopped
```

The following message displays if the simulation is stopped because the stop time you defined has been reached.

```
Defined stop time reached
```

The following messages display if an attempt is made to open a tracefile that is not in the Paragon trace format.

```
Inconsistent header information
Reader cannot be created for non-SDDF file
```

The following message displays if the number of nodes in the trace exceeds the currently-supported maximum of 512.

```
Number of nodes in trace exceeds maximum of 512
```

The following message displays if an attempt is made to load a layout from an environment file that is in improper format.

```
Layout file line ... : Missing/Unknown ...
```

The following message displays if an attempt to open a file was denied due to missing file permissions.

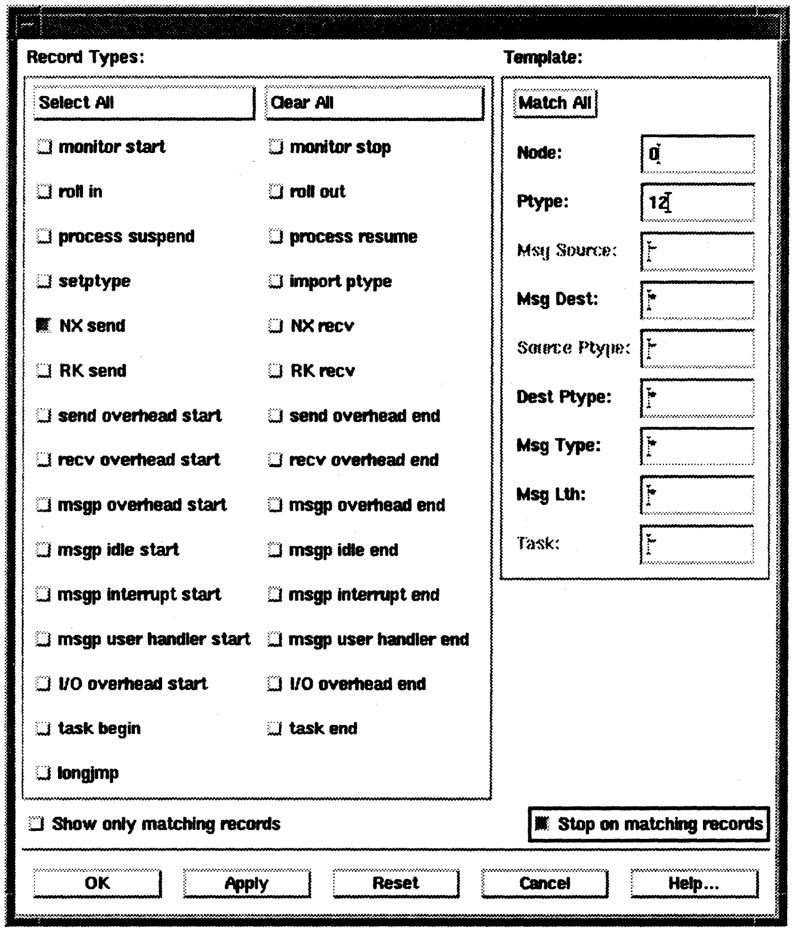
```
Can't open ... for reading/writing
```

The following message displays if the trace is not in ascending order by timestamps.

```
Tracefile not in ascending order by timestamps
```

Trace Filter

The Trace Filter command displays the *Filter Trace Records* dialog. You can use this dialog to select specific trace entries for display in the Trace display and to stop the simulation at specific trace entries. The default keyboard accelerator for *Trace Filter* is *Control-T.* Figure 7-9 shows the *Filter Trace Records* dialog.



Record Types:

Select All | Clear All

☐ monitor start	☐ monitor stop
☐ roll in	☐ roll out
☐ process suspend	☐ process resume
☐ setptype	☐ import ptype
■ NX send	☐ NX recv
☐ RK send	☐ RK recv
☐ send overhead start	☐ send overhead end
☐ recv overhead start	☐ recv overhead end
☐ msgp overhead start	☐ msgp overhead end
☐ msgp idle start	☐ msgp idle end
☐ msgp interrupt start	☐ msgp interrupt end
☐ msgp user handler start	☐ msgp user handler end
☐ I/O overhead start	☐ I/O overhead end
☐ task begin	☐ task end
☐ longjmp	

Template:

Match All

Node:	0
Ptype:	12
Msg Source:	
Msg Dest:	
Source Ptype:	
Dest Ptype:	
Msg Type:	
Msg Lth:	
Task:	

☐ Show only matching records

■ Stop on matching records

OK | Apply | Reset | Cancel | Help...

Figure 7-9. Filter Trace Records Dialog

To use the trace filter, select the record types of interest, specify additional attributes using the *Template*, and specify the actions to be taken. This is particularly useful for focusing on an area of interest in the program for detailed analysis. In Figure 7-9, the simulation stops at the point where the process with ptype 12 on node 0 sends a message to any other process. If such an event is found, the simulation stops and you are notified through a message in the message log. You can then change parameters (to look at the rest of the simulation in more detail) and resume the visualization by pressing the *Resume* button.

Record Types

This field contains toggle buttons you can use to select record types of interest. The field contains one toggle button for every trace record type used by ParaGraph. The *Select All* button selects all record types. The *Clear All* button clears all toggle button selections in the *Record Types* field.

Template

This field contains text fields you can use to specify attributes that further specify the record types of interest. The values allowed are positive integers and the wildcard value "*".

Pressing the *Match All* button causes all the text fields to be filled with the wildcard value. Only those fields that can be used to further specify the record types selected in the *Record Type* field are sensitive. For example, if you press the *task begin* button, the *Node*, *Ptype*, and *Task* fields are sensitive.

Show only matching records

This selection displays in the trace display only the records that match the specification in the Trace Filter. For example, to display only the trace records for node 0, you must enter the value "0" into the *Node* text field and select *Show only matching records.*

Stop on matching records

This selection causes the simulation to stop whenever a trace record that matches the specification in the Trace Filter is encountered.

Dialog buttons

Apply	Makes changes in the dialog effective.
OK	Applies the changes and exits out of the dialog.
Reset	Cancels any changes that have not yet been applied.
Cancel	Cancels changes and exits out of the dialog.
Help	Provides help text for the dialog.

Close All

The Close All command removes all opened ParaGraph displays from the screen. Only displays selected from the *Utilization*, *Communication*, *Task*, and *Other* menus are closed, while dialogs remain displayed. The default keyboard accelerator for *Close All* is *Control-C*.

Utilization, Communication, Task and Other Menus

The *Utilization*, *Communication*, *Task* and *Other* menus allow you to turn the desired displays on and off before the visualization starts or during the course of the visualization. All of these menus are structured in the same way. A check button is provided for each ParaGraph display. Pushing this button brings the corresponding display up, or removes the corresponding display from the screen if it is already being shown. The state of the display is indicated in the menu. Figure 7-10 shows the Utilization menu as an example. In the figure, the Count, Gantt and Summary displays have been selected.

Each of the ParaGraph displays is discussed in detail in the section *ParaGraph Displays*.

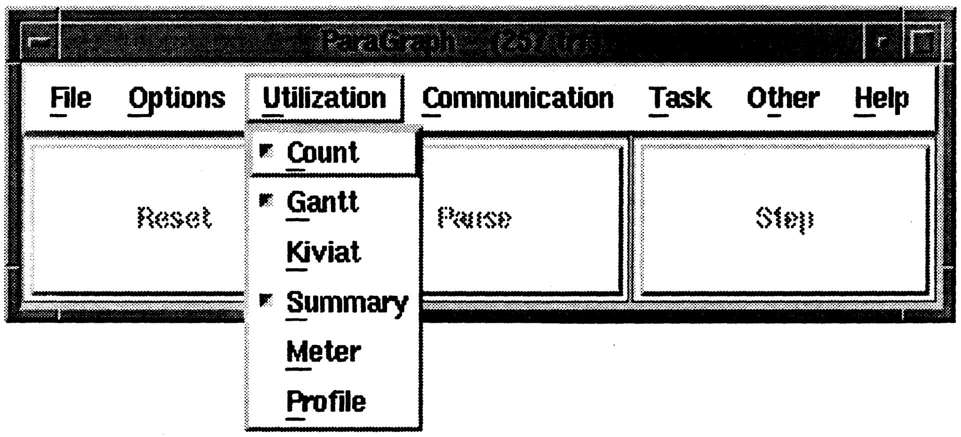


Figure 7-10. Utilization Menu

Utilization Menu

The *Utilization* Menu provides the following buttons: Count, Gantt, Kiviat, Summary, Meter and Profile.

Communication Menu

The *Communication* menu provides the following buttons: Traffic, Spacetime, Queues, Matrix, Meter, Animation, Topology, Network, Node Info and Color code.

Task Menu

The *Task* menu provides the following buttons: Count, Gantt, Status and Summary.

Other Menu

The *Other* menu provides the following buttons: Clock, Trace, Statistics, Processor Status, Phase and Info.

Help Menu

The *Help* menu, available from the main window, provides various levels of help. This menu is always enabled. Through the *Help* menu, you can obtain context-sensitive help and browse through all of the help topics. Figure 7-11 shows the *Help* menu.

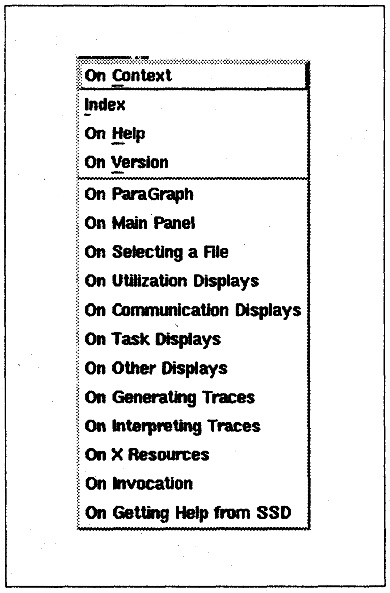


Figure 7-11. Help Menu

On Context

The *On Context* menu item enables context-sensitive help and changes the mouse pointer to a question mark. You can then click on areas of the ParaGraph interface. If there is a help entry for the selected area, ParaGraph displays the help topic text for that feature. You can also obtain context help from the keyboard by pressing the *Help* key (typically mapped to F1) while the keyboard focus is directed to the desired interface feature.

Index

Displays the *Index* dialog. All help topics for ParaGraph are listed in the *Index* dialog.

On Help

Displays the help topic text that explains how to use all of the aspects of help.

On Version

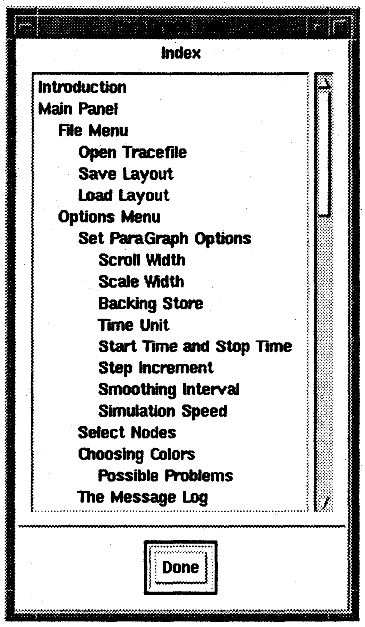
Displays a dialog containing ParaGraph version information.

Additional Topics

The names of all the major ParaGraph help topics follow the *On Version* entry on the *Help* menu. When you select a topic, ParaGraph displays help text for the selected topic.

Help Index

ParaGraph displays the *Index* dialog when you choose the *Index* menu item from the *Help* menu. The help index contains all the topics and sub-topics of help available for ParaGraph. Sub-topics are indented underneath major topics. Figure 7-12 shows the *Index* dialog.


Index

Introduction
Main Panel
File Menu
Open Tracefile
Save Layout
Load Layout
Options Menu
Set ParaGraph Options
Scroll Width
Scale Width
Backing Store
Time Unit
Start Time and Stop Time
Step Increment
Smoothing Interval
Simulation Speed
Select Nodes
Choosing Colors
Possible Problems
The Message Log

Figure 7-12. Help Index

Help Topics

This field contains a scrollable list of all ParaGraph help topics. Select a topic from the list to display the help text for that topic.

Done

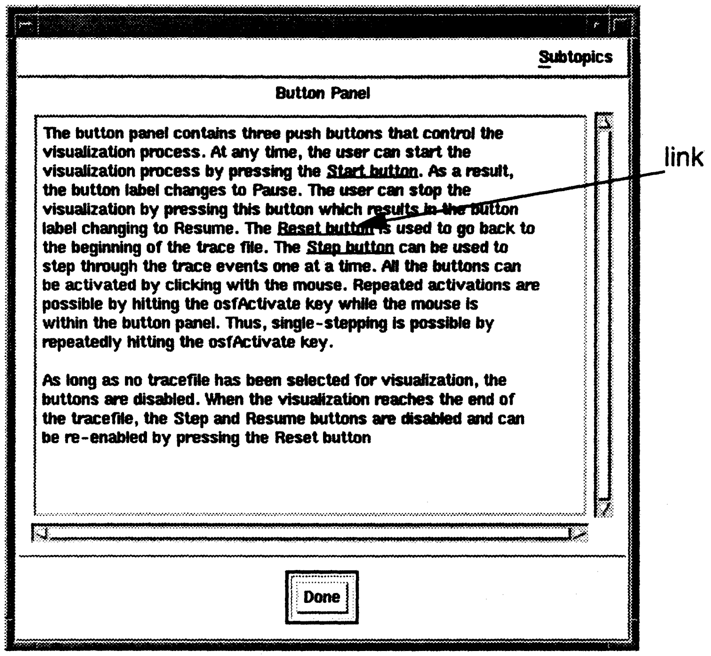
Select *Done* to dismiss the help index dialog.

Help Topic

The help topic dialog appears when you request help for a particular topic. You can select a topic by any of the following:

- selecting the topic in the help *Index* dialog
- selecting a dialog's *Help* button
- using context-sensitive help
- selecting a major topic from the *Help* menu.

Figure 7-13 shows a help topic dialog.



Button Panel

The button panel contains three push buttons that control the visualization process. At any time, the user can start the visualization process by pressing the Start button. As a result, the button label changes to Pause. The user can stop the visualization by pressing this button which results in the button label changing to Resume. The Reset button is used to go back to the beginning of the trace file. The Step button can be used to step through the trace events one at a time. All the buttons can be activated by clicking with the mouse. Repeated activations are possible by hitting the osfActivate key while the mouse is within the button panel. Thus, single-stepping is possible by repeatedly hitting the osfActivate key.

As long as no tracefile has been selected for visualization, the buttons are disabled. When the visualization reaches the end of the tracefile, the Step and Resume buttons are disabled and can be re-enabled by pressing the Reset button

Figure 7-13. Help Topic Dialog

Subtopics

If a topic has subtopics, the help topic dialog contains a *Subtopics* pull-down menu. All of the subtopics are included on the menu. Select an item from the *Subtopics* menu to display help text for the subtopic.

Help Title

The title identifies which topic the help text describes.

Help Text

This field contains the help text for the topic. You can select the text and paste it into other clients.

Items that are underlined in the text represent links to other help topics. Clicking on an underlined text entry displays the help topic dialog for that entry.

Done

Use the *Done* button to dismiss the help topic dialog.

ParaGraph displays

Each of the check buttons in the Utilization, Communication, Task and Other menus brings up the corresponding ParaGraph display. The following sections discuss each of the different ParaGraph displays:

Utilization Displays

The utilization displays can be used to analyze busy, idle, flush, I/O, and overhead times for an application.

Idle time

A processor is categorized as idle if the process executing on that processor has blocked or if it has ceased execution at the end of the run. Examples for conditions that lead to idle times are:

- the process is blocked due to a blocking message passing operation (e.g. csend, crecv).
- the process has been suspended (for example, due to a flick() call).
- the process has not started executing or has stopped executing at the end of the run.

Overhead time

Overhead time is the time that the application process spends executing in the communication subsystem. System overhead (i.e. time that is spent by the operating system) is not measured separately. For example, an asynchronous message passing operation such as isend() would lead to overhead time. A synchronous message passing operation such as crecv() leads to overhead time, which is potentially followed by idle time (in the case that the operation leads to a blocking condition) followed by another portion of overhead time (when the operation completes).

I/O time

I/O time is the time that the application process spends executing file I/O calls.

Flush time

The performance monitoring library allocates a buffer in which the trace events generated by the application process are stored. When this buffer is full, the performance monitoring library flushes the buffer to an event trace server. Depending on the size of the buffer and the type of application, this can cause major perturbations to the application. In order for you to be able to judge whether or not the application was perturbed by buffer flushing, the time spent flushing the buffers is visualized as a separate state.

If the flush time dominates the application's behavior, this is an indication that the trace should be regenerated using larger buffers. Since flush time is essentially idle time for the application, the flush color is drawn next to the idle color in the legends of the utilization displays.

Busy time

A processor is categorized as busy if it is neither in the idle nor in the overhead or I/O states.

The following sections describe the utilization displays.

Count

Figure 7-14 shows the Count display

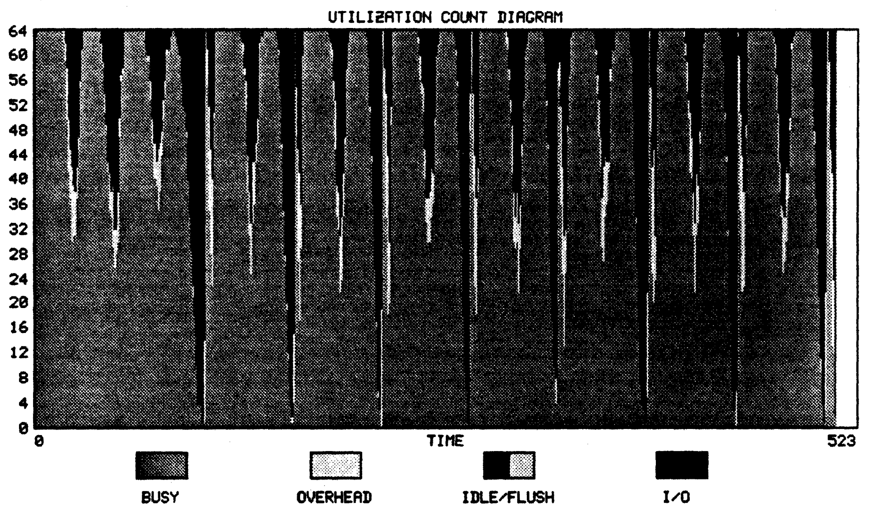


Figure 7-14. Count Display

This display gives a scrolling display of the total number of processors in each of the five states as a function of time. The number of processors is on the vertical axis and time is on the horizontal axis, which scrolls as necessary as the simulation proceeds. The default color scheme used is borrowed from traffic signals: green (go) for busy, yellow (caution) for overhead, brown for I/O, grey for flush, and red (stop) for idle. By convention, green is shown at the bottom, yellow and brown in the middle and grey/red at the top along the vertical axis.

Gantt

Figure 7-15 shows the Gantt display.

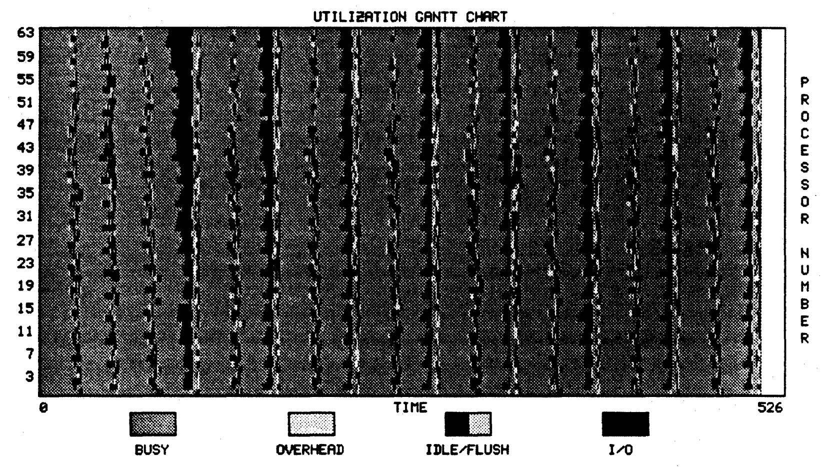


Figure 7-15. Gantt Display

This display shows the activity of individual processors by a horizontal bar chart in which the color of each bar indicates the busy/overhead/I-O/flush/idle status of the corresponding processor as a function of time, again using the traffic-signal color scheme. Processor number is on the vertical axis and time is on the horizontal axis, which scrolls as necessary as the simulation proceeds. The Gantt chart provides the same basic information as the Utilization Count display, but on an individual processor, rather than aggregate, basis.

Kiviat

Figure 7-16 shows the Kiviat display.

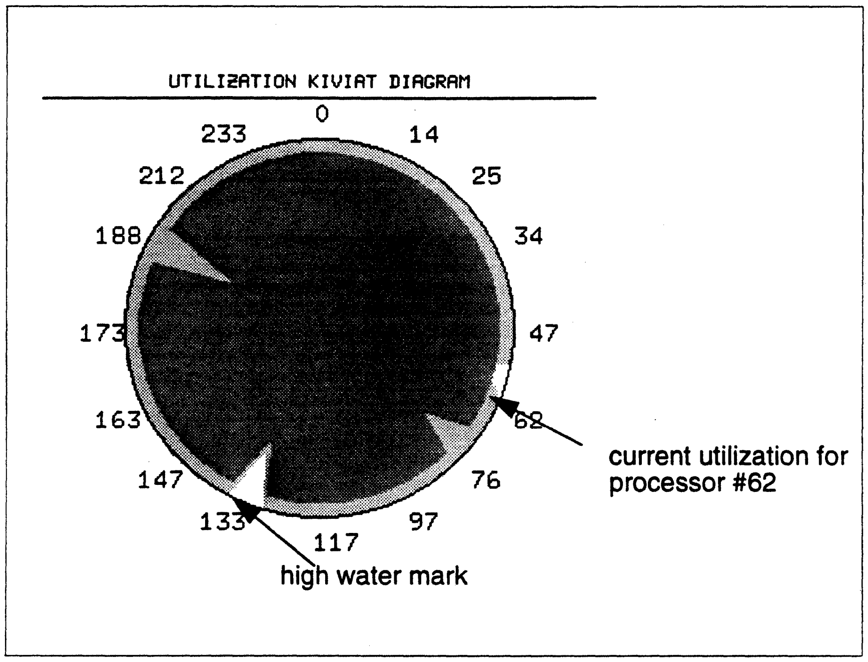


Figure 7-16. Kiviat Display

This display gives a geometric representation of individual processor utilization and overall load balance. Each processor is depicted as a spoke of a wheel. The recent average fractional utilization (computed over the time interval chosen as Smoothing Interval) of each processor portrays a point on the spoke, with the hub of the wheel representing zero (completely idle) and the edge of the spoke representing one (completely busy). Taken together all these points on the spoke determine the vertices of a polygon whose size and shape indicates the load balancing and utilization of the system. There is also a high water mark indicating the maximum utilization so far achieved. Low utilization causes the polygon to be concentrated near the center, while high utilization causes the polygon to lie near the perimeter. Poor load balance across processors causes the polygon to be strongly skewed or asymmetric.

The current utilization is shown in dark shading, while the high water mark seen thus far is shown in lighter shading. The current utilization used in this diagram is in fact a moving average over a time interval of user-specified width (the Smoothing Interval) since instantaneous utilization would of course always be either zero or one for each processor.

At the end of the run, the display shows the average utilization computed over the entire run along with the high water mark.

Summary

Figure 7-17 shows the Summary display.

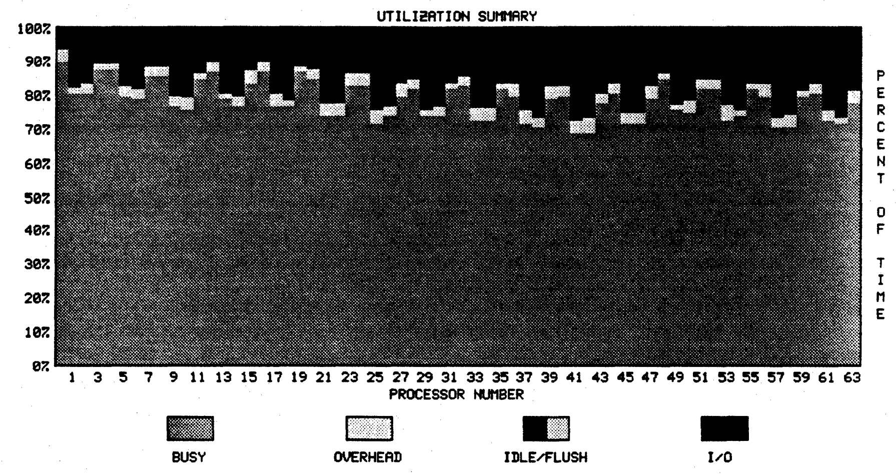


Figure 7-17. Summary Display

This display shows the cumulative percentage of time, over the entire run, that each processor spent in each of the five busy/overhead/I-O/flush/idle states. The percentage of time is shown on the vertical axis and the processor number on the horizontal axis. Again, the green/yellow/brown/grey/red color scheme is used to indicate the four states. In addition to giving a visual impression of the overall efficiency of the parallel program, this display also gives a visual indication of the load balance across processors. At the end of the run, this display shows the summary values for the entire run.

Meter

Figure 7-18 shows the Meter display.

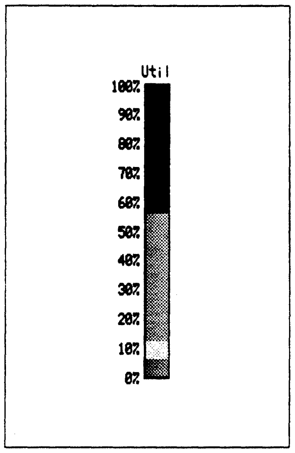


Figure 7-18. Meter Display

This display uses a colored vertical bar, with the usual green/yellow/brown/grey/red color scheme, to indicate the percentage of the total number of processors that are currently in each of the five busy/overhead/I-O/flush/idle states. The visual effect is similar to that of a thermometer or some automobile speedometers. This display provides essentially the same information as the Utilization Count display but saves screen space by changing in place rather than scrolling with time.

Profile

Figure 7-19 shows the Profile display.

For each possible number of processors k, $0 \leq k \leq p$, where p is the maximum number of processors for this run, this display shows the percentage of time during the run that exactly k processors were in a given state (i.e., busy/overhead/I-O/flush/idle). The percentage of time is shown on the vertical axis and the number of processors k is shown on the horizontal axis. The profile for each possible state is shown separately, and you can cycle through the five states by clicking the mouse on the *Option* Menu. This display is defined only at the end of the run.

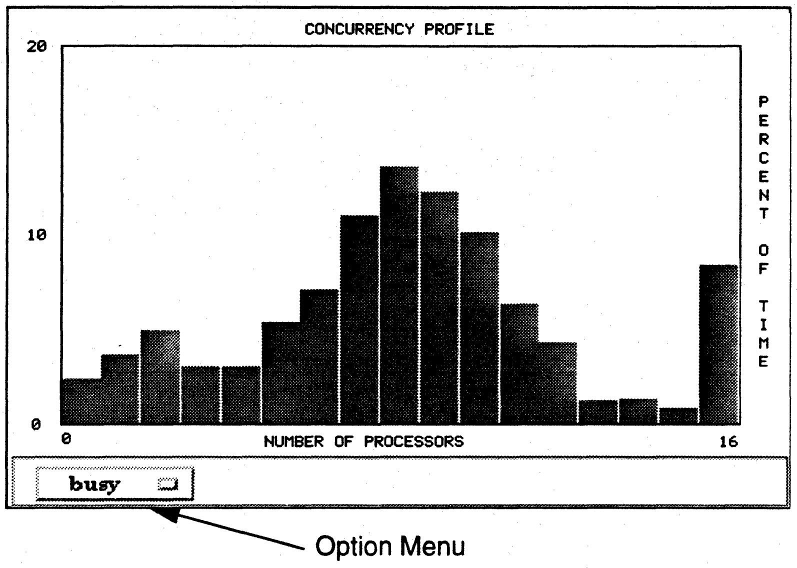


Figure 7-19. Profile Display

Communication displays

The communication displays show message-passing behavior for an application. The PICL programming model supported by the original version of ParaGraph consists of non-blocking send operations and blocking as well as non-blocking receive operations. For the Paragon, this model has to be extended, because a send operation may also lead to a blocking condition (e.g. through a synchronous operation like msgwait). Other operations that have to be supported are the NX probe operations. The following sections describe the communication displays.

Traffic

Figure 7-20 shows the Traffic display.

This display is a simple plot of the total communication traffic in the interconnection network (including message buffers) as a function of time. The curve plotted is the total of all messages that are currently pending (sent but not yet received), and can be optionally expressed either by message count or by volume in bytes (on a per-node basis). The communication traffic shown can also optionally be either the aggregate over all processors or only the messages pending for any individual processor you select. Message volume or count is shown on the vertical axis, and time is shown on the horizontal axis, which scrolls as necessary.

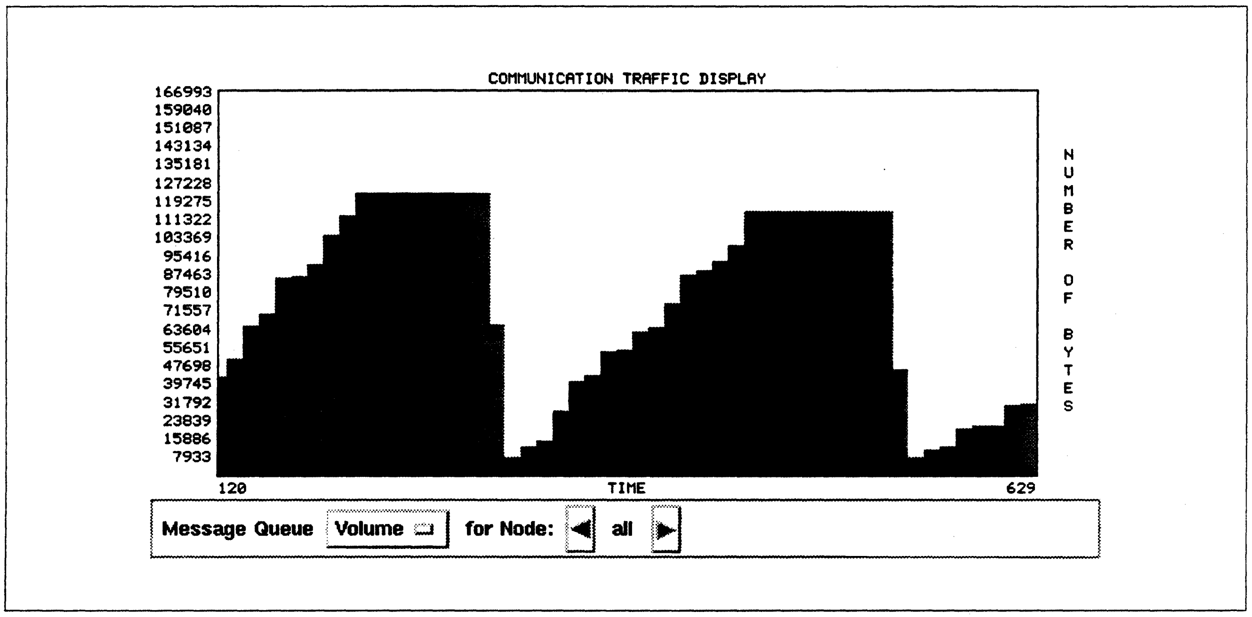


Figure 7-20. Traffic Display

The display type to be used (message volume or message count) is selected through an options menu. The node to be used (all processors or some individual processor) can be selected using the arrows to the right of the options menu. Clicking on an arrow once selects the next or previous node from the nodes that are being visualized. Clicking and holding the arrow advances through the available nodes quickly. The default is to show the aggregate over *all* processors.

Spacetime

Figure 7-21 shows the Spacetime display.

This display shows communication behavior on a per-processor basis. The processor number is on the vertical axis, and time is on the horizontal axis, which scrolls as necessary as time proceeds. Processor activity (running/blocked) is indicated by horizontal lines, one for each processor, with the line drawn in the color that corresponds to the state the processor is in (as seen in the Utilization displays). Messages between processors are depicted by slanted lines between the sending and receiving processor activity lines, indicating the times at which each message was sent and received. These sending and receiving times are from user process to user process (not simply the physical transmission time), and hence the slopes of the resulting lines give a visual indication of how soon a given piece of data produced by one process was needed by the receiving process. If a communication between two nodes on the same processor occurs, this is indicated by drawing an arc instead of a line.

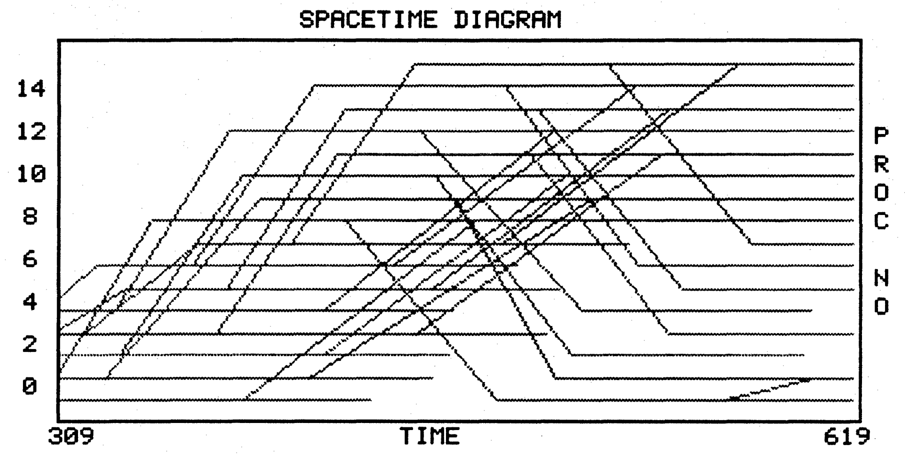


Figure 7-21. Spacetime Display

The communication lines are color coded according to the Color Code display (i.e. the color is determined either by the message length, the message type or the number of hops the message has to travel to reach its destination (see page 7-44)). Each message line is drawn when its receive time has been reached, so this display may appear to be "behind" other displays that depict messages as soon as the send event is encountered.

Queues

Figure 7-22 shows the Queues display.

This display depicts the size of the queue of incoming messages for each processor by a vertical bar whose height varies with time as messages are sent, buffered, and received. The processor number is shown on the horizontal axis. At your option, the queue size can be measured either by the number of messages or by their total length in bytes. The input queue size for a given processor is incremented each time a message is sent to that processor, and decremented each time the user process on that processor receives a message. As before, dark shading depicts the current queue size on each processor and lighter shading indicates the high water mark seen so far. The Message Queue display gives a pictorial indication of whether there is communication congestion in a parallel program (i.e. whether messages are accumulating in the input queue) or the messages are being consumed at about the same rate they arrive.

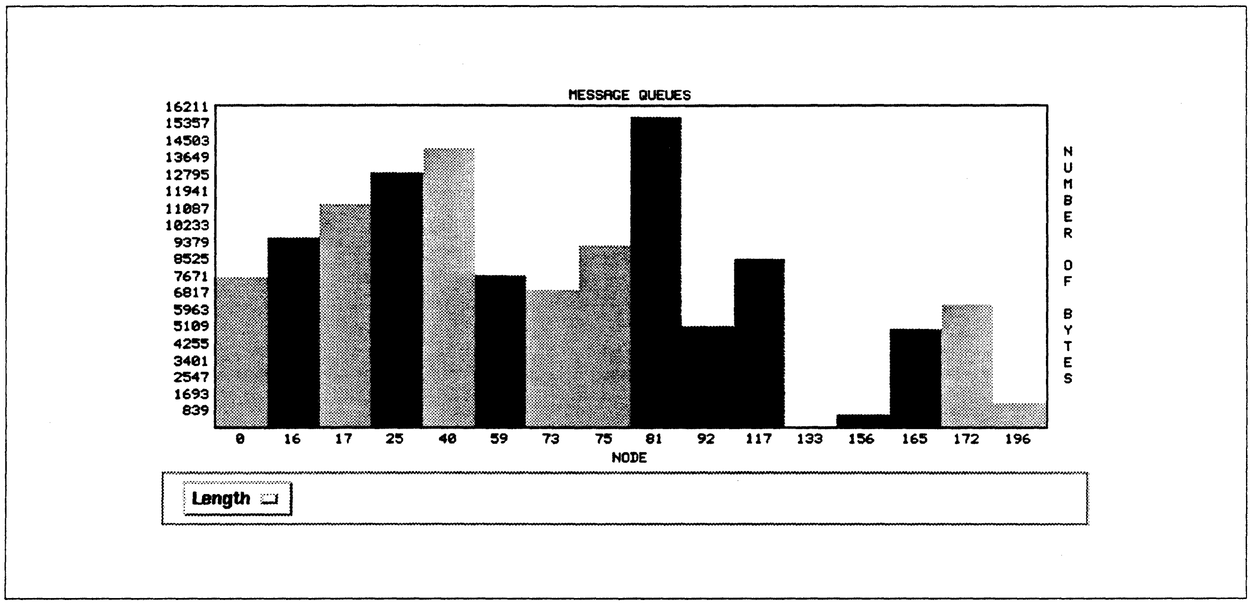


Figure 7-22. Queues Display

Matrix

Figure 7-23 shows the Matrix display.

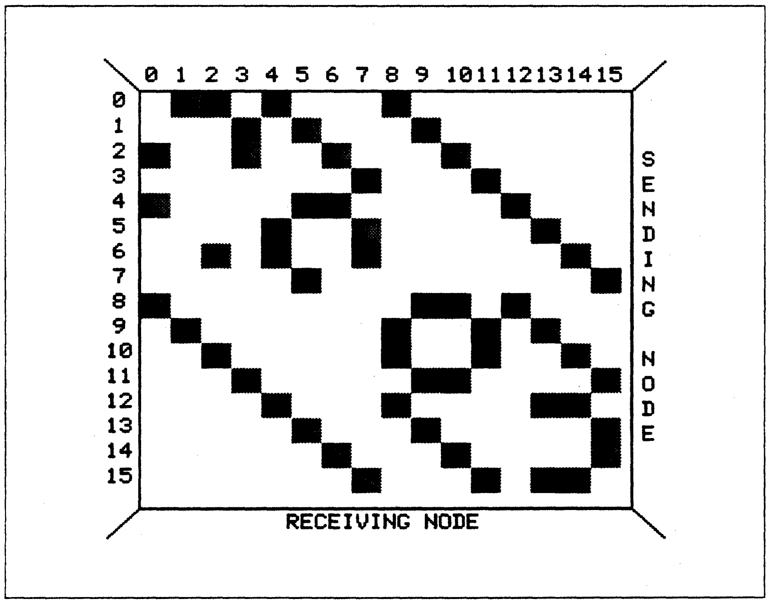


Figure 7-23. Matrix Display

In this display, messages are represented by squares in a two-dimensional array whose rows and columns correspond to the sending and receiving processors, respectively, for each message. During the simulation, each message is depicted by coloring the appropriate square at the time the message is sent, and erasing it at the time the message is received. The color used indicates the size of the message in bytes, as given in the separate Color Code display (see page 7-44) that can also be selected from the menu. Thus, the sizes, durations, and overall pattern of messages are depicted by this display. At the end of the simulation, the Communication Matrix display shows the cumulative communication volume for the entire run between each pair of processors.

Communication Meter

Figure 7-24 shows the Communication Meter display.

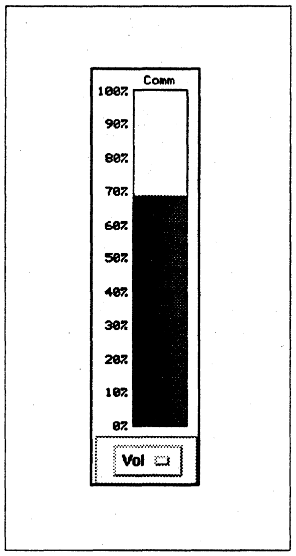


Figure 7-24. Communication Meter Display

This display uses a vertical bar to indicate the percentage of maximum communication volume (or number of messages) currently pending (i.e. sent but not yet received). This display provides essentially the same information as the Communication Traffic display, but saves screen space (which may be needed for other displays) by changing in place rather than scrolling with time. Conceptually, this thermometer-like display is similar to the Utilization Meter display, except that it shows communication instead of utilization. The two are interesting to observe side by side.

Animation

Figure 7-25 shows the Animation display.

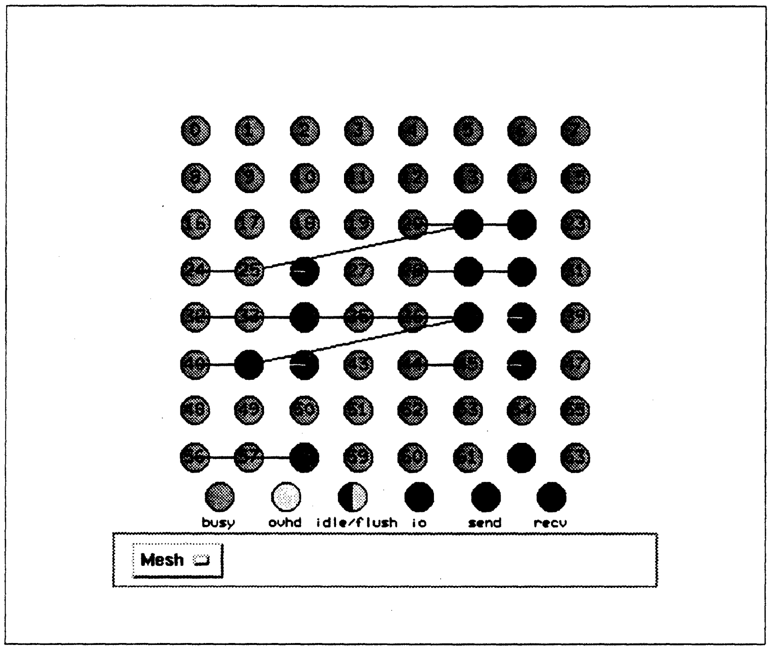


Figure 7-25. Animation Display

In this display, the Paragon system is represented by a graph whose nodes (depicted by numbered circles) represent processors, and whose arcs (depicted by lines between the circles) represent communication links. The nodes in the graph can either be arranged in a circle or in a mesh layout.

The status of each node (busy, overhead, idle, flush, I/O, sending, receiving) is indicated by its color. The sending and receiving states are states that would also be shown as overhead in the Utilization displays. Thus the overhead state in this display is reserved for situations where message passing overhead can not be attributed to a send or receive operation (e.g. message passing overhead produced by a probe operation).

An arc is drawn between the source and destination processors when a message is sent, and erased when the message is received. Thus, both the colors of the nodes and the connectivity of the graph change dynamically as the simulation proceeds. The arcs represent the logical, rather than physical, connectivity of the Paragon network, and possible routing of messages through intervening nodes is not depicted unless the program being visualized does such forwarding explicitly.

Various combinations of states are possible for the sending and receiving processors. For example, both processors could be busy, one having already sent the message and resumed computing, while the other has not yet stopped computing to receive it. Upon conclusion, this display shows a summary of all (logical) communication links used throughout the run.

Topology

Figure 7-26 shows the Topology display.

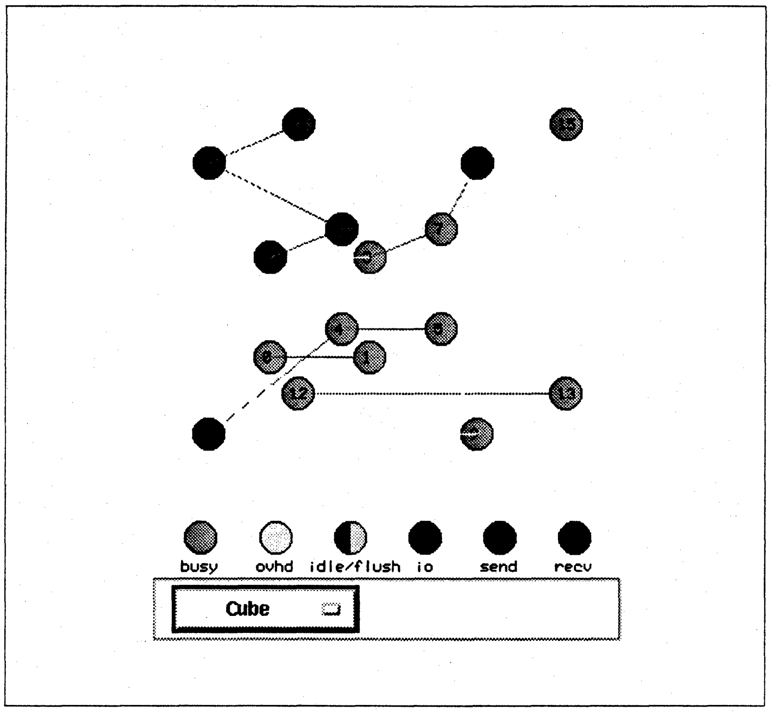


Figure 7-26. Topology Display

This display is similar to the Animation display, except that it provides a number of additional layouts for the nodes in order to exhibit more clearly communication patterns corresponding to various logical communication topologies. The layouts provided include cube, lateral cubes, nested cubes, squares, pinwheel, polytope, tesseract, tree, gem, quatrefoil, rosette, circles, grid, mesh, torus, orbits, ring of rings, web, ring, crosshatch, linear and shuffle arrangements.

The scheme for coloring nodes and drawing arcs is the same as that for the Animation display, except that curved arcs are often used to avoid, as much as possible, intersecting intermediate nodes. If the actual number of nodes is not a power of two, then any "unused" nodes in the selected layout are indicated by black shading. Upon conclusion, this display shows a summary of all (logical) communication links used throughout the run. This display is limited to 16 nodes.

Network

Figure 7-27 shows the Network display.

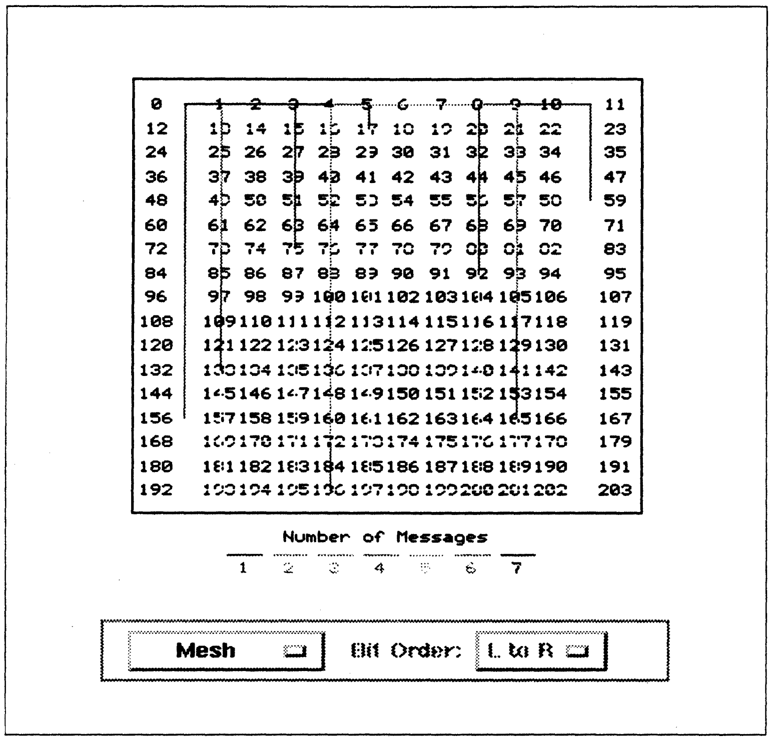
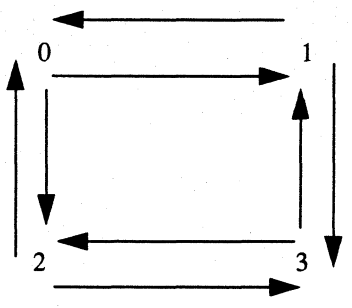


Figure 7-27. Network Display

This display depicts interprocessor communication in terms of various network topologies. Unlike the Animation and Hypercube displays, the Network display shows the actual path that each message takes, which may include routing through one or more intermediate nodes between the original source and ultimate destination. Depicting message routing through a network requires a knowledge of the interconnection topology. The default configuration shows the Paragon's mesh and takes into account the physical layout of the system and partition the application ran on to correctly depict the message routing on the Paragon.

The following configuration represents the physical links of the Paragon network separately. The layout corresponds to that used on the Paragon front panel. The upper horizontal line shows messages flowing from right to left, the lower line shows messages flowing from left to right. Similarly, the left vertical line shows upward message flow, while the right line shows downward message flow. The routing is as follows:



Note that the node numbers used in the network display are *physical* node numbers. These are not necessarily the same as the *logical* node numbers used in the other ParaGraph displays. For example, if the logical nodes 0 and 1 are mapped to physical nodes 17 and 33, a communication between these nodes will be shown as a communication between nodes 0 and 1 in the Animation display but the Network display will show a communication between nodes 17 and 33.

In addition you can select from among several of the most common interconnection networks, each of which may also have a choice of routing schemes. Thus, one might want to choose a different network deliberately in order to get some idea how a program that ran on the Paragon might perform on a different topology. Thus, for example, you can see a visual simulation of the behavior your program might have on a Thinking Machines CM-5 (quadtree). The routing scheme is chosen using the Bit Order option menu (left to right or right to left). If the current network choice does not support a routing scheme, this menu is insensitive. Some of the available topologies are represented as multistage networks, with duplicate sets of source and destination nodes, between which are several "stages" of nodes or switches through which intermediate routing occurs. Networks depicted in this manner include gray code, butterfly, hypercube, omega, baseline, and crossbar. Other available topologies are represented by a single set of nodes that serve as both sources and destinations, with messages moving in either direction through the network. Networks depicted in this manner include binary tree, quadtree, and mesh.

Each physical link in the network is color coded according to the number of messages currently sharing that link. A temperature-like color code is used, so that "hot spots" appear red while less heavily used links appear blue. In monochrome, the message count on a link is indicated instead by the line width, so that, for example, the tree networks look like "fat" trees, as the message count tends to be higher nearer the root.

Unlike the Animation or Hypercube displays, in the Network display the sending or receiving of a message does not always cause the drawing or erasure of a given link, but often merely changes its color to be one step hotter or cooler than it was previously. A given message may use several links, causing each link to be incremented or decremented separately. On conclusion, the coloring of the links indicates the cumulative message count over the entire run, and the color-code legend is recalibrated accordingly to indicate the range of cumulative totals for the various links. Note that for the cumulative totals to be displayed, the display must be opened during the entire length of the simulation and the network type may not be changed during the course of the simulation.

Node Info

Figure 7-28 shows the Node Info display.

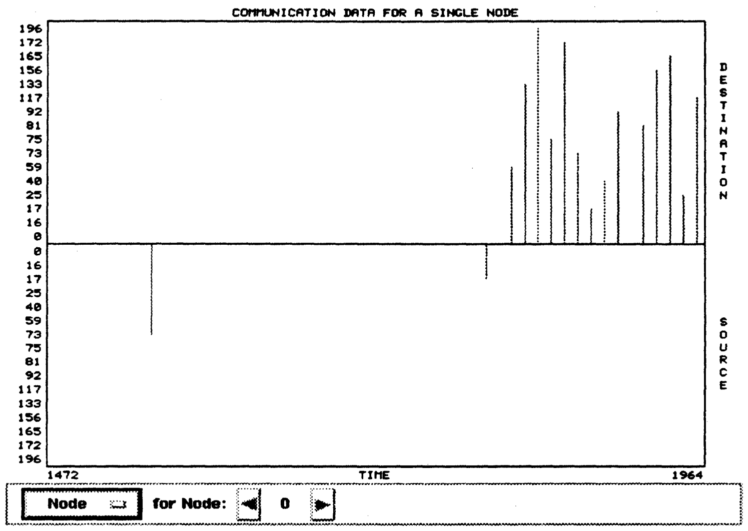


Figure 7-28. Node Info Display

This display provides, in graphical form, detailed communication statistics for a single, user-selected node. The choices of statistics plotted are the source/destination, type, length, and distance traveled for all messages sent to or from the chosen processor. Time is on the horizontal axis, and the chosen statistic is on the vertical axis, with incoming and outgoing messages shown in separate windows. This display is helpful in analyzing communication behavior in detail, especially in perceiving trends or patterns in the communication structure that improve understanding of program behavior and performance. As in the Communication Traffic display, the display type can be chosen from the options menu at the bottom of the display and the node to be visualized can be chosen using the arrows to the right of the options menu.

If several sends or receives occur at the same simulation time unit, only one line is drawn in the default case. This may be changed by selecting a scale value that is larger than two in the Configure dialog in which case every single send and receive is indicated by drawing a cross or horizontal line at the position of the sending or receiving node/message type in addition to the vertical line.

Color Code

Figure 7-29 shows the Color Code display.

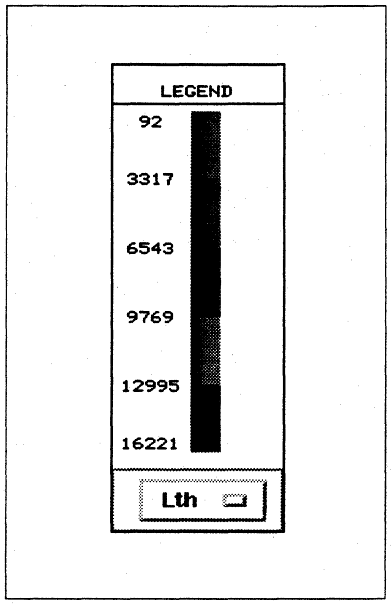


Figure 7-29. Color Code Display

This display permits you to select which statistic determines the color code for coloring the messages in the Spacetime and Communication Matrix displays. The color code can be chosen using the option button at the bottom of the display. The choices include the size of the message in bytes (Lth), the distance between the source and destination nodes (Dist) and the message type (Type). Clicking on the option menu causes the resulting color code to be displayed and the colors to be used in the Spacetime and Communication Matrix displays. The Node Info display uses the same color coding to draw lines but it has an additional option menu to select the color coding choice. The message length in the color code (Lth) changes at the end of the simulation to color code the total message volume over the entire run which is displayed in the Communication Matrix.

Task displays

The task displays use information you provide to depict the portion of your parallel program that is executing at any given time. Specifically, you define "tasks" within the program by using special routines to mark the beginning and ending of each task and assigning the task a task number.

The scope of what is meant by a task is left entirely to you: a task can be a single line of code, a loop, an entire subroutine, or any other unit of work that is meaningful in a given application. For example, in matrix factorization one might define the computation of each column to be a task, and assign the column number as the task number. Tasks are defined simply by calling the **traceblockbegin**() and **traceblockend**() routines, with the desired task number as argument, immediately before and after the selected section of code as shown in the following example:

```
for (i=0; i<ITER; i++) {
  traceblockbegin(i);

    code section

  traceblockend(i);
}
```

The **traceblockbegin** and **traceblockend** routines are part of the performance monitoring library that is linked with the application by default. They cause the performance monitoring subsystem to produce event records that are interpreted appropriately by ParaGraph to depict the given task, using displays described in this section. If the tracefile contains no event records defining tasks, the task displays will simply be blank, but the remaining displays in ParaGraph will still show their normal information.

Tasks can be nested, one inside another, but if so these should be properly bracketed by matching task begin and end records. More than one processor can be assigned the same task (or, more accurately, each processor can be assigned its own portion of the same task); indeed, the model supported is that all processors collaborate on each task, rather than that each task is assigned to a single processor. In many contexts, such as the matrix example mentioned above, there is a natural ordering and corresponding numbering of the tasks in a parallel program.

In most of the task displays described below, the task numbers are indicated by a color coding. Since the number of tasks may be larger than the number of colors that can be easily distinguished, a limited number of colors (64) is used and the colors are "recycled" to depict successive task numbers. To aid in distinguishing consecutively numbered tasks ParaGraph strides through these 64 colors in groups of eight rather than in strict rainbow sequence. You can override these default task colors by using the Color command in the Options menu or by setting the appropriate X resources.

The following sections describe the Task displays.

Task Count

Figure 7-30 shows the Task Count display.

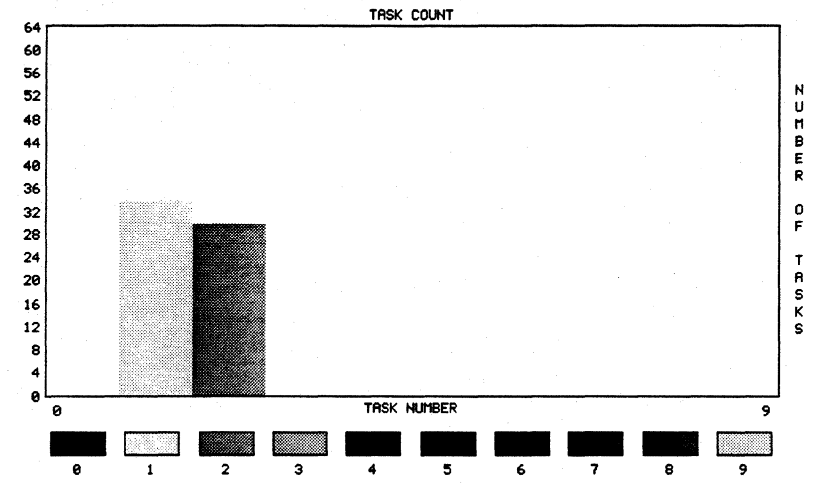


Figure 7-30. Task Count Display

During the simulation, this display shows the number of processors that are executing a given task at the current time. The number of tasks is shown on the vertical axis and the task number is shown on the horizontal axis. At the end of the run, this display changes to show a summary over the entire run. Specifically, it shows the average number of processors that were executing each task over the lifetime of that process (i.e., the time interval starting when the first processor began executing the task and ending when the last processor finished the task).

Task Gantt

Figure 7-31 shows the Task Gantt display.

This display depicts the task activity of individual processors by a horizontal bar chart in which the color of each bar indicates the current task being executed by the corresponding processor as a function of time. Processor number is on the vertical axis and time is on the horizontal axis, which scrolls as necessary as the simulation proceeds. This display can be compared with the Utilization Gantt chart to correlate busy/overhead/I-O/flush/idle status with the task information.

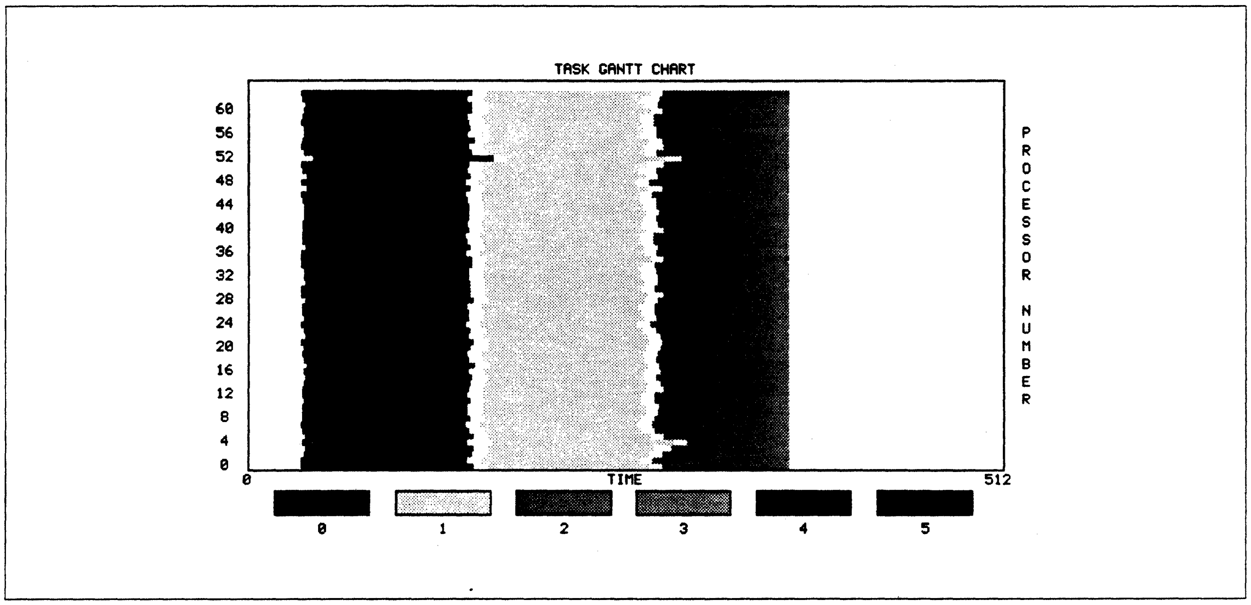


Figure 7-31. Task Gantt Display

Task Status

Figure 7-32 shows the Task Status display.

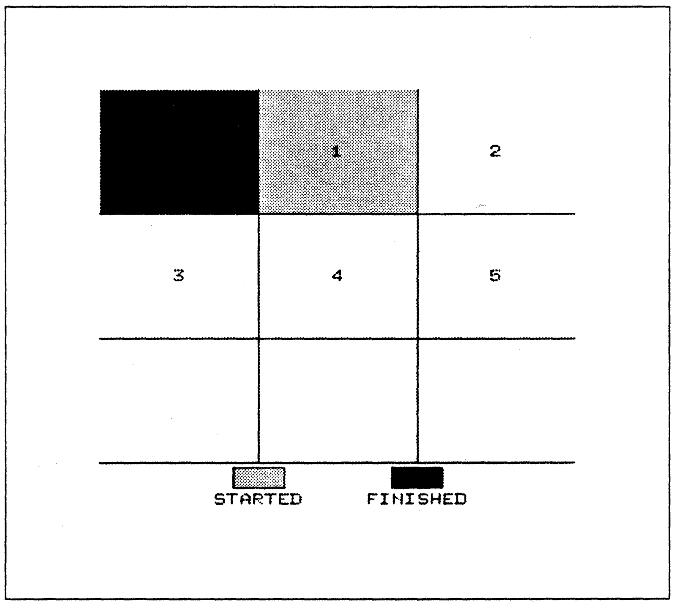


Figure 7-32. Task Status Display

In this display the tasks are represented by a two-dimensional array of squares, with task numbers filling the array in row-wise order. Initially, all of the squares are white. As each process type is scheduled, its corresponding square is lightly shaded to indicate that the task is now in progress. When a task is subsequently finished, its corresponding square is then darkly shaded.

Task Summary

Figure 7-33 shows the Task Summary display.

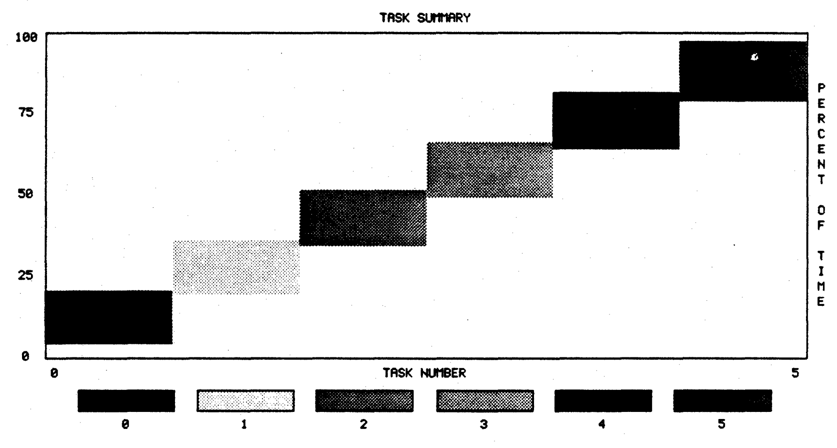


Figure 7-33. Task Summary Display

This display, which is defined only at the end of the simulation run, indicates the duration of each task (from earliest beginning to last completion by any processor) as a percentage of the overall execution time of the parallel program, and furthermore places the duration interval of each task within the overall execution interval of the parallel program. The percentage of the total execution time is shown on the vertical axis, and the task number is shown on the horizontal axis.

Other Displays

The following sections describe the other displays available with ParaGraph.

Clock

Figure 7-34 shows the Clock display.

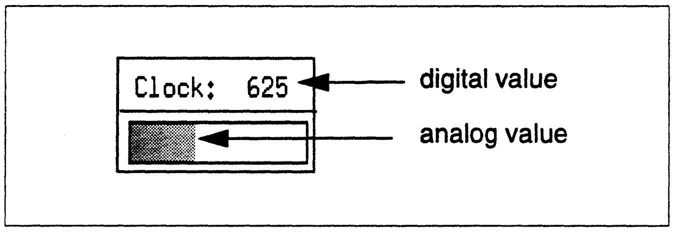


Figure 7-34. Clock Display

This display provides both digital and analog clock readings during the graphical simulation of the parallel program. The current simulation time is shown as a numerical reading, and the proportion of the full tracefile that has been completed thus far is shown by a colored horizontal bar. The clock reading is updated synchronously with the other displays, and it ticks through all integral time values, not just those that happen to come from event timestamps.

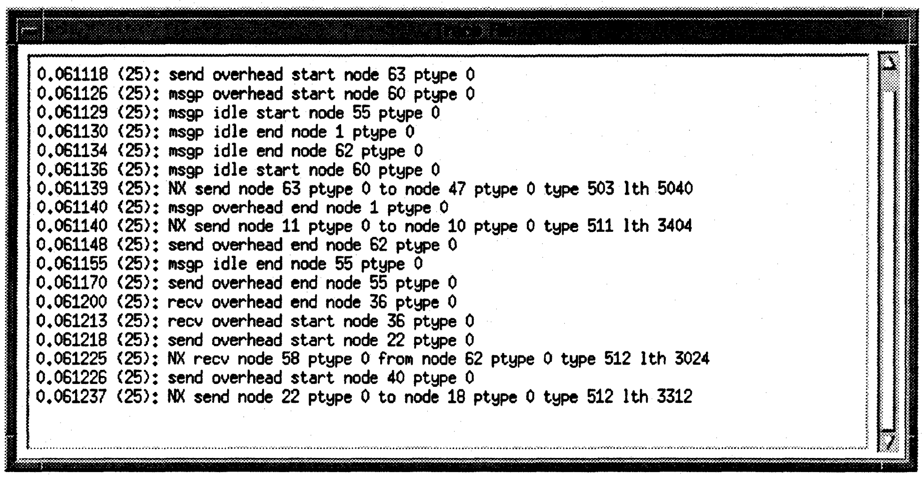
The relationship between simulation time units and real time is explained in the section *Time Unit* on page 7-14.

Trace

Figure 7-35 shows the Trace display.

This is a non-graphical display that prints an annotated version of each trace event as it is read from the tracefile. It is primarily useful in the single-step mode for debugging or other detailed study of the parallel program on an event-by-event basis.

Although the trace records are drawn in this display one at a time, space is allowed to show several consecutive trace records, and the display scrolls vertically as necessary with time. A scrollbar is provided at the left of the display. Trace events are printed into the window if the display is opened and the contents of the window are deleted whenever the display is closed



```
0.061118 (25): send overhead start node 63 ptype 0
0.061126 (25): msgp overhead start node 60 ptype 0
0.061129 (25): msgp idle start node 55 ptype 0
0.061130 (25): msgp idle end node 1 ptype 0
0.061134 (25): msgp idle end node 62 ptype 0
0.061136 (25): msgp idle start node 60 ptype 0
0.061139 (25): NX send node 63 ptype 0 to node 47 ptype 0 type 503 lth 5040
0.061140 (25): msgp overhead end node 1 ptype 0
0.061140 (25): NX send node 11 ptype 0 to node 10 ptype 0 type 511 lth 3404
0.061148 (25): send overhead end node 62 ptype 0
0.061155 (25): msgp idle end node 55 ptype 0
0.061170 (25): send overhead end node 55 ptype 0
0.061200 (25): recv overhead end node 36 ptype 0
0.061213 (25): recv overhead start node 36 ptype 0
0.061218 (25): send overhead start node 22 ptype 0
0.061225 (25): NX recv node 58 ptype 0 from node 62 ptype 0 type 512 lth 3024
0.061226 (25): send overhead start node 40 ptype 0
0.061237 (25): NX send node 22 ptype 0 to node 18 ptype 0 type 512 lth 3312
```

Figure 7-35. Trace Display

The individual entries printed into the trace display have the following format:

<event time> <simulation time>: <event name> <event parameters>

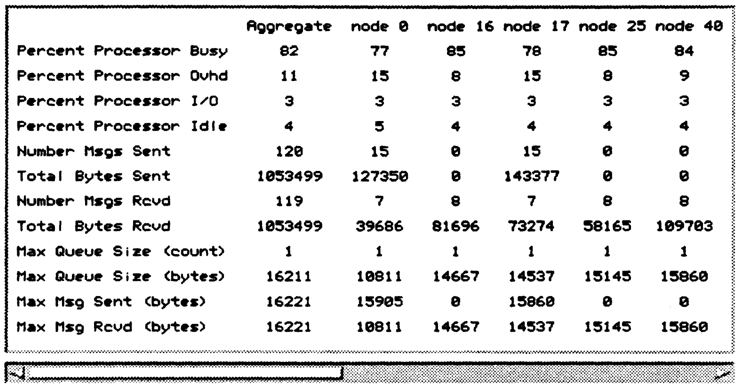
Thus, the last line in the above figure indicates that at time 0.061237 seconds (corresponding to simulation time unit 25) the ptype 0 process on node 22 sent a message of type 512 and length 3312 to the ptype 0 process on node 18.

Note that updating the trace display for every trace record is very expensive and slows the simulation down considerably. Thus, the trace display should mostly be used in single-step mode or in conjunction with the "Trace Filter" option (see "Trace Filter" on page 7-21).

Statistical Summary

Figure 7-36 shows the Statistical Summary display.

This is a non-graphical display that gives numerical values for various statistics summarizing processor utilization and communication, both for individual processors and aggregates over all processors. A scrollbar is provided at the bottom of the display to let you scroll through the values displayed in the window.



	Aggregate	node 0	node 16	node 17	node 25	node 40
Percent Processor Busy	82	77	85	78	85	84
Percent Processor Ovhd	11	15	8	15	8	9
Percent Processor I/O	3	3	3	3	3	3
Percent Processor Idle	4	5	4	4	4	4
Number Msgs Sent	120	15	0	15	0	0
Total Bytes Sent	1053499	127350	0	143377	0	0
Number Msgs Rcvd	119	7	8	7	8	8
Total Bytes Rcvd	1053499	39686	81696	73274	58165	109703
Max Queue Size (count)	1	1	1	1	1	1
Max Queue Size (bytes)	16211	10811	14667	14537	15145	15860
Max Msg Sent (bytes)	16221	15905	0	15860	0	0
Max Msg Rcvd (bytes)	16221	10811	14667	14537	15145	15860

Figure 7-36. Statistical Summary Display

Processor Status

Figure 7-37 shows the Processor Status display.

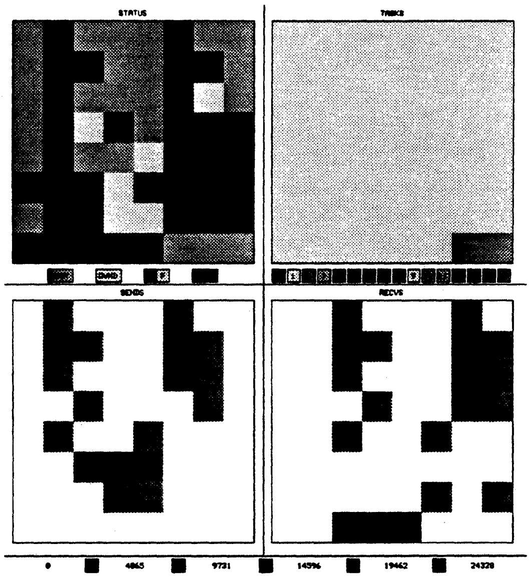


Figure 7-37. Processor Status Display

This is a comprehensive display that attempts to capture detailed information about processor utilization, communication, and tasks, but in a compact format that scales up well to large numbers of processors. This display contains four subdisplays, in each of which the processors are represented by a two-dimensional array of squares, with processor numbers filling the array in row-wise order.

The upper left subdisplay shows the current state of each processor (busy/overhead/I-O/flush/idle), using the usual green/yellow/brown/grey/red color scheme.

The upper right subdisplay shows the task currently being executed by each processor. The legend at the bottom of the subdisplay shows the colors used for the different task numbers. These are the same as the ones used in the task displays. Because of space limitations, only a limited number of tasks are shown in the legend. If the number of tasks exceeds this number, you can refer to the legend in the task displays to determine the color code.

The lower left subdisplay shows the volume in bytes of messages currently being sent by each processor, and the lower right subdisplay shows the volume in bytes of messages currently awaiting receipt by each processor; both of these communication subdisplays indicate message volume in bytes using the same color code as discussed previously for the other communication displays. The color coding is also shown at the bottom of the subdisplays.

Phase Portrait

Figure 7-38 shows the Phase Portrait display.

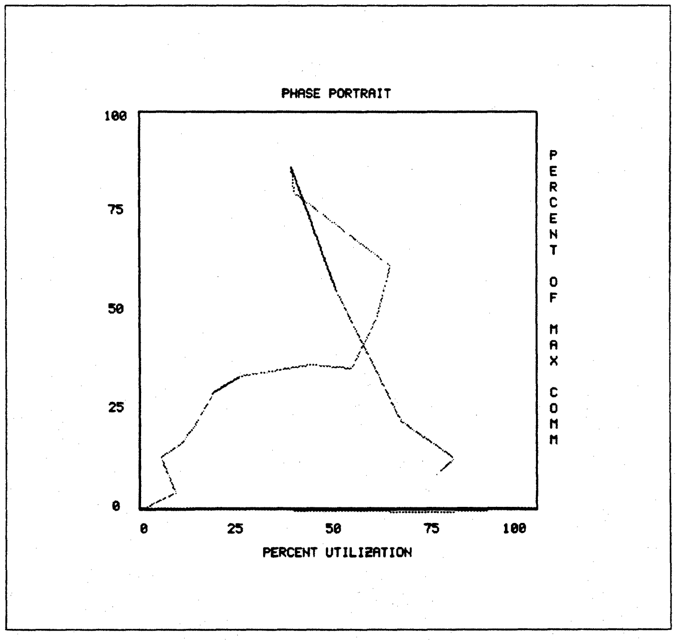


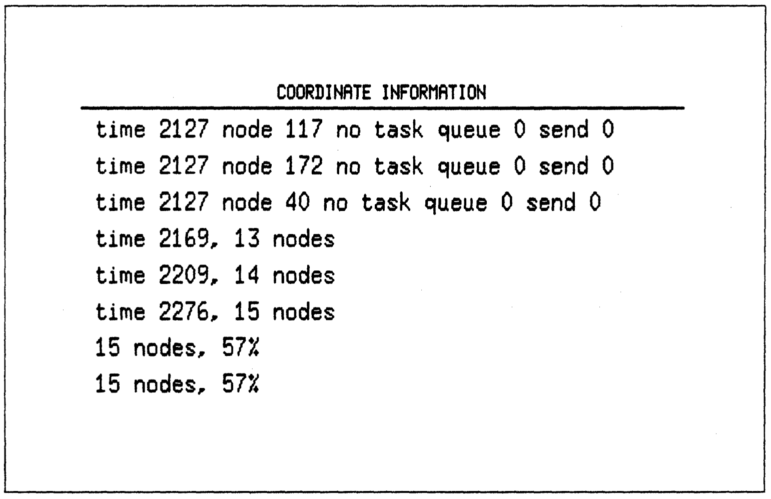
Figure 7-38. Phase Portrait Display

This display illustrates the relationship over time between communication and processor utilization. At any given point in time, the current percentage utilization (i.e., the percentage of processors that are in the busy state), and the percentage of the maximum volume of communication currently in transit, together define a single point in a two-dimensional plane. This point changes with time as communication and processor utilization vary, thereby tracing out a trajectory in the plane that is plotted graphically in this display, with communication and utilization on the two axes.

Since the overhead and potential idleness due to communication inhibit processor utilization, one expects communication and utilization generally to have an inverse relationship. Thus one expects the phase trajectory to tend to lie along a diagonal of the display. This display is particularly useful for revealing repetitive or periodic behavior in a parallel program, which tends to show up in the phase portrait as an orbit pattern. The color used for drawing the trajectory is determined by the current task number on processor 0 (default is black if no such task is active), so by setting task numbers appropriately, you can color code the trajectory to highlight either major phases or individual orbits.

Info

Figure 7-39 shows the Info display.



```
COORDINATE INFORMATION
time 2127 node 117 no task queue 0 send 0
time 2127 node 172 no task queue 0 send 0
time 2127 node 40 no task queue 0 send 0
time 2169, 13 nodes
time 2209, 14 nodes
time 2276, 15 nodes
15 nodes, 57%
15 nodes, 57%
```

Figure 7-39. Info Display

This is a non-graphical display used to write information produced by mouse clicks on some of the other displays. Many of the displays respond to mouse clicks by printing in the Info display the coordinates (in units meaningful to you) of the point at which the cursor is located at the time the button is pressed. This feature is intended to enable you to determine precisely information that may be difficult to read accurately from the axis scales alone. In addition, clicking a mouse button with the cursor placed on one of the nodes in the Animation display causes the following information to be printed in the Info display: simulation time, node number, current task number (if any), number of incoming messages pending, number of outgoing messages pending. The latter information can be used to determine the exact state of the nodes more precisely.

The following displays are supported by the Info display:

Utilization Count	prints the simulation time and the number of nodes that correspond to the x and y coordinates of the point you clicked on.
Utilization Gantt	prints the simulation time and the number of the selected node
Utilization Kiviat	prints the node number and the percentage of utilization
Utilization Summary	prints the node number and the percentage value.
Utilization Meter	prints the utilization percentage value.
Utilization Profile	prints number of nodes and the percentage value.
Communication Traffic	prints simulation time and message load (count or volume).
Spacetime	prints simulation time and number of the selected node.
Communication Queues	prints number of selected node and queue value (length or count).
Communication Matrix	prints source and destination processor.
Animation	prints simulation time, node number of the node you clicked on and the following information about that node: current task number (if any), number of incoming messages pending, number of outgoing messages pending.
Node Info	prints simulation time, node number, message type, and message length or distance depending on what subdisplay is currently selected.
Task Count	prints selected task number and number of nodes.
Task Gantt	prints simulation time and number of selected node.
Task Summary	prints task number and percentage value.
Processor Status	prints node number that corresponds to the square you clicked on and the following information about that node: current task (if any), volume of outgoing messages pending (in bytes) and volume of incoming messages pending.
Phase Portrait	prints utilization and communication percentage values.

Hints for Using ParaGraph

This section provides a few hints to help you use ParaGraph.

Interpretation of Trace Events

In order to be able to interpret the data produced by ParaGraph correctly, some details about how the different trace events are interpreted by ParaGraph should be kept in mind:

Message Passing Operations

Most message-passing operations are visualized as overhead time in the ParaGraph displays. For example, a probe() call leads to two trace records, where one denotes the time of entry into the operation and another the time of exit from the operation.

In addition to the message passing overhead records, there are records that mark the times at which send and receive calls are processed by the message passing library. These are used by the communication displays to mark the time of send and receive operations.

In addition, the Paragon programming model supports the notion of handler-driven communication. Thus, at any time a process may be interrupted by a message passing handler. In this case, the time spent within the message passing library is once again visualized as overhead time. The time spent within the handler is visualized as busy time.

The Paragon trace format contains no information about message operations that are issued by the user code but canceled afterwards (e.g. using the msgcancel() or flushmsg() calls or force types). If a message send operation is issued and no corresponding receive operation is found, this shows up in the Communication Traffic, Communication Queues and Communication Matrix displays as messages that are still in the message queues even though this may not be really the case because the message has been flushed from the buffers.

Trace Size

Perhaps the most important piece of advice is to keep the tracefile to be viewed as small as possible without losing the phenomenon to be studied. The best way to accomplish this is to use a relatively small number of processors and a brief execution time. Although ParaGraph currently supports the use of up to 512 processors, and has no limit on the duration of the simulation run, the size of the tracefile for a large number of processors and/or a long execution time can be enormous (many megabytes). Such large tracefiles can quickly consume large amounts of disk space and requires a great deal of time for ParaGraph to preprocess and then animate visually.

Fortunately basic algorithm behavior and most fundamental bottlenecks and inefficiencies in parallel programs are usually already apparent when viewed with small numbers of processors and relatively small test problems that run quickly. Moreover, many programs display repetitive behavior, so that only a few iterations need be examined in detail in order to get the gist of their behavior, rather than a long sequence of replicated behavior.

Performance Monitoring Buffer Size

The size of the performance monitoring buffers allocated by the performance monitoring library can have a dramatic effect on the perturbation introduced by performance monitoring. If the buffers are too small, they will fill very quickly and the application will have to flush the buffers repeatedly. The time spent flushing the buffers is essentially idle time for the application, even though it is visualized separately in the Utilization displays. The user should also note that if an application is synchronous in nature, flushing the buffers for one process will often cause other application processes to go idle.

By inspecting the amount of flush time in the utilization display, the user should be able to judge, whether a monitoring run needs to be repeated with larger performance monitoring buffers. However, the user should keep in mind that increasing the performance monitoring buffer size may also perturb the application because it may lead to increased paging activity.

The size of the performance monitoring buffers can be specified on the IPD command line (-bufsize option) or from the XIPD instrument dialog. As a rule of thumb, a buffer size that is equal to the total size of the trace produced by a given application divided by the number of processes can be used.

Parameters

The various parameters in the Configure dialog can have a dramatic effect on the behavior of ParaGraph for a given tracefile, and you may or may not find the default values to be the most desirable. For example, during preprocessing a rough heuristic is used to choose an appropriate time unit, and the value chosen strongly affects the appearance and behavior of the scrolling displays. An attempt is made to choose a value that fills at least one window width but not need to scroll more than a few window widths. The value chosen automatically may be so large that it obscures detail you would like to see, or so small that the simulation runs for too long. So, you should feel free to adjust the value for the time unit, if desired.

Note, however, that the scale width parameter also affects the visual resolution of the scrolling displays, so it may also be changed to produce a desired effect. In addition, the speed of the drawing is strongly affected by the type and amount of scrolling employed, so this is subject to experimentation as well. In using the Kiviat Diagram and Phase Portrait displays, some experimentation with the smoothing interval, as well as the time unit, may be required to produce the most meaningful visual results.

As pointed out previously, the execution speed of ParaGraph is normally determined by how fast it can read trace records and perform the resulting drawing. If the visual simulation is too rapid for the eye to follow, then its execution can be slowed down either by using the slow motion slider or else by selecting some additional displays, especially those that scroll with time. If the visual simulation is too slow, it can be speeded up by using fewer displays at a time or selecting jump scrolling. Changing the time unit and/or scale width also affects the drawing speed, so these are subject to experimentation as well. Finally, the step button or repeatedly hitting pause/resume can also be used to control the speed with which the animation unfolds. By some combination of these means, you should be able to produce an animation speed that can be followed visually in sufficient detail, yet does not take an inordinate amount time to finish.

For traces that have a large number of nodes it may be a good idea to focus on subsets of nodes at a time (using the Select Nodes command). This brings out more detail for the nodes that are visualized.

Restrictions

Scalability

The maximum number of nodes currently supported by ParaGraph is 512. Because of their limited scalability, some of the displays are restricted to even smaller numbers of nodes. The limitations are as follows:

- Spacetime display: 256 nodes
- Ring option in the Animation display: 256 nodes
- Network display: 512 physical nodes. Note that this limit may be reached even if the number of logical nodes that are traced is less than 512 (for example if logical nodes 0 and 1 correspond to physical nodes 0 and 550).
- Node Info display: 256 nodes
- Topology display: 16 nodes

These restrictions refer to the number of nodes that are visualized. It is thus possible to partly eliminate them by focusing on subsets of nodes through the Select Nodes command. This works for all the displays except for the Network display since this is the only display that shows physical rather than logical nodes.

Controlling Process

Paragon applications have a process that resides on a service node and controls the application. This controlling process is not currently visualized by ParaGraph because the method for tracing controlling processes is not yet clear. In principle, it would be no problem to include the controlling process in the visualization, even though the different scheduling characteristics of service nodes as opposed to compute nodes could create problems.

Window Placement and Window Managers

All the ParaGraph displays are implemented as transient windows. This makes it easy to remove all the displays from the screen (by simply iconifying the main ParaGraph window) and bring them back up in the same place (by de-iconifying). However, this means that under some window managers (for example, *mwm*) it is not possible to raise the main window above the ParaGraph display. It is therefore a good idea to have a fixed place for the main ParaGraph window (for example, by setting the Paragraph*geometry resource). Also, when using the *twm* window manager, you should make sure to customize the window manager in such a way that transient windows appear with a window title. This can be accomplished by specifying `DecorateTransients` in the *.twmrc* file.

Possible Problems

The following section gives some more details on problems that may occur when generating traces for ParaGraph and on how these problems can be avoided.

Generating Traces

As mentioned before, event tracing can generate massive amounts of data. The exact data rate depends on the characteristics of the application. Communication intensive applications generate more data than compute intensive applications but data rates of up to 0.5 MB per node per second are common. If massive amounts of data are generated and have to be written to disk, IPD may hang or appear to hang for long periods of time.

Thus, you should try to reduce the amount of data that needs to be captured using the selective instrumentation facilities provided by IPD's instrument command. Possible approaches include:

- Use the start-location and stop-location arguments to capture only an inner loop of the application.
- Use the context argument to restrict monitoring to a subset of nodes used by the application.

Please refer to the sections on Trace Size and Performance Monitoring Buffer Size for more information.

In general, writing traces to an NFS mounted file system is slower than writing to a local file system so generating large traces on an NFS mounted file system should be avoided.

Page Warmup

When tracing applications that have an iterative structure, it is often the case that the first iteration takes considerably longer than subsequent iterations because of paging effects. This can cause misleading performance traces to be generated. This can be avoided by not tracing the first iterations.This can be done by inserting a statement that is not executed for the first iterations and using the start-location argument to the instrument command to turn performance monitoring on when the statement is executed. Alternatively, the application can be stopped after a few iterations using an IPD breakpoint and performance monitoring can be turned on after the application is stopped.

Intrusion Caused by Flushing Buffers

When the performance monitoring buffers are full or when the write-location specified using the instrument command is reached, the event buffers are flushed over the message passing network and written to disk. If this occurs while performance monitoring is still going on for a part of the application, this can cause indirect intrusion. For example, if many nodes simultaneously start flushing their buffers while one node is still doing I/O, the I/O server may service the request issued by the node that is still being traced with considerable delay. In such a situation, you should make sure that the buffers are flushed after *all* nodes have reached the location that was specified as the end location for performance monitoring using the instrument command. One way to make sure this is the case is to add a global synchronization statement (gsync) just after the section of code that is being profiled and specify the function exit to this global synchronization as the write-location when issuing the instrument command.

Global Clock

When interpreting a trace, ParaGraph performs consistency checks to make sure the trace was generated correctly. If a problem is found, an error message is written to the Message Log. Please refer to the section *Message Log* on page 7-19 for a description of the error messages.

ParaGraph expects traces to be in ascending order by timestamps and message send events to be seen before corresponding receive events. Messages may appear to have been received before they are sent if only part of the application run is traced or if messages are flushed after being sent using flushmsg(), msgcancel() or force types. If you are not using this functionality and see one of the following error messages, the hardware monitoring support that implements the Paragon's global clock may be failing:

```
Tracefile not in ascending order by timestamps:
Message received before sent:
```

Please consult your system administrator to make sure this is not the case.

Busy waiting loops

You should be aware that every message passing operation causes at least two trace events to be generated (one for function entry, one for function exit). If a message passing operation is executed inside a busy waiting loop, this can cause considerable amounts of trace data to be generated. For example, this will be the case if an asynchronous probe operation is executed inside a loop to wait for completion of a message passing operation. Event tracing this kind of construct should be avoided.

Use of Colors

As mentioned before, ParaGraph supports both monochrome and color screens. Since ParaGraph makes extensive use of color, an 8-bit color display should be preferred when working with ParaGraph.

By default, ParaGraph tries to allocate its colors as read-only colorcells from the default colormap. The number of different colors used by ParaGraph is quite large (around 90). In some cases, the default colormap may not have enough colorcells available. In this case, ParaGraph switches to a private colormap. This has the effect of making all colors available but the colors may change when the mouse cursor leaves the ParaGraph displays. To avoid this behavior, the **-s** command line flag can be used to force ParaGraph to allocate read-only colorcells from the default colormap. In many cases, more colors can be made available in the default colormap by freeing the default colormap before ParaGraph is invoked (e.g. by calling **xstdcmap -delete default**).

Users should also keep in mind that additional colorcells are needed to invoke the color palette. Thus, the number of colors available in the color palette may change depending on the number of free colorcells, or the color palette may be disabled if no more colorcells are available.

Configuring ParaGraph

You can configure ParaGraph by using an X resource file. You can make the resource entries in your *.Xdefaults* file (which resides in your home directory) or in a file named *Paragraph* (located either in your home directory or in a directory specified by the *XAPPLRESDIR* environment variable). The entries in the *.Xdefaults* file take precedence over the *Paragraph* file entries.

Along with the resources corresponding to the standard X toolkit command line options, the following ParaGraph application resources are provided to configure ParaGraph:

Paragraph*ScrollValue	This resource determines whether ParaGraph displays scroll smoothly or jump scroll by a user-specified amount. The resource parameter is Smooth, Jump1/8, Jump1/4, Jump1/2 or Jump1.
Pargraph*ScaleWidth	This resource determines the number of pixels used to depict one time unit. The resource parameter is 1, 2, 4, 8 or 16.

Paragraph*BackingStore	This resource determines whether backing store is used in the ParaGraph displays. The resource parameter is on or off.
Paragraph*StepIncrement	This resource determines how many consecutive records from the tracefile are processed each time the step button is pressed. The resource parameter is a positive integer value.
Paragraph*helpfile	This resource determines the name of the help file used by ParaGraph's help utility.
Paragraph*font1	The font used to label the inside of the displays. This should be a fixed size font.
Paragraph*font2	The font used to label the Animation, Kiviat, Clock and Task Status displays. It should be a fixed size font that is bigger than font1.
Paragraph*font3	The font used to label the Animation and Topology displays for large node numbers. It should be a fixed size font that is smaller than font1.
Paragraph*MPsupport	The resource used to turn ParaGraph's multi-process support on and off.
Paragraph*fgColor	The foreground color of the ParaGraph displays.
Paragraph*bgColor	The background color of the ParaGraph displays.
Paragraph*busyColor	The color used to depict the busy state in the Utilization displays and the Animation and Topology displays.
Paragraph*ovhdColor	The color used to depict the overhead state in the Utilization displays and the Animation and Topology displays.
Paragraph*idleColor	The color used to depict the idle state in the Utilization displays and the Animation and Topology displays.
Paragraph*ioColor	The color used to depict the I/O state in the Utilization displays and the Animation and Topology displays.
Paragraph*flushColor	The color used to depict the flush state in the Utilization displays and the Animation and Topology displays.

Paragraph*sendColor	The color used to depict the sending state in the Animation and Topology displays.
Paragraph*recvColor	The color used to depict the receiving state in the Animation and Topology displays.
Paragraph*trafColor	The color used to draw the queue values in the CommunicationTraffic display.

Paragraph*lgnd1Color through **Paragraph*lgnd5Color**

The colors used to color code message lengths in the color code legend display.

Paragraph*msgqColor	The color used to draw message queue values in the Communication Queuesdisplay.
Paragraph*msghColor	The color used to draw message queue values highwater mark in the Communication Queues display.

Paragraph*nwk1Color through **Paragraph*nwk7Color**

The colors used to color code the number of messages that travel across links in the Communication Network display.

Paragraph*clockColor	The color used to draw the time bar in the clock display.
Paragraph*bkbgColor	The color used to draw started tasks in the Task Status display.
Paragraph*bkedColor	The color used to draw finished tasks in the Task Status display.
Paragraph*kivtColor	The color used to draw utilization values in the Utilization Kiviat display.
Paragraph*kivhColor	The color used to draw the utilization highwater mark in the Utilization Kiviat display.

Paragraph*task1Color through **Paragraph*task63Color**

The colors used to color code task numbers, message types and message distances.

Default Configuration

The following resource definitions are the ParaGraph default configuration.

Paragraph*font1	6x12
Paragraph*font2	8x13
Paragraph*font3	5x8
Paragraph.BackingStore	on
Paragraph.ScrollValue	Jump1/8
Paragraph.StepIncrement	1
Paragraph.ScaleWidth	1
Paragraph*helpfile	ParaGraph.hlp
Paragraph*fgColor	black
Paragraph*bgColor	white
Paragraph*busyColor	SpringGreen2
Paragraph*ovhdColor	brown4
Paragraph*flushColor	light grey
Paragraph*idleColor	red2
Paragraph*ioColor	orange
Paragraph*sendColor	blue
Paragraph*recvColor	yellow
Paragraph*trafColor	blue
Paragraph*lgnd1Color	dodger blue
Paragraph*lgnd2Color	medium purple
Paragraph*lgnd3Color	magenta3
Paragraph*lgnd4Color	maroon1

Paragraph*lgnd5Color	red
Paragraph*msgqColor	medium purple
Paragraph*msghColor	plum2
Paragraph*nwk1Color	RoyalBlue1
Paragraph*nwk2Color	cyan2
Paragraph*nwk3Color	medium spring green
Paragraph*nwk4Color	SpringGreen2
Paragraph*nwk5Color	gold
Paragraph*nwk6Color	dark orange
Paragraph*nwk7Color	red2
Paragraph*clockColor	cyan2
Paragraph*bkbgColor	bisque2
Paragraph*bkedColor	LightSalmon4
Paragraph*kivtColor	orchid
Paragraph*kivhColor	plum2

The Parallel Make Utility 8

The parallel **make** utility, **pmake**, for the Paragon system brings the advantages of parallel processing to a traditionally time-consuming part of program development - building and updating programs that consist of multiple source files. **pmake** is based on GNU **make**. In addition to the features of GNU **make**, **pmake** gives you control over parallel execution in the compute partition of the Paragon system and provides other features designed to improve compatibility with other **make** utilities.

NOTE

pmake is an extension of GNU **make** and is distributed under the terms of the GNU General Public License. As such, Intel will provide a complete, machine-readable copy of the **pmake** source code upon request. For more information, contact Intel's Customer Service Response Center, as described in the section *Comments and Assistance* in the Preface of this book.

Most computer applications consist of numerous source modules, each of which may refer to one or more include files. Whenever any of these files is changed during the development process, the following must occur:

- Each changed file must be recompiled.
- All files that depend upon the changed file must be updated.
- All of the files must be relinked to update the application.

The purpose of a **make** utility is to make this process as automatic and efficient as possible. Generally, **make** is used to recompile large programs, but you can use it for any task in which files must be updated automatically whenever the files they depend upon change.

This chapter provides an overview of **pmake** and the makefile description file and describes differences between **pmake** and GNU **make**. For detailed information on GNU **make**, refer to the *GNU Make* manual. To receive a copy of the *GNU Make* manual, contact the Customer Service Response Center as described in the section *Comments and Assistance* in the Preface of this book.

Invoking pmake

To invoke **pmake**, use the **pmake** command as follows:

pmake *[options]* [*macro_definition* ...] [*target* ...]

options can be one or more of the following **pmake** command line options:

-b	Has no effect; exists so older-version **make** dependency files continue to work.
-c	Does not try to find a corresponding Revision Control System (RCS) or Source Code Control System (SCCS) file and check it out if the file does not exist.
-C *dir*	Changes to directory *dir* before reading the description files or doing anything else. If multiple **-C** options are specified, each is interpreted relative to the previous one: **-C/ -C**`etc` is equivalent to **-C** `/etc`. This is typically used with recursive invocations of **pmake**.
-d	Prints debugging information in addition to normal processing. The debugging information includes information about the files considered for processing, the comparison of file-times, the files that need processing, and the implicit rules being considered and actually applied.
-e	Does not reassign environment variables within the description file.
-f *file*	Reads *file* for a description of how to build the target file. If you do not specify the **-f** option, **pmake** looks in the current directory for a description file named *makefile* or *Makefile*. If a – (dash) follows the **-f** option, **pmake** reads standard input. You can specify more than one description file by entering the **-f** option more than once (with its associated *file* argument).
-F	Causes a fatal error if a description file is not present.
-i	Ignores error codes returned by commands and continues to execute until finished. This is similar to the pseudotarget command **.IGNORE:**, which can be specified in the description file. The **pmake** command normally stops if a command returns a nonzero code.

-I *dir*	Specifies a directory *dir* to search for description files to be included. You can specify the **-I** option multiple times in a command line to specify multiple directories to search. The directories are searched in the order specified.
-j [*jobs*]	Specifies the maximum number of jobs that can run simultaneously. The default is the partition size, or 1 if the **pmake** command is running in the service partition. If there is more than one **-j** option, the last one is effective. If the **-j** option is given without an argument, the **pmake** command does not limit the number of jobs that can run simultaneously.
-k	Stops processing the current target if an error occurs, but continues with other branches that do not depend on the target that failed.
-l [*load*]	Specifies that no new jobs should be started if there are other jobs running and the load average is at least the value of *load* (a floating-point number). Specifying the option with no argument removes a previous load limit.
-m	Searches for machine-specific subdirectories automatically. On a Paragon system, if a *PARAGON* subdirectory exists in the current directory, the **-m** option adds the *PARAGON* subdirectory to the directory list specified by the *VPATH* special variable. See the "Special Variables" section for more information.
-n	Echoes commands that would be executed, but does not execute them.
-N	Disables all configuration file (*Makeconf*) processing.
-o *file*	Does not process *file* even if it is older than its dependencies, and does not process anything because of changes in *file*. Essentially, the file is treated as very old and its rules are ignored.
-p	Echoes all the environment variables, macro definitions, and target descriptions before executing any commands. This also prints the version information given by the **-v** option. To print the database without trying to remake any files, use the following: `pmake -p -f /dev/null`
-P [*partition*]	Runs the **pmake** command as a parallel application in the partition specified by the *partition* argument. The name you specify for the *partition* argument must be *.compute* or the name of a subpartition in the *.compute* partition. If your specify the **-P** option without an argument, the default partition is the value of the *NX_DFLT_PART* environment variable, or *.compute* if *NX_DFLT_PART* is not set. If you do not specify the **-P** option, the default partition is the *.service* partition.

-q	Does not execute the commands in the description file. Returns a status code of zero if the object files are up-to-date; otherwise, returns a nonzero value.
-r	Eliminates the built-in implicit rules and clears out the default list of suffixes for suffix rules.
-s	Does not echo the commands being executed. This is similar to the pseudotarget command **.SILENT:**, which would be specified in the description file.
-S	Stops processing the current target if an error occurs and does not continue to any other branch. This is the default. This cancels the effect of the **-k** option. This is not necessary except in a recursive **pmake** where **-k** might be inherited from the top-level **pmake** via *MAKEFLAGS* or if you set **-k** in *MAKEFLAGS* in your environment.
-t	Touches the targets; marks the files up-to-date without running commands to update them, or creates the target if it does not exist.
-u	Does not unlink files that were automatically checked out from SCCS or RCS.
-U	Has no effect; exists so older-version **make** dependency files continue to work.
-v	Prints the version of the **pmake** command, a copyright, a list of authors, and a notice that there is no warranty. After this information is printed, processing continues normally. To get this information without doing anything else, use the following: `pmake -v -f /dev/null`
-w	Prints a message containing the working directory before and after other processing in the directory. This may be useful for tracking down errors from complicated nests of recursive **pmake** commands.
-W *file*	Pretends that the target *file* has just been modified. When used with the **-n** option, this shows you what would happen if you were to modify that file. Without **-n,** it is almost the same as running a *touch* command on the given file before running the **pmake** command, except that the modification time is changed only in the imagination of the **pmake** command.

The other parameters to the **pmake** command are defined as follows:

macro_definition	Specifies a macro to use with the definition file. Use the same macro syntax as required for the definition file. Enclose strings in quotes. Spaces and tabs are ignored. See the "Macros" section for more information on using macros.
target	Name of the target to build or update. Target names are typically executable files, but this is not always the case. If *target* is not specified, **pmake** uses the first target in the definition file.

pmake Extensions to GNU make

While **pmake** is based on GNU **make**, **pmake** offers some additional features. These include additional parallel execution control, extensions to macro definition, configuration file support, and other additional features.

Parallel Controls

pmake is designed to update multiple target files in parallel. Parallel execution can occur in either the service partition or the compute partition. **pmake** provides the following options for parallel control:

-P	Specifies the partition in which **pmake** runs jobs. The default is the service partition.
-j	Specifies the maximum number of jobs that can run in parallel.
-l	Restricts pmake from executing commands in parallel when the system load average reaches a limit.

Using a Compute Partition

When you use the **-P** option to specify one of the compute partitions on the Paragon system, **pmake** calls **nx_initve**() to become a gang-scheduled parallel application. Then **pmake**, running in the service partition, acts as the controlling process, sending commands out in parallel to nodes in the compute partition as the nodes become available.

The **-j** option allows you to specify the maximum number of jobs that can run in parallel. If you do not use the **-j** option, the maximum number of jobs defaults to the number of nodes in the partition, or one node if **pmake** is running in the service partition. If you use the **-j** option followed by the optional *jobs* argument, **pmake** runs up to the number of jobs specified in parallel.

The number of jobs **pmake** can run in parallel is not limited to the number of nodes in the partition, because multiple jobs can run on a node. If you use the **-j** option without the *jobs* argument, the maximum number of jobs **pmake** can run in parallel is unlimited.

The following examples illustrate the use of the **-j** and **-P** options with the **pmake** command. The first example runs *N* jobs in parallel in the compute partition, where *N* is the number of nodes in the partition.

```
pmake -P.compute
```

The next example runs up to ten jobs in parallel in the compute partition. If there are more than ten nodes in the compute partition, only the first ten are used. If there are less than ten nodes, some nodes run multiple commands at once.

```
pmake -P.compute -j10
```

The next example runs as many jobs as possible in the compute partition. If there are twenty commands that can be run in parallel and only five nodes in the compute partition, each node runs four commands.

```
pmake -P.compute -j
```

Specifying the **-P** option causes **pmake** to become a gang-scheduled parallel application. Therefore, any use of the **-P** option in subsequent recursive invocations of **pmake** is ignored, because it is already gang-scheduled. Therefore, you need to use the **-P** option at a level where it can do the most good.

For example, if you make the files in two directories, one with many files and another with a few files, you would do better to invoke parallelism in updating the large directory, rather than at the upper level, where the parallelism would be wasted. Suppose you entered the following **pmake** command:

```
pmake -j2 -P.compute
```

Used on the following *makefile*, the previous command would update the two targets *big* and *little* simultaneously.

```
all: big little

big:
        cd bigdir; $(MAKE)

little:
        cd littledir; $(MAKE)
```

The target *little* would be updated quickly, while *big*, which involves many compiles, might take several hours to build, and the benefits of parallelism would be lost. In this case, it would be better to invoke the top-level **pmake** without the **-j** and **-P** options and to use the options at the second level as in the following:

```
all: big little

big:
        cd bigdir; $(MAKE) -j8 -P.compute
little:
        cd littledir; $(MAKE) -j2 -P.compute
```

pmake relies on the dependencies defined in the description file to determine the files that can be updated in parallel. These dependency definitions prevent two files, one of which is dependent upon another, from being updated simultaneously. All commands that update a single file are assumed to be sequential, and are run in the order in which they appear in the description file.

For example, if *file2* is dependent upon *file1*, all commands that update *file1* are run sequentially before commands updating *file2* are executed. If there is no dependency between *file2* and *file1*, commands updating *file2* may be run in parallel with commands updating *file1*. It is, therefore, quite important to ensure that your description file clearly defines all dependencies.

Using the Service Partition

If you do not use the **-P** option, **pmake** runs and executes all *makefile* commands in the service partition. In the service partition, **pmake** uses the **fork()** call to start commands simultaneously, and relies on process migration and load balancing to ensure parallel execution. Using the **-j** option allows you to specify the number of jobs that **pmake** can run in parallel. If you do not use the **-j** option, **pmake** runs as a single process on one node of the service partition.

Controlling System Loading

Another parallel option, **-l**, allows you to restrict **pmake** from executing multiple commands in parallel when the system load average gets too high. The *load* argument to the **-l** option allows you to specify a load average beyond which **pmake** limits jobs. In this way, you can ensure that the system is not excessively slowed down. **pmake** always allows one command to execute, even if the load average is over the specified limit. However, if the load average is over the limit and one or more commands are already executing, **pmake** will not start any more commands. Specifying the **-l** option with no *load* value removes all previous load limits. The **-l** option is most useful when running in the service partition, where there is more likely to be contention for system time.

Macro Extensions

pmake provides the special macro **$$@** and three macro references (pattern replacement references using regular expressions, C-shell-style modifiers, and conditional expressions) in addition to the macro capabilities of GNU **make**.

Special Macro

The macro **$$@** provides compatibility with System V **make**. This macro is used on the dependency line of a rule and is interpreted as the current target (the one currently being processed).

Pattern Replacement

You can use a pattern replacement reference that causes a replacement string to be substituted for a regular expression within the macro expansion. This has the form:

$(*MACRO*/*reg-expression*/*replacement*)

where *reg-expression* is a regular expression, (as described in the online manual page **regexp**()), and *replacement* is the replacement string. The syntax also allows you to use semicolons in place of slashes.

Modifiers

You can use modifiers similar to the C-shell file name modifiers in variable expressions with the form:

$(*MACRO*:*X*)

In this reference, *X* may be **t** (tail), **h** (head), **r** (root) or **e** (extension).

Conditional Expressions

You can define conditional variables using C-style colon expressions as follows:

$(*MACRO*?*value1*:*value2*)

An expression of this form evaluates to *value1* if *MACRO* is defined, and *value2* if it is not.

Configuration File Support

pmake supports the use of a *configuration file*. This file, if it exists, must be named *Makeconf*. **pmake** searches backwards for this file from the current directory to the root directory. Only the first *Makeconf* file found in the path is evaluated, so an empty *Makeconf* file in a searched directory terminates the search. Although **pmake** searches for a *Makeconf* file by default, and does not return an error if no *Makeconf* is found, you can explicitly disable all *Makeconf* processing by using the **-N** option.

On finding a *Makeconf* file, **pmake** evaluates it as though its contents were at the top of the *makefile*. Although you can define any global variables or other global information in a *Makeconf* file, the file is most useful when your software project is organized into completely separate source and object directory trees.

To support separate source and object directory trees, the *Makeconf* file can define the variable *OBJECTDIR* to be the root of the object directory tree. This can be defined either as an absolute path, or a path relative to the location of *Makeconf*.

There is special processing for the *OBJECTDIR* definition. Before running any commands, **pmake** goes to the object directory and modifies its search path to include the path back to the corresponding source directory. For example, suppose a directory named *$SRC* is your root source directory, and that this directory has a *Makeconf* file containing the following absolute path *OBJECTDIR* definition:

```
OBJECTDIR=/topdir/objdir
```

In this case, invoking **pmake** from within the directory *$SRC/subdir* causes **pmake** to create the directory */topdir/objdir/subdir* (if it does not exist), and to use that as the object directory. This ensures that the object directory structure is the same as the source directory structure.

You can also specify the *OBJECTDIR* definition as a pathname relative to the directory containing the *Makeconf* directory, and the directory structure will be created correctly.

After determining and setting the path for the object directory, **pmake** executes *makefile* commands, reading from the source directory, and creating or modifying files in the object directory.

By default, **pmake** looks for source files in the directory containing the *makefile*. You may, however, specify a different source directory by defining the SOURCEDIR variable in the *Makeconf* file. This definition has the following form:

```
SOURCEDIR=path1[:path2]...[:pathn]
```

As with the *OBJECTDIR* definition, you can define the *SOURCEDIR* paths as absolute or relative paths. As the syntax shows, you can also specify additional source directory trees to be searched. You can assign these alternate source root paths as a colon-separated list to the *SOURCEDIR* variable in the *Makeconf* file. This assignment permits you to work with source files stored in various trees. You can also specify these paths as absolute paths or as paths relative to the location of the *Makeconf* file.

Other Differences Between pmake and GNU make

There are other minor differences between **pmake** and GNU **make**. These are mainly enhancements to improve compatibility with other **make** programs.

Command Line options

pmake supports the following command line options not supported by GNU **make**:

-c	Causes **pmake** to not try to find and check out a corresponding SCCS or RCS file when a file does not exist.
-m	Causes **pmake** to search machine-specific subdirectories automatically.
-N	Disables all *Makeconf* processing.
-P	Specifies that **pmake** commands run in a compute partition.
-u	Causes **pmake** to not unlink files automatically checked out from SCCS or RCS. This option can be useful when an error occurs where an intermediate source file must be made to make an object file. Use the **-u** option to see the contents of the intermediate source file, rather than allowing **pmake** to remove it.

Special Targets

pmake provides two special targets for specifying entry and exit code: **.INIT** and **.EXIT**. If you define **.INIT** in your description file, this target and its dependencies are built before any other targets are processed. Defining **.EXIT** causes this target and its dependencies to be processed after all other targets are built.

Early versions of GNU **make** automatically exported all variables (macros) from the *makefile* to the environment. In contrast, the **pmake** utility does not, but does allow you to export variables explicitly by using the special target **.EXPORT**. Variables listed as dependencies of this target are expanded and exported to the environment in which **pmake** runs its commands.

Include Statement

In addition to the standard GNU **make include** statement, **pmake** adds another **include** statement with the following form:

-include *filename*

The standard form of the **include** statement includes and processes the named file, and returns an error if the file is not found. The form of the include with the added dash prefix, includes and processes the named file as does the other version, but does not return an error if the file is not found.

The makefile Description File

To use **pmake**, you need a *makefile*, also called a *description file*. A description file can contain four types of statements:

Rules	Rules define when and how to update files (called *targets* of the rules). Rules usually have a single target. A rule lists any files that the targets depend upon, called *dependencies* of the target, and *commands* to create or update the targets.
Variable definitions	A *variable definition* in a makefile assigns a text string to a variable, allowing you to use that variable name in place of the complete text string. For example, you could define a variable to be a list of all of the object files.
Directives	Directives are special commands. Available directives include directives that read another makefile and conditional directives that determine whether parts of the makefile are read based on the value of variables.
Comments	A "#" in a line starts a comment. The # character and subsequent characters in the line are ignored. You can continue a comment across multiple lines if the last character in a comment line is a backslash (\). A comment cannot be placed within a **define** directive or within some commands where the shell determines what a comment is. Comments can be in all other makefile lines.

The following sections describe some aspects of **pmake** description files. For complete information on description file statements and how to construct and use a description file, refer to the *GNU Make* manual.

When a description file exists for a program, invoking **pmake** executes all the commands needed to build the program. **pmake** uses the rules in the description file and the last-modification time of the target and the files that a target depends on to decide which targets need updating.

Description files contain a sequence of entries that define target names and dependencies and describe the rules for updating the targets. A typical entry includes a dependency line and a series of commands, and takes the following form:

target1 [*target2* ...] :[:][*dependency* ...] [; *command*]
[*command*] [; *command* ...]

The dependency line begins with one or more target names separated by spaces. A single or double colon separates the target(s) from a list of zero or more dependencies for the target(s). If no dependencies are given, the target files are always updated if they do not exist. Otherwise, a target is updated only if a dependency has changed since the target was last updated. A single command can also appear on the dependency line, separated from the dependencies by a semicolon. Alternatively, commands can appear on subsequent lines, provided each command line begins with a tab character. When a target requires updating, **pmake** executes the specified commands.

Dependency Lines

Dependency lines can take the following forms:

target... : [*dependency*] ...

Single-colon rules. Words following the colon are added to the dependency list for the target(s). If a target is named in more than one single-colon rule, the dependencies for all of its entries are concatenated to form that target's complete dependency list. In that case, only one of the single colon rules may include commands for remaking the target.

target... :: [*dependency*] ...

Double-colon rules. When used in place of a single colon (:), the double colon (::) allows a target to be checked and updated with respect to alternate dependency lists. Each double colon rule is considered independently when deciding whether and how to update a particular target.

target... : *target-pattern* : *dep-pattern...*

Static-pattern rules. The *target-pattern* and *dep-pattern* values specify how to compute the dependencies for each target. Each target is matched against the *target-pattern* to extract a part of the target name, called the stem. The stem is then substituted into each of the *dep-pattern* values to make the dependency names, for example, the following dependency line specifies that *foo.c* and *foo.h* are dependencies of *foo.o*:

```
$(OBJECTS): %.o : %.c %.h
```

.s1.s2 :	Double-suffix rules. The rule tells how to make a file *foo.s2* from the file *foo.s1* where *foo* is an arbitrary stem and *s1* and *s2* are suffixes.
.s1 :	Single-suffix rules. The rule tells how to make a file *foo* from the file *foo.s1* where *foo* is an arbitrary stem and *s1* is a suffix.

target-pattern : *dependency-pattern*...

Pattern rules. The target and dependency patterns each contain the wild card character, % (percent). The % in the *target-pattern* is matched against a stem of a target name. That stem is substituted for the % in the *dependency-pattern*. This creates the dependency for the target, for example, %.o : %.c with the stem *foo* creates the target *foo.o* from the dependency *foo.c*.

Commands

You can preface the description file commands to remake a target with one or more of the following special characters. The special characters are not passed to the shell but have the following effect within **pmake**:

–	**pmake** ignores any non-zero error code returned by the command.
+	**pmake** executes the command even if the **-n**, **-q** or **-t** options are specified.
@	**pmake** does not print the command before executing it.

Included Description Files

You can include description files within other description files by using the **include** directive. When **pmake** encounters an **include** directive within a description file, it temporarily stops processing the first description file, processes the included description file, then resumes processing the original description file.

include *filename*	Include and process *filename*. An error occurs if the file is not found
-include *filename*	Include and process *filename*. No error occurs if the file is not found.

Macros

You can use macros to simplify and improve the portability of description files. You can define macros on the command line or within the description file. Macro definitions can have the following general forms:

MACRO = value	Recursively expanding macro definition. The macro value is installed verbatim; if it contains references to other macros, those references are expanded when the macro is evaluated.
MACRO := value	Simply expanding macro definition. The value is evaluated once, when the macro is installed; imbedded macro references are evaluated at that time.
override *MACRO = value*	Override directive. Causes macro definition within a description file to override definition from the command line.

Macro references take the form $(*macro-name*) or ${*macro-name*} and can appear anywhere within the description file. pmake supports several special forms.

$(...$(*MACRO*)...)	Nested macro references.
$(*MACRO:suffix1=suffix2*)	Suffix replacement references. The value of *suffix1* is replaced by *suffix2* in the expansion of *MACRO*. The *suffix1* must occur at the end of a word.
$(*MACRO:pattern1=pattern2*)	Pattern replacement references.The *pattern1* and *pattern2* values each contain the wild card character, %. Occurrences of *pattern1* in the expansion of *MACRO* are replaced by *pattern2* with the % character matching any stem.
$(*MACRO/reg-expression/replacement*)	Pattern replacement references. The replacement string is substituted for the reg-expression within the macro expansion. The valid forms of regular expressions are described in **regexp(3)**. Semicolons may be used in place of the slashes that separate the *MACRO*, *reg-expression*, and *replacement* strings.
$(*MACRO:X*)	C-shell style modifiers. *X* may be **t** (tail), **h** (head), **r** (root) or **e** (extension).
$(*MACRO?value1:value2*)	Conditional expressions. Evaluates to *value1* if *MACRO* is defined and *value2* otherwise.

Special Macros

The following internal macros are automatically set as each target is processed:

$@	The name of the current target.
$*	The base name of the current target.
$<	The name of the current dependency file.
$?	The list of dependencies that are newer than the target.
$%	The name of the library member being processed.
$^	The list of all dependencies.
$$@	The current target (valid only on the dependency line).

Special Variables

Some variables have special meaning for the **pmake** command. Some of these variables are set automatically by the **pmake** command or they can be set as environment variables. The following special variables are supported:

cputype	The CPU type of the target system in lower-case (for example, `i860`). This variable is set automatically by the **pmake** command.
CPUTYPE	The CPU type of the target system in upper-case (for example, `I860`). This variable is set automatically by the **pmake** command.
MAKE	The command line with which **pmake** was invoked, excluding options. This variable is set automatically by the **pmake** command.
MAKEFILES	A list of description files to be read before any others. This variable may be set in the environment.
MAKEFLAGS	A list of the options specified on the command line. This variable is set automatically by the **pmake** command.
MAKELEVEL	The current level of make recursion. This variable is set automatically by the **pmake** command.
OBJECTDIR	The root of the object tree where **pmake** will build its targets.
SHELL	The shell to use for command execution. The default is */bin/sh*. This variable may be set in the environment or in a description file.

SOURCEDIR	The roots of the source tree where **pmake** will search for sources.
SUFFIXES	The list of default, known suffixes from built-in suffix rules. This variable is set automatically by the **pmake** command.
target_machine	The machine architecture of the target system in lower-case (for example, `paragon`). This variable is set automatically by the **pmake** command.
TARGET_MACHINE	The machine architecture of the target system in upper-case (for example, `PARAGON`). This variable is set automatically by the **pmake** command.
VPATH	A colon-separated list of directories to search for dependency files. This variable may be set in the environment or in a description file.

Pseudotarget Names

The **pmake** command assigns special meanings to the following pseudotargets:

.DEFAULT	If it appears in the description file, the rule for this target is used to process a target when there is no other entry for it.
.EXIT	If defined in the description file, **pmake** processes this target and its dependencies after all other targets are built.
.EXPORT	Variables listed as dependencies of this target are expanded and exported to the environment in which **pmake** runs it's commands. **pmake** does not normally export variables defined within a definition file.
.IGNORE	Ignore errors. When this target appears in the description file, **pmake** ignores non-zero error codes returned from commands.
.INIT	If defined in the description file, this target and its dependencies are built before any other targets are processed.
.PHONY	The dependencies of this target are considered to be "phony" targets. When it is time to consider such a target, **pmake** will run its commands unconditionally regardless of whether a file with that name exists or what its last modification time is.
.PRECIOUS	List of files not to delete. **pmake** does not remove any of the files listed as dependencies for this target when interrupted. **pmake** normally removes the current target when it receives an interrupt.
.SILENT	Run silently. When this target appears in the description file, **pmake** does not echo commands before executing them.
.SUFFIXES	The dependencies of this target are the suffixes that **pmake** will search for when applying suffix rules.

Conditional Execution

pmake provides conditional execution directives that control what parts of the description file **pmake** sees. The general format of a conditional directive is the following:

conditional-directive
text-if-true
endif

Another format is the following:

conditional-directive
text-if-true
else
text-if-false
endif

There are four different conditional directives. They are:

ifeq (*arg1*, *arg2*)	The conditional evaluates to true if *arg1* is equal to *arg2*.
ifneq (*arg1*, *arg2*)	The conditional evaluates to true if *arg1* is not equal to *arg2*.
ifdef *variable-name*	The conditional evaluates to true if *variable-name* is defined.
ifndef *variable-name*	The conditional evaluates to true if *variable-name* is not defined.

For complete information on how to construct and use a description file, refer to the *GNU Make* manual.

Index