

April 1996

Order Number: 312545-005

**Paragon™ System
Application Tools User's Guide**

Intel® Corporation

Copyright ©1996 by Intel Server Systems Product Development, Beaverton, Oregon. All rights reserved. No part of this work may be reproduced or copied in any form or by any means...graphic, electronic, or mechanical including photocopying, taping, or information storage and retrieval systems...without the express written consent of Intel Corporation. The information in this document is subject to change without notice.

Intel Corporation makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Intel Corporation assumes no responsibility for any errors that may appear in this document. Intel Corporation makes no commitment to update or to keep current the information contained in this document.

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

Intel software products are copyrighted by and shall remain the property of Intel Corporation. Use, duplication, or disclosure is subject to restrictions stated in Intel's software license agreement. Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraphs (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at 252.227-7013. Intel Corporation, 2200 Mission College Boulevard, Santa Clara, CA 95052-8119. For all Federal use or contracts other than DoD, Restricted Rights under FAR 52.227-14, ALT. III shall apply.

The following are trademarks of Intel Corporation and its affiliates and may be used only to identify Intel products:

286	i386	Intel	iPSC
287	i387	Intel386	Paragon
i	i486	Intel387	
	i487	Intel486	
	i860	Intel487	

Other brands and names are the property of their respective owners.

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.

LIMITED RIGHTS

The information contained in this document is copyrighted by and shall remain the property of Intel Corporation. Use, duplication or disclosure by the U.S. Government is subject to Limited Rights as set forth in subparagraphs (a)(15) of the Rights in Technical Data and Computer Software clause at 252.227-7013. Intel Corporation, 2200 Mission College Boulevard, Santa Clara, CA 95052. For all Federal use or contracts other than DoD Limited Rights under FAR 52.2272-14, ALT. III shall apply. Unpublished—rights reserved under the copyright laws of the United States.



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 system graphical tools follow the Motif style guide. This manual assumes the Motif window manager functionality. The Paragon system 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 system 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 system 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 Scalable 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

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

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

United Kingdom Intel Corporation (UK) Ltd.
Scalable Systems Division
Pipers Way
Swindon SN3 IRJ
England
0800 212665 (toll free)
(44) 793 491056
(44) 793 431062
(44) 793 480874
(44) 793 495108

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

World Headquarters
Intel Corporation
Scalable Systems Division
15201 N.W. Greenbrier Parkway
Beaverton, Oregon 97006
U.S.A.
(503) 677-7600 (Monday through Friday, 8 AM to 5 PM Pacific Time)
Fax: (503) 677-9147

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
(Internet)



Table of Contents

Chapter 1 ParAide

Using ParAide	1-1
Invoking ParAide	1-1
Launching Graphical Tools	1-3
Loading Applications	1-3
Managing Files	1-3
Windows, Menus, and Commands	1-4
File Menu	1-5
Open History Dialog	1-6
Mesh Menu	1-7
Load Application Dialog	1-7
Application Information	1-9
Application Options	1-9
Partition Selection	1-11
Load Options	1-13
Load Status Messages	1-16
Control Buttons	1-16
Load History Dialog	1-17

Tool Menu	1-18
XIPD Dialog	1-20
Go to ParaGraph Dialog	1-22
XProf and XGprof Dialogs	1-24
Command Menu	1-25
Customizing the Command Menu	1-25
Viewing Command Output	1-26
Help Menu	1-28
Help Index	1-29
Help Topic	1-30
Icon Strip	1-32
Terminal Area	1-33
Status Line	1-34
Selecting Files and Directories	1-34
Configuring ParaAide	1-36
Default Configuration	1-39
Environment Variables	1-40
Chapter 2	
Interactive Parallel Debugger	
Compiling for Debugging	2-2
Running IPD	2-2
IPD Commands	2-3
Syntax of IPD Commands	2-5
Using IPD	2-7
Context, Execution Point, and Scope	2-7
Loading a Program for Debugging	2-9

Controlling the Debug Environment	2-10
Defining Aliases	2-10
Recording Debugging Sessions	2-11
Command Files	2-11
On-line Help	2-12
Controlling Program Execution	2-12
Running a Program	2-12
Breakpoints, Tracepoints, and Watchpoints	2-13
Defining the Signal Set	2-13
Performance Monitoring	2-14
Examining and Modifying Programs	2-14
Source Code Listing	2-15
Program Disassembly	2-16
Message Queue Display	2-16
Variable, Address, and Register Display	2-17
Assigning Values	2-18
Debugging Threaded Applications	2-19
Debugging MPI Applications	2-20
Examining Core Files	2-23
Debugging Hints	2-24
Multi-Line Calls and Statements	2-24
Referencing Unnamed Fortran Main Programs	2-24
Displaying Fortran Variable Types	2-25
Using Keyboard Interrupts	2-26
Using IPD in a Sample Session	2-26
Creating the Example Program	2-27
Editing the .ipdrc File	2-27
Starting IPD	2-27
Loading the Program	2-28
Getting Help	2-28
Setting Up the Debugging Environment	2-30
Running the Program	2-32

Tracking the Fault	2-33
Exiting IPD	2-40

Chapter 3

XIPD

Using XIPD	3-1
Starting XIPD	3-3
Session History File	3-5
Creating a New Session	3-6
Start-up	3-6
Session Name	3-7
Loading a Program	3-9
Establishing a Viewpoint	3-10
Working with a Program's Source Code	3-11
Executing the Program and Examining Messages	3-12
On-Line Help	3-13
Windows, Menus, and Commands	3-14
Session Menu	3-15
Load Application Dialog	3-17
Application Information	3-18
Application Options	3-19
Partition Selection	3-20
Load Options	3-22
Load Status Messages	3-25
Control Buttons	3-25
Core Analysis Dialog	3-27
Preferences Dialog	3-33
Code Location Dialog	3-38
System Command Dialog	3-39

Defaults Menu	3-40
Filter Menu	3-42
Display Menu	3-43
Pending Sends Dialog	3-45
Pending Receives Dialog	3-46
Traceback Dialog	3-47
Tools Menu	3-49
XProf Dialog	3-50
XGprof Dialog	3-51
ParaGraph Dialog	3-52
Create Performance Data Dialog	3-54
Help Menu	3-58
Help Index	3-60
Help Topic	3-61
Legend	3-62
Process Type Selection	3-63
Communicator Selection	3-63
Execution Controls	3-64
Node Viewpoint Panel	3-65
Routine List	3-67
Floating the Routine List	3-69
Console Input/Output	3-70
Control Buttons	3-70
Message Area	3-71
File Selection	3-71
Code Window	3-73
Data Display	3-77
Data Modification	3-79
Show Communicator	3-80
Data Watch Point	3-81

Configuring XIPD	3-82
Default Configuration	3-87
Environment Variables	3-89

Chapter 4

Program Profiling: prof and gprof

prof	4-1
Invoking prof	4-2
Sample prof Output	4-3
Standard Output	4-3
Enhanced (Multi-Thread) Output	4-4
Overhead Routines	4-6
gprof	4-6
Invoking gprof	4-7
Sample gprof Output	4-9

Chapter 5

XProf

Using XProf	5-2
Invoking XProf	5-2
Choosing a Profile Output Directory	5-4
Setting prof Runtime Options	5-4
Selecting a Profile Output File	5-4
Examining prof Output	5-5
Windows, Menus, and Commands	5-5

Main Window	5-5
File Menu	5-6
Select Profile Directory	5-6
Options Menu	5-8
Enter prof Settings	5-9
Help Menu	5-11
Help Index	5-12
Help Topic	5-13
Profile Directory Contents List	5-14
Prof Output Window	5-15
Output Window File Menu	5-16
Save prof Output Dialog	5-17
Output Window Help Menu	5-18
Prof Output	5-19
Configuring XProf	5-19
Default Configuration	5-20

Chapter 6

XGprof

Using XGprof	6-2
Invoking XGprof	6-2
Choosing a Profile Output Directory	6-3
Setting gprof Runtime Options	6-4
Selecting a Profile Output File	6-4
Examining gprof Output	6-4
Windows, Menus, and Commands	6-4

Main Window6-5

- File Menu6-5
 - Select Profile Directory6-6
- Options Menu6-8
 - gprof Settings6-8
- Help Menu6-11
 - Help Index6-12
 - Help Topic6-13
- Profile Directory Contents List6-14

Gprof Output Window6-15

- Output Window File Menu6-16
 - Save gprof Output Dialog6-17
- Output Window Help Menu6-18
- Prof Output6-19

Configuring XGprof6-19

- Default Configuration6-20

Chapter 7 ParaGraph

Overview7-1

Invoking ParaGraph7-2

Display Overview7-3

- Utilization Displays7-3
- Communication Displays7-4
- Task Displays7-5
- Other Displays7-5

Windows, Menus & Commands	7-6
File Menu	7-7
Open	7-8
Save Layout	7-10
Load Layout	7-11
Exit	7-11
Options Menu	7-11
Configure	7-12
Select Nodes	7-16
Colors	7-17
Message Log	7-19
Trace Filter	7-21
Close All	7-23
Utilization, Communication, Task and Other Menus	7-23
Help Menu	7-25
Help Index	7-26
Help Topic	7-27
ParaGraph displays	7-29
Utilization Displays	7-29
Count	7-30
Gantt	7-31
Kiviat	7-32
Summary	7-33
Meter	7-34
Profile	7-34
Communication displays	7-35
Traffic	7-35
Spacetime	7-36
Queues	7-37
Matrix	7-38
Communication Meter	7-39
Animation	7-40

Topology	7-41
Network	7-42
Node Info	7-44
Color Code	7-45
Task displays	7-46
Task Count	7-47
Task Gantt	7-47
Task Status	7-48
Task Summary	7-49
Other Displays	7-50
Clock	7-50
Trace	7-50
Statistical Summary	7-51
Processor Status	7-52
Phase Portrait	7-53
Info	7-54
Hints for Using ParaGraph	7-56
Interpretation of Trace Events	7-56
Trace Size	7-56
Performance Monitoring Buffer Size	7-57
Parameters	7-57
Restrictions	7-58
Window Placement and Window Managers	7-59
Possible Problems	7-59
Generating Traces	7-59
Page Warmup	7-60
Intrusion Caused by Flushing Buffers	7-60
Global Clock	7-60
Busy waiting loops	7-61
Use of Colors	7-61

Configuring ParaGraph	7-61
Default Configuration	7-64

Chapter 8

The Parallel Make Utility

Invoking pmake	8-2
pmake Extensions to GNU make	8-5
Parallel Controls	8-5
Using a Compute Partition	8-5
Using the Service Partition	8-7
Controlling System Loading	8-7
Macro Extensions	8-8
Special Macro	8-8
Pattern Replacement	8-8
Modifiers	8-8
Conditional Expressions	8-8
Configuration File Support	8-9
Other Differences Between pmake and GNU make	8-10
Command Line Options	8-10
Special Targets	8-10
Include Statement	8-11
The makefile Description File	8-11
Dependency Lines	8-12
Commands	8-13
Included Description Files	8-13
Macros	8-14
Special Macros	8-15
Special Variables	8-15
Pseudotarget Names	8-16
Conditional Execution	8-17

List of Illustrations

Figure 1-1.	Main Window	1-4
Figure 1-2.	File Menu	1-5
Figure 1-3.	Open History Dialog	1-6
Figure 1-4.	Mesh Menu	1-7
Figure 1-5.	Load Application Dialog	1-8
Figure 1-6.	Application Options Section of Load Application Dialog	1-10
Figure 1-7.	Partition Selection Section of Load Application Dialog	1-12
Figure 1-8.	Load Options Section of Load Application Dialog	1-14
Figure 1-9.	New Mesh Process Type Dialog	1-15
Figure 1-10.	Load History Dialog	1-18
Figure 1-11.	Tool Menu	1-19
Figure 1-12.	Enter XIPD Options Dialog	1-20
Figure 1-13.	Go to ParaGraph Dialog	1-22
Figure 1-14.	Go to XProf and Go to XGprof Dialogs	1-24
Figure 1-15.	Command Menu	1-25
Figure 1-16.	Command Viewer Dialog	1-27
Figure 1-17.	Help Menu	1-28
Figure 1-18.	Help Index Dialog	1-30
Figure 1-19.	Help Topic Dialog	1-31
Figure 1-20.	Icon Strip	1-32
Figure 1-21.	Terminal Area	1-33
Figure 1-22.	Select a File to Open Dialog	1-35
Figure 2-1.	Shadow Columns in the Gauss-Seidel Example	2-26
Figure 3-1.	Start-up Dialog	3-6
Figure 3-2.	Session Name Dialog	3-8
Figure 3-3.	XIPD Main Window	3-9
Figure 3-4.	Main Window	3-14
Figure 3-5.	Session Menu	3-16

List of Illustrations

Figure 3-6.	Load Application Dialog	3-18
Figure 3-7.	Application Options Section of Load Application Dialog	3-19
Figure 3-8.	Partition Selection Section of Load Application Dialog	3-21
Figure 3-9.	Load Options Section of Load Application Dialog	3-23
Figure 3-10.	New Mesh Process Type Dialog	3-24
Figure 3-11.	Core Analysis Dialog	3-28
Figure 3-12.	Core File Summary	3-32
Figure 3-13.	Core File Summary With Thread Information	3-33
Figure 3-14.	Preferences Dialog	3-34
Figure 3-15.	Code Location Dialog	3-38
Figure 3-16.	System Command Dialog	3-40
Figure 3-17.	Defaults Menu	3-41
Figure 3-18.	Filter Menu	3-43
Figure 3-19.	Display Menu	3-44
Figure 3-20.	Pending Sends Dialog	3-45
Figure 3-21.	Pending Receives Dialog	3-46
Figure 3-22.	Traceback Dialog	3-48
Figure 3-23.	Traceback Dialog With Multiple Threads	3-48
Figure 3-24.	Tools Menu	3-49
Figure 3-25.	XProf Dialog	3-50
Figure 3-26.	XGprof Dialog	3-51
Figure 3-27.	Enter ParaGraph Settings Dialog	3-52
Figure 3-28.	Create Performance Data Dialog	3-54
Figure 3-29.	Help Menu	3-59
Figure 3-30.	Help Index Dialog	3-60
Figure 3-31.	Help Topic Dialog	3-61
Figure 3-32.	Main Window Legend	3-62
Figure 3-33.	Execution Controls	3-64
Figure 3-34.	Node Viewpoint	3-66
Figure 3-35.	Nodes With and Without Multiple Threads	3-66

List of Illustrations

Figure 3-36.	Node Information Panel	3-67
Figure 3-37.	Node Information Panel With Show Threads Enabled	3-67
Figure 3-38.	Routine List	3-68
Figure 3-39.	Filtered Routine List	3-69
Figure 3-40.	File Selection Dialog	3-71
Figure 3-41.	Code Window	3-73
Figure 3-42.	Code Menu (NX Applications)	3-74
Figure 3-43.	Action Menu (MPI Applications)	3-75
Figure 3-44.	Action Menu	3-75
Figure 3-45.	Code Window Help Menu	3-76
Figure 3-46.	Data Display Dialog	3-78
Figure 3-47.	Data Modification Dialog	3-79
Figure 3-48.	Show Communicator Dialog	3-81
Figure 3-49.	Data Watchpoint Dialog	3-81
Figure 5-1.	XProf Main Window	5-5
Figure 5-2.	File Menu	5-6
Figure 5-3.	Select Profile Directory Dialog	5-7
Figure 5-4.	Options Menu	5-8
Figure 5-5.	Enter prof Settings Dialog	5-9
Figure 5-6.	Help Menu	5-11
Figure 5-7.	Help Index Dialog	5-12
Figure 5-8.	Help Topic Dialog	5-13
Figure 5-9.	Profile Output File List	5-14
Figure 5-10.	Output Window	5-15
Figure 5-11.	Output Window File Menu	5-16
Figure 5-12.	Save prof Output Dialog	5-17
Figure 5-13.	Output Window Help Menu	5-18

List of Illustrations

Figure 6-1.	Main Window	6-5
Figure 6-2.	File Menu	6-5
Figure 6-3.	Select Profile Directory Dialog	6-6
Figure 6-4.	Options Menu	6-8
Figure 6-5.	Enter gprof Settings Dialog	6-9
Figure 6-6.	Help Menu	6-11
Figure 6-7.	Help Index Dialog	6-12
Figure 6-8.	Help Topic Dialog	6-13
Figure 6-9.	Profile Output File List	6-14
Figure 6-10.	Output Window	6-15
Figure 6-11.	Output Window File Menu	6-16
Figure 6-12.	Save gprof Output Dialog	6-17
Figure 6-13.	Output Window Help Menu	6-18
Figure 7-1.	ParaGraph Main Window	7-6
Figure 7-2.	File Menu	7-8
Figure 7-3.	Open Tracefile Dialog	7-9
Figure 7-4.	Example Layout File	7-10
Figure 7-5.	Set ParaGraph Options Dialog	7-12
Figure 7-6.	Select Nodes Dialog	7-16
Figure 7-7.	Select Colors Dialog	7-17
Figure 7-8.	Message Log Window	7-19
Figure 7-9.	Filter Trace Records Dialog	7-21
Figure 7-10.	Utilization Menu	7-23
Figure 7-11.	Help Menu	7-25
Figure 7-12.	Help Index	7-27
Figure 7-13.	Help Topic Dialog	7-28
Figure 7-14.	Count Display	7-30
Figure 7-15.	Gantt Display	7-31
Figure 7-16.	Kiviat Display	7-32

List of Illustrations

Figure 7-17.	Summary Display	7-33
Figure 7-18.	Meter Display	7-34
Figure 7-19.	Profile Display	7-35
Figure 7-20.	Traffic Display	7-36
Figure 7-21.	Spacetime Display	7-37
Figure 7-22.	Queues Display	7-38
Figure 7-23.	Matrix Display	7-38
Figure 7-24.	Communication Meter Display	7-39
Figure 7-25.	Animation Display	7-40
Figure 7-26.	Topology Display	7-41
Figure 7-27.	Network Display	7-42
Figure 7-28.	Node Info Display	7-44
Figure 7-29.	Color Code Display	7-45
Figure 7-30.	Task Count Display	7-47
Figure 7-31.	Task Gantt Display	7-48
Figure 7-32.	Task Status Display	7-48
Figure 7-33.	Task Summary Display	7-49
Figure 7-34.	Clock Display	7-50
Figure 7-35.	Trace Display	7-51
Figure 7-36.	Statistical Summary Display	7-52
Figure 7-37.	Processor Status Display	7-52
Figure 7-38.	Phase Portrait Display	7-53
Figure 7-39.	Info Display	7-54

List of Tables

Table 1-1.	Display Resources	1-37
Table 1-2.	Size Resources	1-37
Table 1-3.	Partition Resources	1-37
Table 1-4.	Default ParAide Resource Settings	1-39
Table 2-1.	Execution Control Commands	2-3
Table 2-2.	Program Examination and Modification Commands	2-4
Table 2-3.	Debug Environment Commands	2-5
Table 2-4.	Fortran Variable Type Display	2-25
Table 3-1.	XIPD Support of IPD Commands	3-2
Table 3-2.	XIPD Color Resources	3-83
Table 3-3.	XIPD Pattern Resources	3-84
Table 3-4.	XIPD Font Resources	3-85
Table 3-5.	XIPD Size Resources	3-85
Table 3-6.	Other XIPD Resources	3-86
Table 3-7.	Default XIPD Resource Settings	3-87
Table 5-1.	XProf Application Resources	5-19
Table 5-2.	Default XProf Resource Settings	5-20
Table 6-1.	XGprof Application Resources	6-19
Table 6-2.	Default XGprof Resource Settings	6-20



This chapter describes ParAide, the graphical user interface that serves as a launching point to the other graphical tools in the Paragon™ system 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 section “Windows, Menus, and Commands” on page 1-4.

Invoking ParAide

To invoke ParAide on the Paragon system do the following:

1. Enter the following command on your workstation:

```
% xhost + paragon_system
```

where *paragon_system* is the name of the Paragon system on which you are going to run ParAide.

2. Log onto the Paragon system.

3. Set the *DISPLAY* environment variable to your workstation as in the following example:

```
% setenv DISPLAY machine_name : 0
```

where *machine_name* is the name of your workstation.

4. Check to be sure you have */usr/bin/X11* in your search path.
5. Invoke ParAide.

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

```
paraide [-nomenus] [-noicons] [-noshell] [-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 terminal area. **-shell** is the default.
- rows** *minrows* Sets the minimum number of rows for the terminal area 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 terminal area 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 on page 1-4 shows the 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 <i>File</i> , <i>Mesh</i> , <i>Tool</i> , and <i>Help</i> menus.
Icon strip	Allows you to quickly bring up other graphical tools from the Paragon toolset, edit a file, or run an application.
Terminal area	Contains a user shell running on the Paragon system.
Status line	Contains messages about work in progress.

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, XIPD-lite, ParaGraph, XProf, and XGprof.

Loading Applications

The *Mesh* menu provides access to the Load Application dialog to control loading an application into the mesh.

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

The load command is inserted into the "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 terminal area again. This avoids re-using the Load Application 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 1-1 shows the ParAide main window.

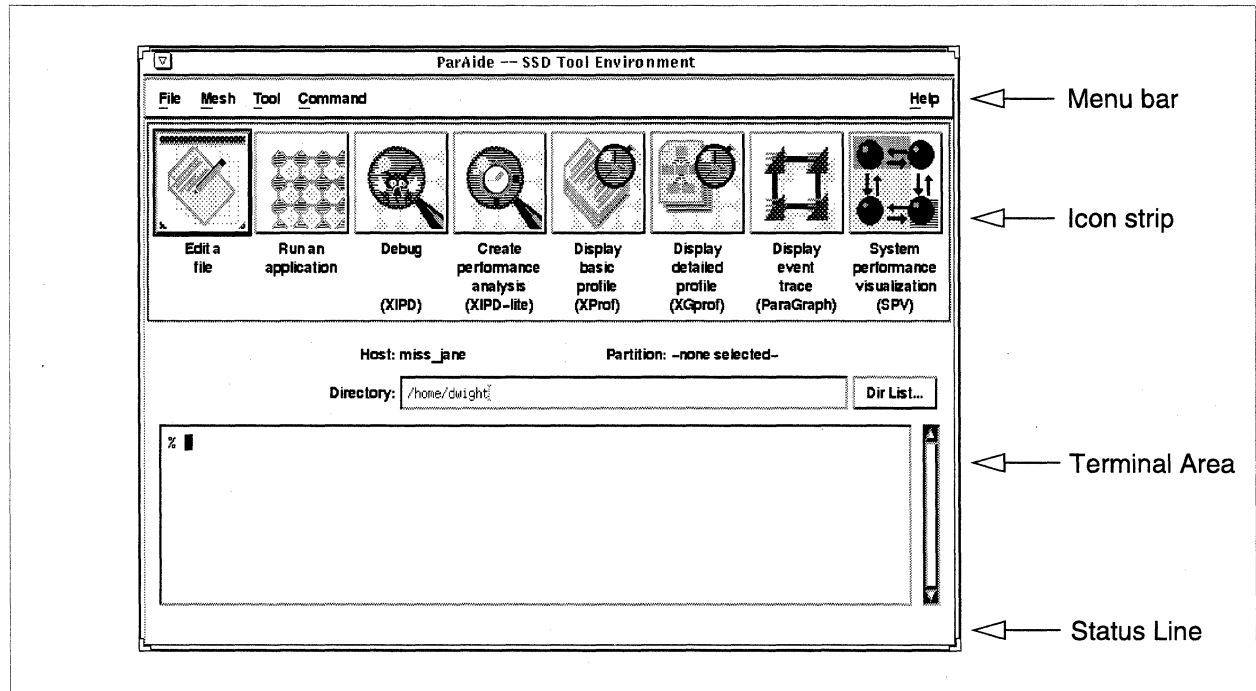


Figure 1-1. Main Window

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 doing the following:

- Editing a file
- Running an application
- Invoking XIPD
- Invoking XIPD in performance-analysis only mode (XIPD-lite)
- Invoking XProf
- Invoking XGprof
- Invoking ParaGraph
- Invoking SPV

The terminal area runs like an embedded xterm, providing support for interactive jobs and job control.

The status line contains messages about work in progress.

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 1-2 shows the *File* menu.

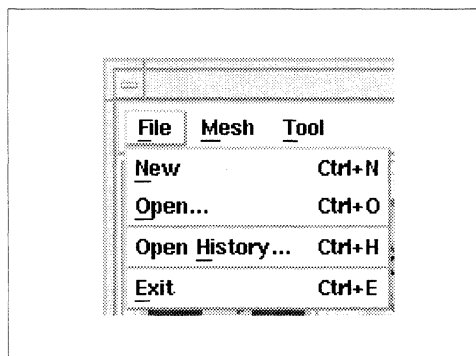


Figure 1-2. File Menu

New

New brings up a new, empty editing program. The default keyboard accelerator for *New* is **<Ctrl-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 **<Ctrl-O>**. The file selection dialog is described in the section "Selecting Files and Directories" on page 1-34.

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 **<Ctrl-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 **<Ctrl-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 1-3 shows the *Open History* dialog.

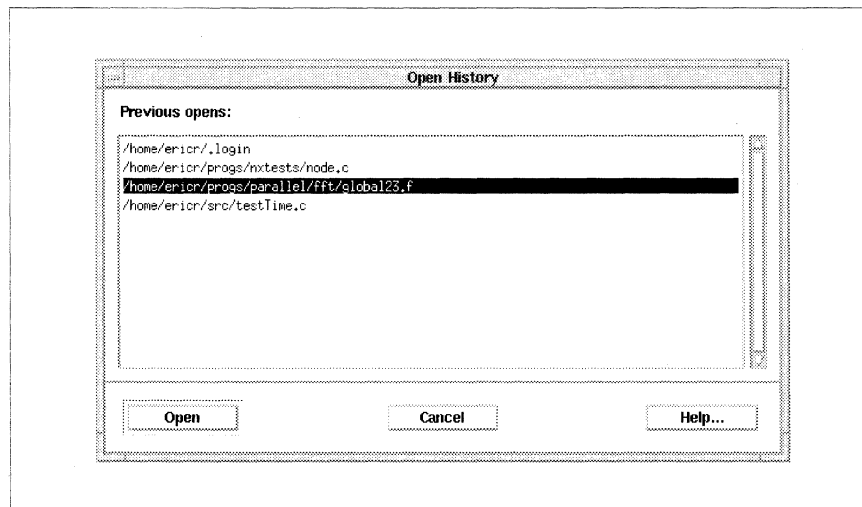


Figure 1-3. Open History Dialog

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 load programs into the mesh. Figure 1-4 shows the *Mesh* menu.

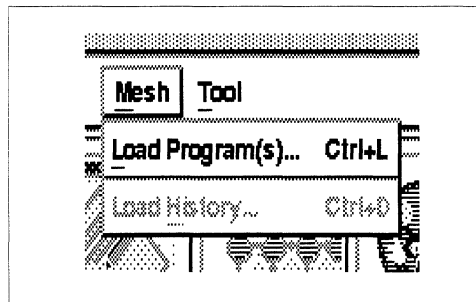


Figure 1-4. Mesh Menu

Load Program(s)

Load Program(s) display the *Load Application* dialog. The default keyboard accelerator for *Load Program(s)* is <Ctrl-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 Application* dialog. The default keyboard accelerator for *Load History* is <Ctrl-D>.

Load Application Dialog

The Load Application dialog allows you to load an application. You can bring up the *Load Application* dialog with the *Load Program(s)* item from the Mesh menu. The *Load Application* dialog contains the following regions:

- Application Information
- Application Options
- Partition Selection

- Load Options
- Load Status Messages
- Control Buttons

Figure 1-5 shows the *Load Application* dialog.

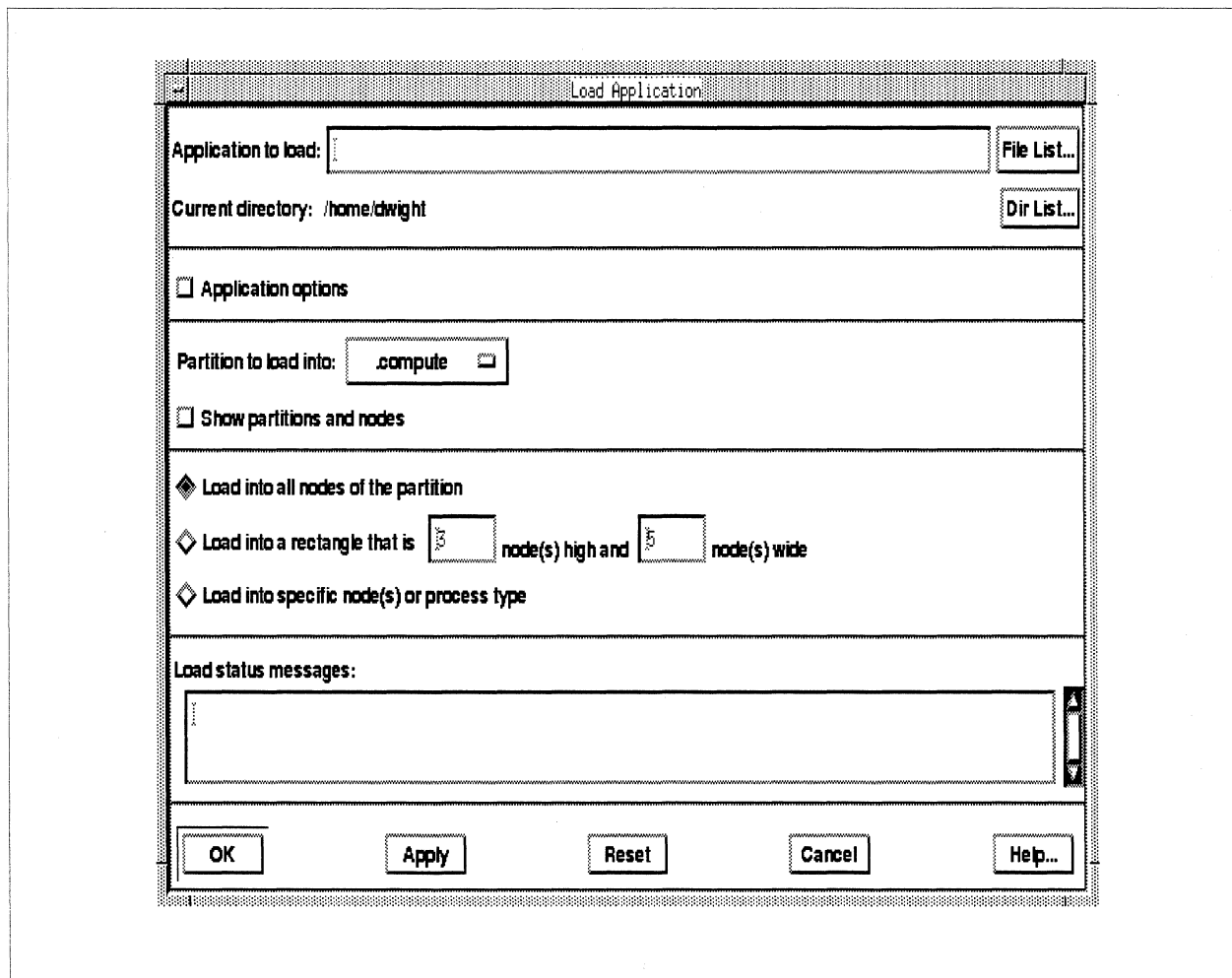


Figure 1-5. Load Application Dialog

Application Information

The application information area of the *Load Application* dialog allows you to do the following:

- specify an application to load
- identify the current directory

Application to Load

The *Application to Load* text field allows you to enter the name of the application you wish to load. Next to this text field is the *File List* button. Selecting this button displays the file selection dialog. If you select a file from the file selection dialog, the entry in the *Application to Load* text field is updated. The file selection dialog is described in the section “Selecting Files and Directories” on page 1-34.

Current Directory

The *Current Directory* field lists the current directory. Next to this field is the *Dir List* button. Selecting this button displays the file selection dialog. If you select a directory from the file selection dialog, the *Current Directory* field is updated. The file selection dialog is described in the section “Selecting Files and Directories” on page 1-34.

Application Options

The Application Options area of the *Load Application* dialog is an expandable area that allows you to do the following:

- specify command line arguments
- specify an input file for redirection
- specify an output file for redirection
- specify if the application is a controlling process

When the *Load Application* dialog appears, the Application Options section only contains a toggle button. When you set this button, an area containing the application options opens under the toggle button. If you reset the button, this area is collapsed and only the toggle button is visible.

Figure 1-6 shows the expanded Application Options section.

Application options

Arguments:

Input file: File List...

Output file: File List...

Controlling / service application

Figure 1-6. Application Options Section of Load Application Dialog

Arguments

The *Arguments* text field allows you to specify command line arguments to be passed to the application being loaded.

Input file

The *Input File* text field allows you to specify the name of a file to be used for input redirection. If you do not want to use input redirection, this field should be left blank. Next to the *Input File* text field is a *File List* button. Selecting this button displays the file selection dialog. If you select a file from the file selection dialog, the entry in the *Input File* text field is updated. For a description of the file selection dialog, refer to the Section “Selecting Files and Directories” on page 1-34.

Output file

The *Output File* text field allows you to specify the name of a file to be used for output redirection. If you do not want to use output redirection, this field should be left blank. Next to the *Output File* text field is a *File List* button. Selecting this button displays the file selection dialog. If you select a file from the file selection dialog, the entry in the *Output File* text field is updated. For a description of the file selection dialog, refer to the Section “Selecting Files and Directories” on page 1-34.

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/service application

The *Controlling/service application* toggle button should be set if the application being loaded is a controlling process.

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 set the *Controlling/service application* toggle button to suppress passing these options.

Partition Selection

The Partition Selection area of the *Load Application* dialog allows you to select in which partition the application should run. This area allows you to do the following:

- specify the partition to load into
- show partitions and nodes

Partition to Load Into

Partition to Load Into contains the name of the currently selected partition and a pull-down menu that includes the names of all partitions in the *.compute* partition. Selecting a name from the menu updates the currently selected partition.

Show Partitions and nodes

If you set the *Show partitions and nodes* toggle button in the Partition Selection area, a graphical display of the partition tree appears under the toggle button. This display allows you to do the following:

- select a partition
- view the partition location

Figure 1-7 shows the expanded Partition Selection section.

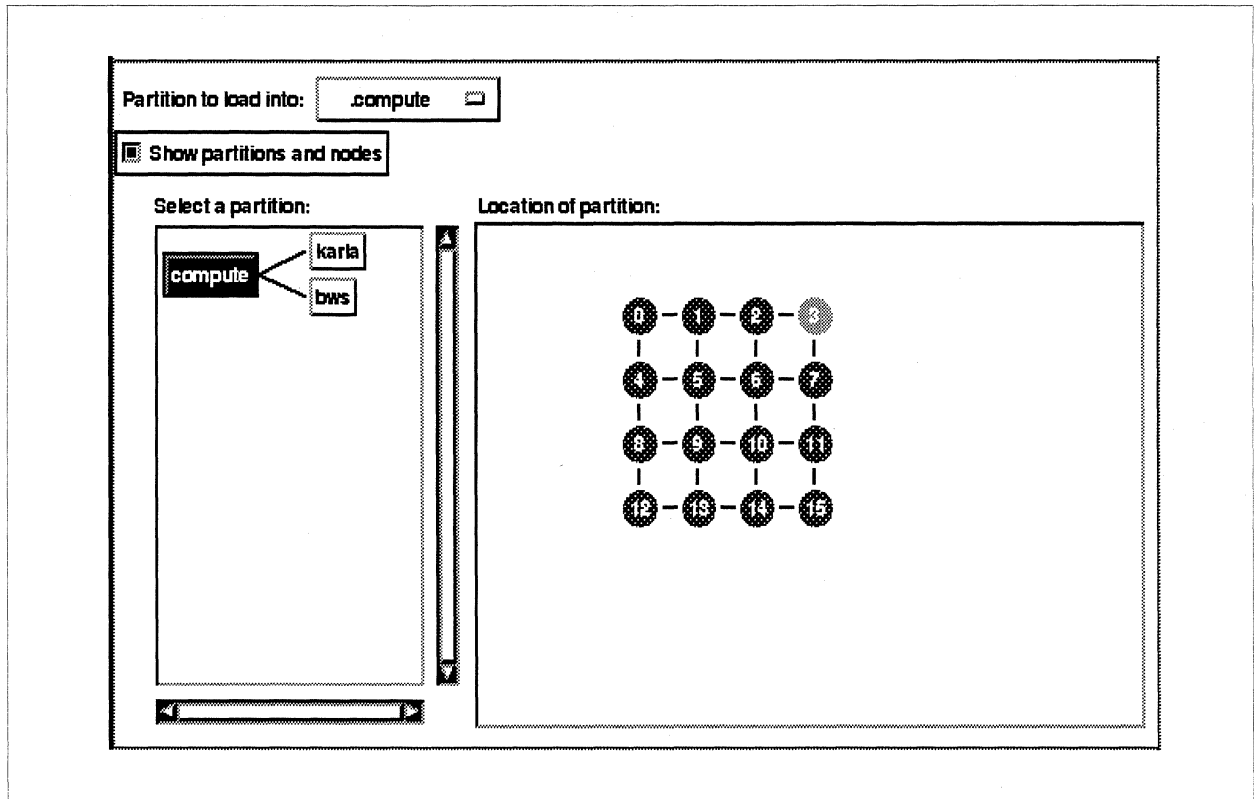


Figure 1-7. Partition Selection Section of Load Application Dialog

Select a Partition

This area shows a graphical display of the partitions allocated under the *.compute* partition. The currently-selected partition is highlighted. When you select a partition, the *Partition to Load Into* field is updated.

Location of Partition

This area shows a graphical representation of the physical mesh, highlighting the nodes in the currently-selected partition. The colors and patterns used to highlight the nodes are display-specific.

Load Options

The Load Options area of the *Load Application* dialog allows you to specify which nodes in the selected partition should be loaded with the application. You can do one of the following:

- load into all nodes
- load into a rectangle of nodes
- load into specific nodes

Initially, this area contains only three radio buttons for the three load options, and the load into all nodes option is set.

Load into All Nodes

If this radio button is set, the application is loaded onto all nodes in the selected partition.

Load into Rectangle

The *Load into a rectangle* option allows you to specify a rectangle of nodes into which the application is loaded. You select this option by setting the radio button and specifying the height and width of the rectangle in the adjacent boxes. You must be sure that the dimensions you specify for the rectangle are valid for the selected partition. If you specify an invalid rectangle, you will receive an allocator error message when ParAide attempts to load your application into the rectangle.

Load into Specific Nodes

If you set the radio button to load into specific nodes, the Load Options area expands to include the following:

- a node panel to select specific nodes
- a process type option menu
- a new process type push button
- buttons to set load mesh height and width
- a button to select all nodes
- a button to unselect all nodes

Figure 1-8 shows the expanded Load Options section.

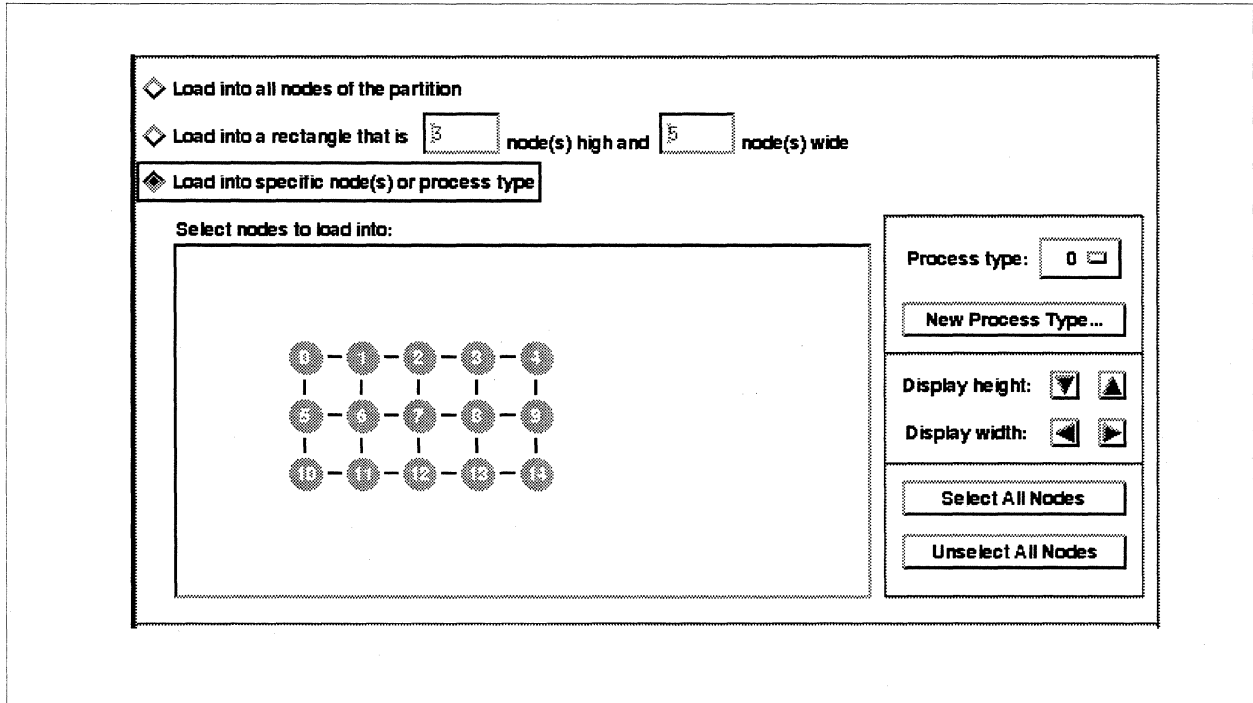


Figure 1-8. Load Options Section of Load Application Dialog

Node Selection

The node panel allows you to select specific nodes to load your application into. The node panel contains a representation of the nodes in the selected partition. The nodes are displayed as a mesh. The height and width of the mesh is logical (based on the number of nodes in the mesh), since the nodes in the partition may actually be scattered in a non-rectangular pattern. When you select specific nodes for loading, your selection must start with the first node in the specified partition, and the nodes selected must be contiguous.

Nodes in the display are in one of three states:

- Unselected** Can be loaded into
- Selected** Will be loaded at the next *OK* or *Apply*
- Loaded** Already loaded and can't be loaded again

The colors and patterns used to render the state of the nodes are display-specific.

You can select a node by clicking on it. You can select a group of nodes by holding down the first mouse button and dragging the pointer over the group. Clicking on a selected node deselects the node.

Process Type Selection

The *Process type* menu allows you to choose a process type. Each process type allocated with the *New Process Type* push button is given a specific set of nodes for loading. When you select a process type from the *Process type* menu, the node panel is updated to reflect the status of the nodes associated with that process type.

New Process Type

When you select the *New Process Type* push button, a dialog appears to allow you to specify a new process type. Enter a number in the dialog box and press the dialog's *OK* button to allocate a new process type for loading into. This new process type is then incorporated onto the process type menu.

Figure 1-9 shows the *New Mesh Process Type* dialog.

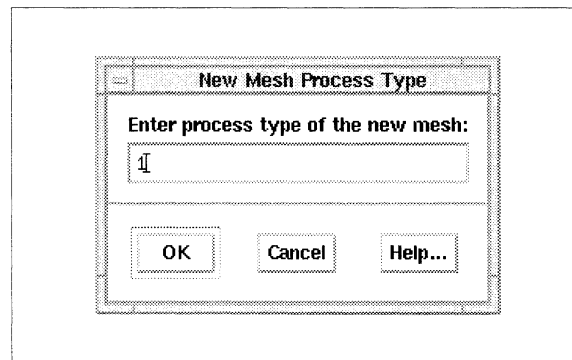


Figure 1-9. New Mesh Process Type Dialog

Display Height / Width

The *Display height* and *Display width* arrows have the following effects on the load mesh:

Display height down	decreases the height and increases the width
Display height up	increases the height and decreases the width
Display width left	decreases the width and increases the height
Display width right	increases the width and decreases the height

If it is not possible to do the requested change on the load mesh, a beep will sound.

Select All Nodes

The *Select All Nodes* push button changes all nodes in the partition that are not currently in the *loaded* state to the *selected* state.

Unselect All Nodes

The *Unselect All Nodes* push button changes all nodes in the partition that are not currently in the *loaded* state to the *unselected* state.

Load Status Messages

The Load Status Messages area of the *Load Application* dialog is a scrolling text area that contains messages about the current state of the application load process. This area clears every time the dialog appears. You can not edit the text in this area, but you can select it and paste it into another X client.

Control Buttons

The control buttons on the *Load Application* dialog have the following functions:

OK

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

- No file name is given.
- No nodes are selected and *Controlling/service application* is not set.
- Nodes are selected and *Controlling/service application* 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 terminal area.

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/service application* is set.
- No nodes are selected and *Controlling/service application* is not set.

Reset

Reset eliminates all changes since the last *Apply* or since the dialog appeared, whichever is most recent.

Cancel

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

Help

Help displays help topic text about the *Load Application* 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 1-10 shows the *Load History* dialog.

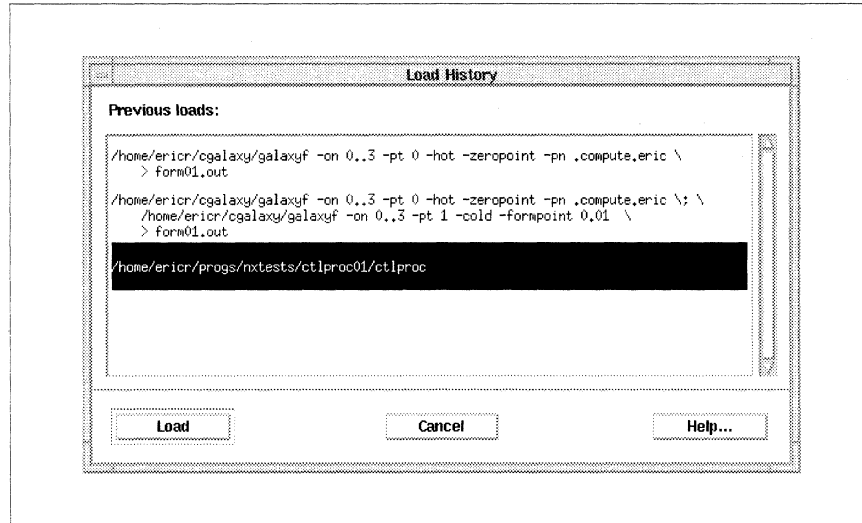


Figure 1-10. Load History Dialog

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 terminal area 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 1-15 shows the *Tool* menu.

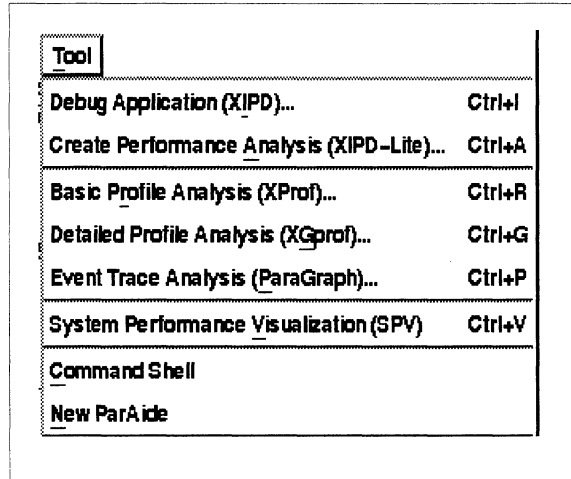


Figure 1-11. Tool Menu

Debug application (XIPD)

This menu item displays the XIPD dialog, with the *performance analysis only* option not chosen. The default keyboard accelerator for this item is <Ctrl-I>.

Create performance analysis (XIPD-Lite)

This menu item displays the XIPD dialog, with the *performance analysis only* option chosen. The default keyboard accelerator for this item is <Ctrl-A>.

Basic profile analysis (XProf)

This menu item displays the *Go to XProf* dialog. The default keyboard accelerator for this item is <Ctrl-R>.

Detailed profile analysis (XGprof)

This menu item displays the *Go to XGprof* dialog. The default keyboard accelerator for this item is <Ctrl-G>.

Event trace analysis (ParaGraph)

This menu item displays the *Go to ParaGraph* dialog. The default keyboard accelerator for this item is <Ctrl-P>.

System performance visualization (SPV)

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*. The default keyboard accelerator for this item is <Ctrl-V>.

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 1-12 shows the *Enter XIPD Options* dialog.

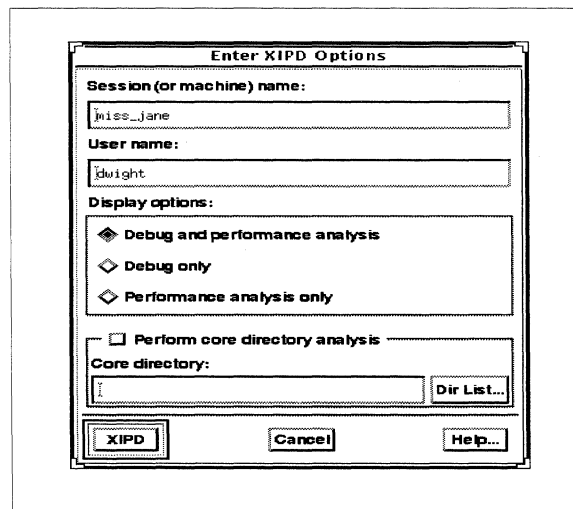


Figure 1-12. Enter XIPD Options Dialog

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 on which ParAide is running.

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 profiling elements of XIPD by selecting *Debug only* mode. The valid display options are the following:

<i>Debug and profile</i>	Enables both debugging and performance profiling interface elements.
<i>Debug only</i>	Enables only debugging interface elements.
<i>Profile only</i>	Enables only profiling interface elements.

Perform core directory analysis

Selecting the *Perform core directory analysis* button allows you to analyze a core file directory.

The *Core directory* text field allows you to specify the core directory for analysis. This field is enabled only if the *Perform core directory analysis* button is set. If you leave the field blank, XIPD selects the directory specified by the **XIpd.coreDirectoryName** resource.

If you select the *Directory List* button, XIPD brings up the file selection dialog. If you select a directory from the dialog, XIPD puts the selection into the *Core directory* text field.

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 1-13 shows the *Go to ParaGraph* dialog.

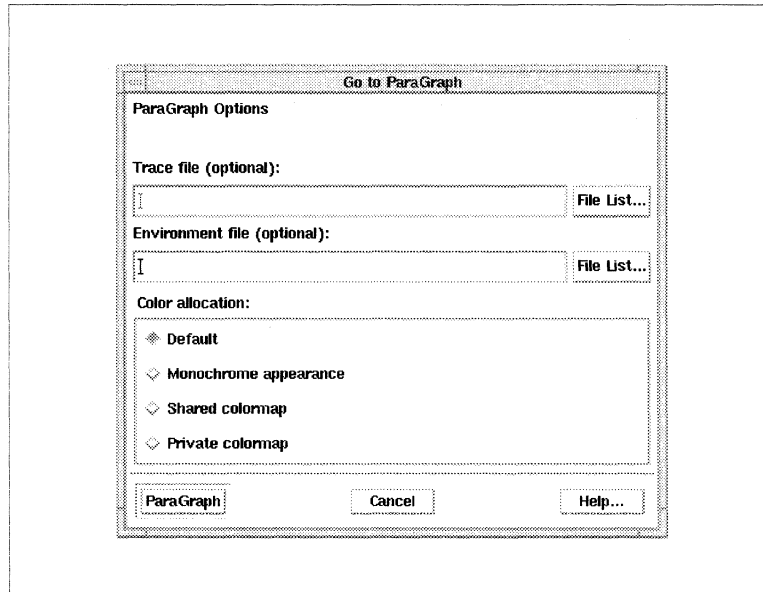


Figure 1-13. Go to ParaGraph Dialog

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 file selection dialog. You can also leave the field blank. The file selection dialog is described in the section “Selecting Files and Directories” on page 1-34.

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 file selection dialog. You can also leave the field blank. The file selection dialog is described in the section “Selecting Files and Directories” on page 1-34.

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:

<i>Default</i>	ParaGraph executes without any specific color allocation directives.
<i>Monochrome appearance</i>	ParaGraph displays in monochrome (black and white).
<i>Shared colormap</i>	ParaGraph executes with the shared colormap. With this setting ParaGraph may not be able to allocate enough colors for certain functions.
<i>Private colormap</i>	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 1-14 shows the *Go to XProf* and *Go to XGprof* dialogs.

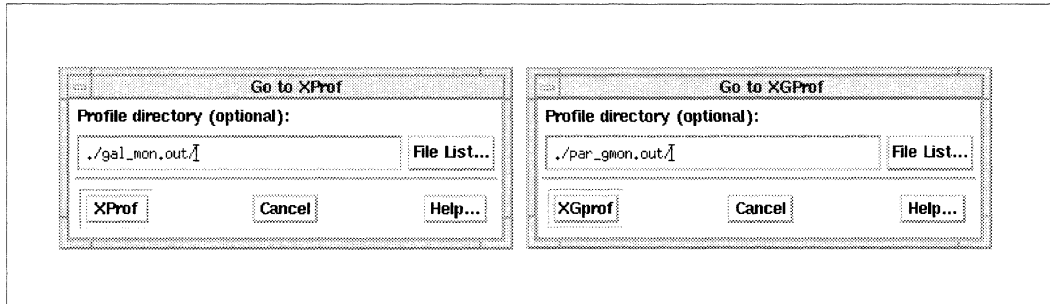


Figure 1-14. Go to XProf and Go to XGprof Dialogs

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 file selection dialog. You can also leave the field blank. The file selection dialog is described in the section “Selecting Files and Directories” on page 1-34.

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.

Command Menu

The Command menu gives you the ability to easily execute non-interactive commands. Figure 1-15 shows the *Command* menu.

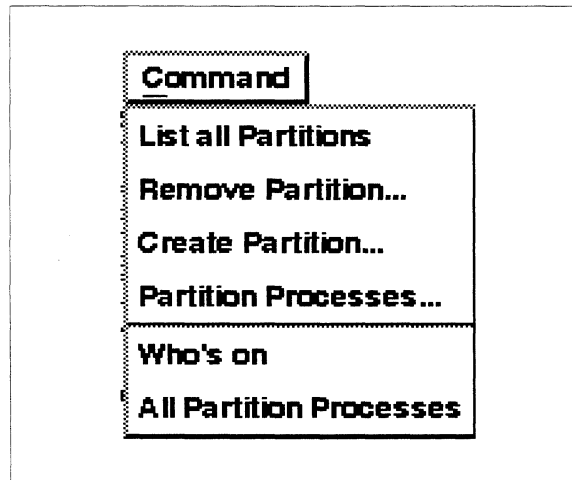


Figure 1-15. Command Menu

The list of commands in the *Command* menu is defined by the **Paraide.commandMenuItems** application resource, so you can customize the *Command* menu to include any non-interactive commands you choose. See the section *Customizing the Command Menu* for a description of how to use the **Paraide.commandMenuItems** application resource to define new *Command* menu items.

Customizing the Command Menu

You can use the **Paraide.commandMenuItems** application resource to customize the *Commands* menu to contain any non-interactive commands you choose. This resource is a multi-line resource, and each line defines an item in the *Command* menu. Each definition includes the text that appears for the menu item and what to do when the item is selected.

The general syntax for a command definition is as follows:

```
command(item_name) [[ask(question)] action(command_string)
```

The parameters are defined as follows:

- | | |
|------------------|---|
| <i>item_name</i> | The name that appears in the Command menu for the item. If you enter only a dash (-) as the <i>item_name</i> , a menu separator is created at that point in the menu. |
| <i>question</i> | The question text that appears in a prompt dialog when the menu item is selected. |

<i>action</i>	The action to be taken. <i>action</i> can be one of the following:
do	Executes the command.
view	Brings up the command viewer. See the section <i>Viewing Command Output</i> for a description of the command viewer dialog.
term	Sends the command being executed to the terminal area of the main window.

command_string The actual command to be executed.

The default application resource for the *Command* menu appears as follows, and serves as an example for designing your own menu items for the *Command* menu:

```
Paraide.commandMenuItems: \
command(List all Partitions) view(lspart -r .)\n\
command(Remove Partition...) ask(Partition to remove?)
do(rmpart -f $answer)\n\
command(Create Partition...) ask(mkpart arguments)\
do(mkpart $answer)\n\
command(Partition Processes...) ask(Processes for which\
partition?) view(pspart $answer)\n\
command(-)\n\
command(Who's on) view(who)\n\
command(All Partition Processes) view(pspart -r .)
```

Viewing Command Output

The *Command Viewer* dialog allows you to execute a non-interactive command and review the standard output and error output from the command.

Figure 1-3 shows the *Command Viewer* dialog.

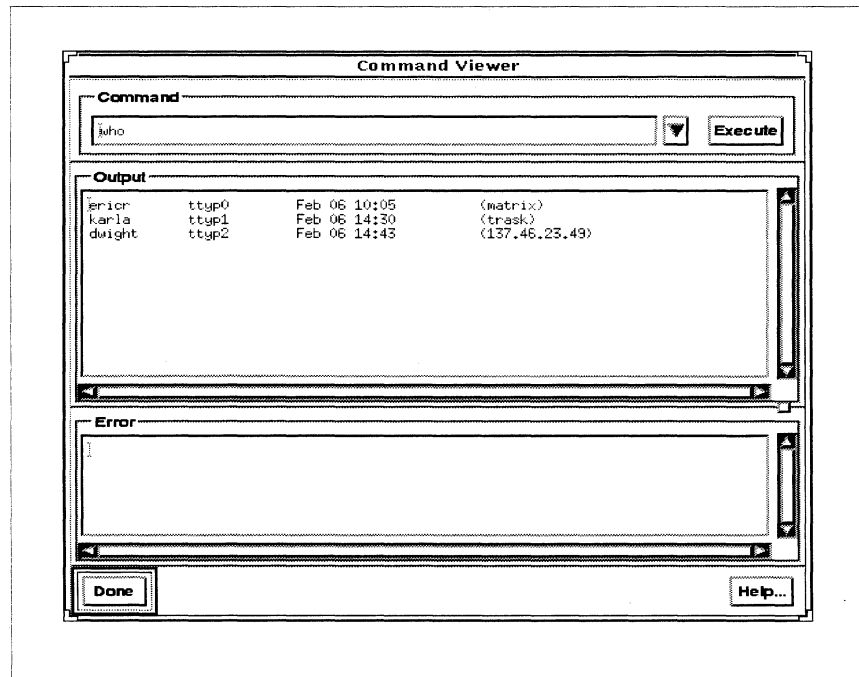


Figure 1-16. Command Viewer Dialog

Command

The *Command* field contains the command that has been executed or is to be executed. You can execute the command by pressing the **<Return>** key while this field has the keyboard focus, or you can select the *Execute* button.

History

When you press the *History* button (the button to the right of the *Command* field), a pop-up menu appears. This menu contains a list of commands that have been executed, with the most recent command first on the list. If you select a command from the list, the *Command* field, *Output* field, and *Error* field are restored to the result of the selected command.

Execute

If you select the *Execute* button, the command specified in the *Command* field is executed.

Output

The *Output* field contains any standard output read from executing the command specified in the *Command* field. You can not modify the text of the *Output* field.

Error

The *Error* field contains any standard error output read from executing the command specified in the *Command* field. You can not modify the text of the *Error* field.

Done

Selecting the *Done* button dismisses the *Command Viewer* dialog.

Help

Selecting the *Help* button displays help text for the *Command Viewer* 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 1-17 shows the *Help* menu.

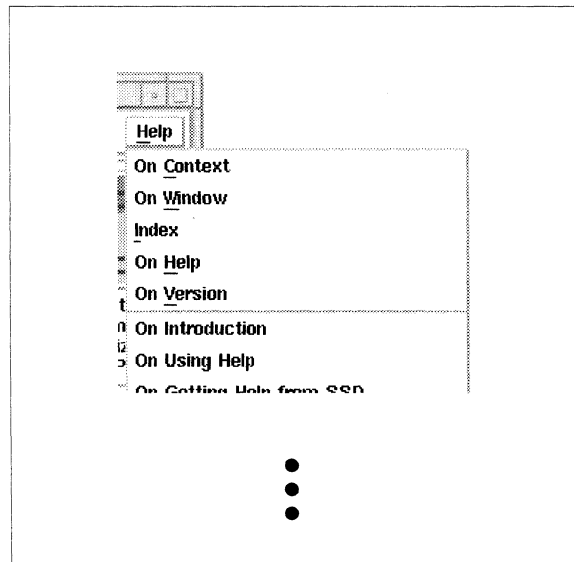


Figure 1-17. Help Menu

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 1-18 shows the *Index* dialog.

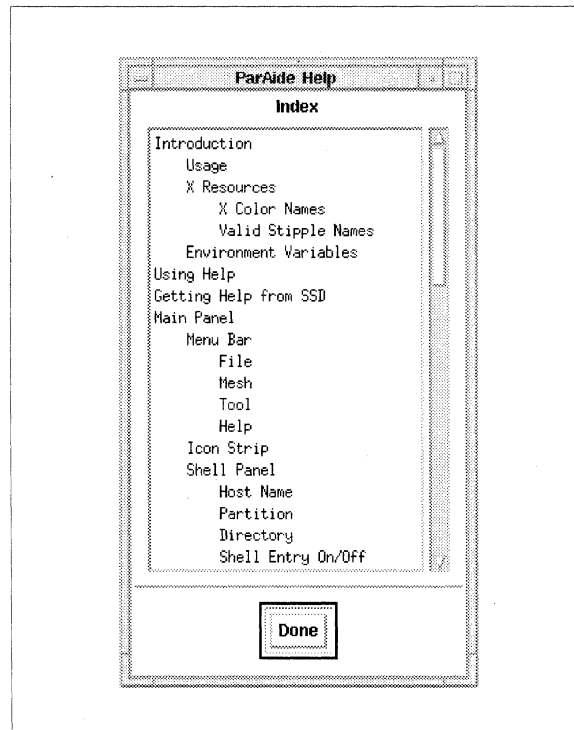


Figure 1-18. Help Index Dialog

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 *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 1-19 shows a help topic dialog.

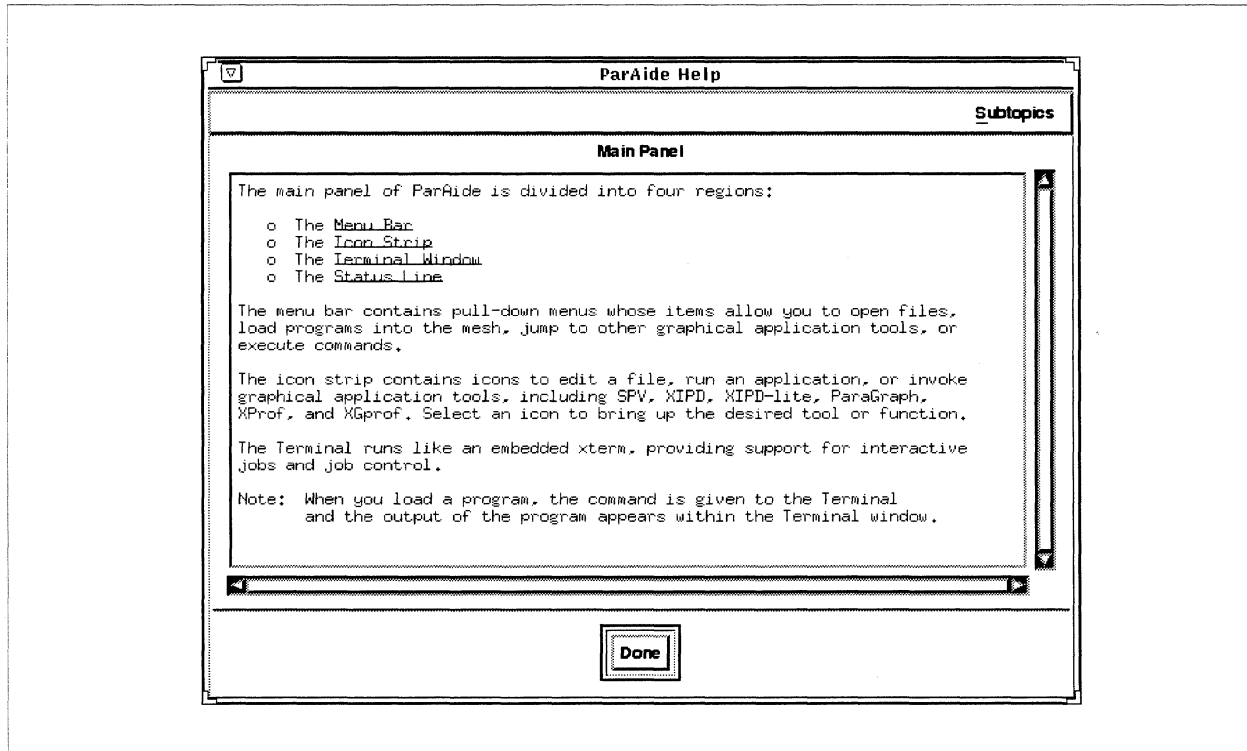


Figure 1-19. Help Topic Dialog

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 1-20 shows the icon strip.

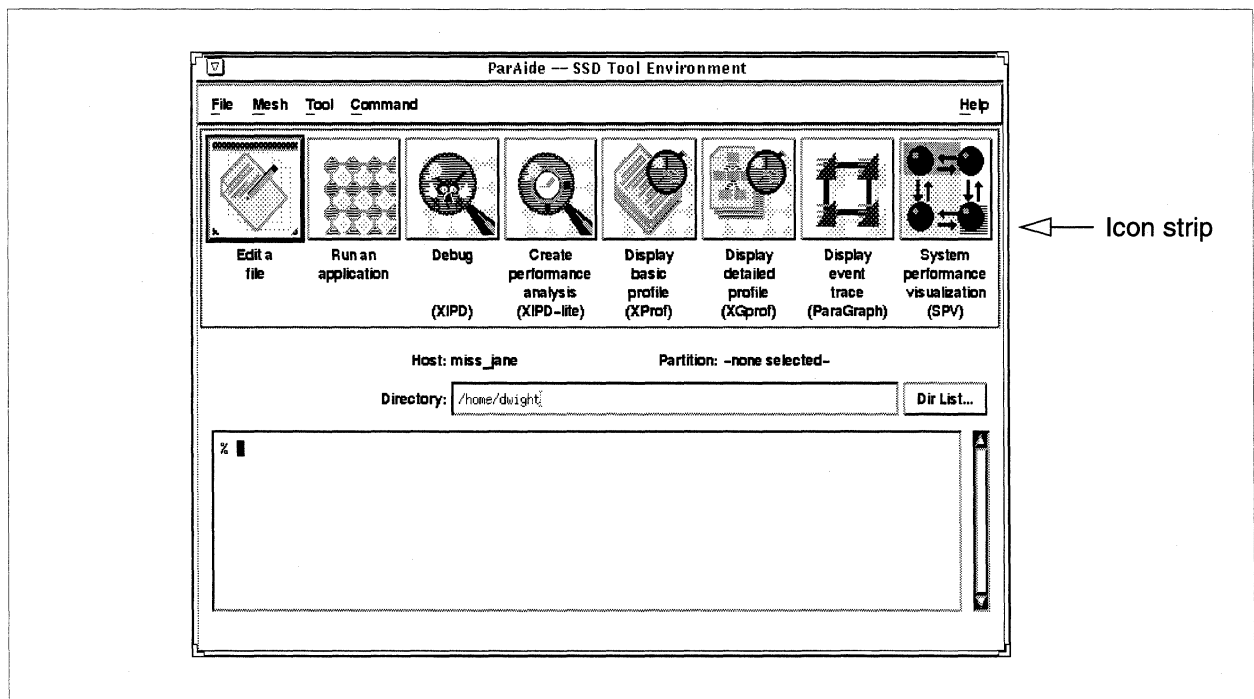


Figure 1-20. Icon Strip

Icons are present for each graphical tool in the Paragon toolset. This includes SPV, XIPD, XIPD-lite, 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 section “Tool Menu” on page 1-18 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. The file selection dialog is described in the section “Selecting Files and Directories” on page 1-34.

When you choose the *Run an application* icon, ParAide displays the *Load Application* dialog.

Terminal Area

The ParAide terminal area is located beneath the icon strip in the main window. Figure 1-21 shows the terminal area.

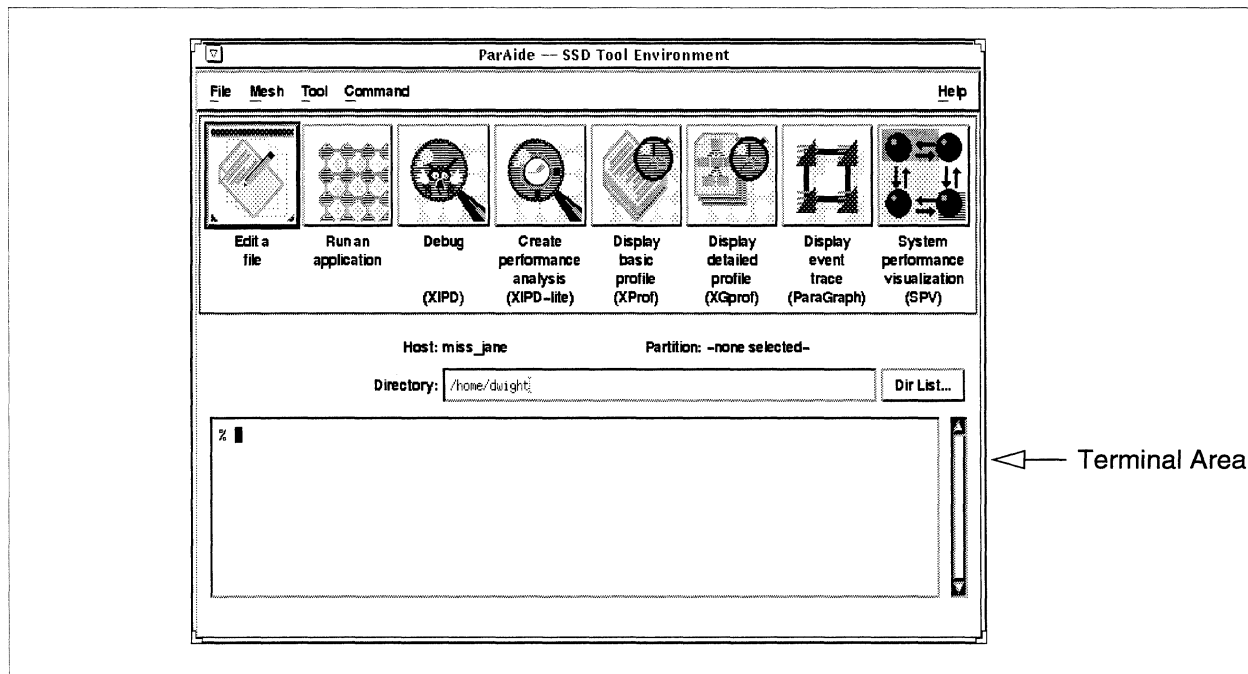


Figure 1-21. Terminal Area

The Terminal behaves like an embedded xterm, providing support for interactive jobs (such as vi) and job control (such as pressing <CTRL-C>). When you load a program, the command is given to the Terminal and the output of the program appears within the Terminal window.

The Terminal area allows you to do the following:

- Display which machine is in use and what partition is selected.
- Display the current directory and allow it to be changed.
- Display the output of loaded programs and allow you to enter shell commands.
- Run interactive jobs and exercise job control.

Host

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

Partition

This label lists the name of the partition you 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 file selection dialog. The file selection dialog is described in the section "Selecting Files and Directories" on page 1-34.

Status Line

The ParAide status line is at the bottom of the ParAide main window. The status line prints messages about work in progress. The status line displays a message for any of the following conditions:

- Any graphical tool is started.
- A Load command is sent to the Terminal.
- An editor window is started.
- A command that does not bring up the command viewer is started via the *Command* menu.

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 1-22 shows the *Select a File to Open* dialog.

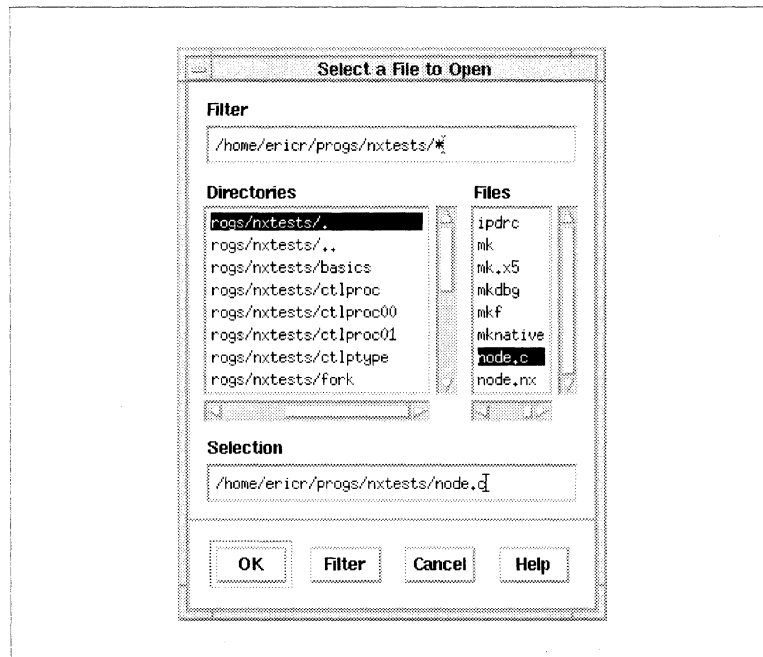


Figure 1-22. Select a File to Open Dialog

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 Application* dialog, pressing *OK* copies the name of the file in the *Selection* field to the *Load Application* dialog's text field as the name of the executable to load.

Dialog Buttons

<i>OK</i>	Passes the entry in <i>Selection</i> to the originating dialog and dismisses the <i>Select a File to Open</i> dialog.
<i>Filter</i>	Obtains a new file list, and possibly a new directory list.
<i>Cancel</i>	Dismisses the <i>Select a File to Open</i> dialog and passes nothing to the originating dialog.
<i>Help</i>	Displays help topic text about the <i>Select a File to Open</i> 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 ParAide application resources listed in the following tables are provided to configure ParAide.

Table 1-1 lists the ParAide display resources.

Table 1-1. Display Resources

Resource	Purpose
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 terminal area for the ParAide main window.
Paraide.showIconLabels	A boolean (True / False) value that controls the display of description labels beneath icons.

Table 1-2 lists the ParAide size resources.

Table 1-2. Size Resources

Resource	Purpose
Paraide.shellRows	An integer that defines the minimum number of rows in the terminal area scrolling text region.
Paraide.shellColumns	An integer that defines the minimum number of columns in the terminal area scrolling text region.
Paraide.nodeSize	An integer that defines the size (in pixels) of the nodes drawn on the mesh.

Table 1-3 lists the ParAide partition resources.

Table 1-3. Partition Resources (1 of 2)

Resource	Purpose
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.

Table 1-3. Partition Resources (2 of 2)

Resource	Purpose
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.
- lineHoriz** Horizontal lines.
- lineVert** Vertical lines.

Default Configuration

Table 1-4 lists the default ParAide resource settings.

Table 1-4. Default ParAide Resource Settings

Resource	Setting
Paraide.showMenus	True
Paraide.showIconStrip	True
Paraide.showIconLabels	True
Paraide.showShell	True
Paraide.shellRows	10
Paraide.shellColumns	80
Paraide.nodeSize	28
Paraide.nonPartitionNodeColor	#acacac (color) #cccccc (grayscale) black (monochrome)
Paraide.partitionNodeColor	#00daa0 (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	<i>Null-String</i> (color/grayscale) black (monochrome)
Paraide.partitionNodeStipple	<i>Null-String</i> (color/grayscale) "dotsDark" (monochrome)
Paraide.partitionSelectedNodeStipple	<i>Null-String</i> (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:

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

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 symbolic, source-level debugger for parallel programs that run under the 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
- C++
- Fortran
- i860™ processor assembly language

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

IPD also allows you to examine core files, using a subset of the command set.

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™ System 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 uncertain tracebacks when you use the **frame** command.
- Mconcur** **-Mconcur** forces **-O2**, so line numbers are generated for each basic block. That decreases the amount of line-number and symbolic information available through the debugger.
- 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 may have unpredictable results.

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

Running IPD

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

ipd

You can cause IPD to execute a set of commands automatically when you start 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™ System 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 most switches 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. 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 most commands 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 run IPD.

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

Table 2-1. Execution Control Commands (1 of 2)

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.
signal	si	Set or display the set of signals IPD reports.
step	ste	Execute the next source statement or instruction.
stop	sto	Stop execution of processes.
trace	tr	Set or display tracepoints.

Table 2-1. Execution Control Commands (2 of 2)

Command	Minimum Abbreviation	Description
wait	wai	Wait until processes stop running; wait for input from an application.
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.
commshow	com	Display or list debug handles for MPI communicators.
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.
msgstyle	msg	Set or display how contexts are formatted (MPI or NX style).
recvqueue	rec	Display posted receives not yet satisfied.
process	p	Display current state of processes.
status	sta	Display current IPD status.
threads	th	Control number of threads displayed for each process.
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.
coreload	cor	Load core files for examination.
quit or exit	q or exi	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.
unset	uns	Delete command line variables.
system or !	sy	Execute a operating system 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 or a core file.

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*; *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

<i>command</i>	One of the IPD commands
<i>arguments</i>	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.
<i>-switch</i>	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. 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.
<i>(context)</i>	The context argument is always defined within parentheses. The context argument defines the set of processes 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. All commands treat the semicolon as a command separator unless preceded by a backslash (\;), in which case the semicolon is passed as part of the argument list.

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 pass a “#” to an argument to a command, precede it with a backslash (“#\”).

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 operating system 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 debug 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 the variable is recognized and accessible. The execution point determines which variables are in scope. It also determines which source file a line number refers to, if not explicitly stated.

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, IPD loads the program and sets the initial default context. 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 change the default context from all nodes to just node 1.

```
(all:0) context (1:0)
(1:0) >
```

The following example is useful when debugging host-node applications. It changes the default context to be that of the host process. In this case, the host process did not define a ptype for itself, so it is omitted from the context.

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

The following example shows using the **context** command to display the list of processes currently loaded, thus providing the options of what could be used to form a valid context.

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

Nodes	Ptype	Program
=====	=====	=====
(0..4)	0	/home/johnd/gauss

IPD gives you several ways to determine the current point of execution, and thus the current scope, for each process: **frame**, **process**, and **list**. While you have access to any point in the program(s) 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

and/or the line number on the command line. If the variable is only in scope on some processes and those are the only processes you want to consider, you can change the context rather than specifying file or routine names. If processes are in a different scope that happen to use the same variable name and the variable name is used in a command without being qualified with a file name or a routine name, you will see the value of that variable appropriate for each different scope of the different processes.

The **frame** command displays the sequence of routines that lead to the current point of execution, with the current routine being at the top. If a routine was compiled with line number debug information, a source line number is displayed, showing the current point of execution and the point of call for the other routines. Consider the following example. 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:

```
(all:0) > disp iam
***** (all:0) *****

** gauss.f{}gauss()#3 **

*** ERROR: cannot parse expression
***      Symbol 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. To display this variable, you would use the command as follows:

```
(all:0) > disp shadow() iam
***** (all:0) *****

** gauss.f{}gauss()#37 iam **
iam = 0
```

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. You use the same syntax for loading your program that you use from the shell, simply place the word **load** at the beginning. The one exception is when using file redirection (<). In this case, the file redirection must be specified immediately after the executable file name, with all other arguments following the redirection file name.

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 load 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 default 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 any other arguments. IPD supports both input and 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 default partition, sets the process type to 1, then loads the file *gauss2* on nodes 1..3, and sets the process type to 2. The default context is set to the first program loaded, *gauss.1*.

```
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 personalized versions of the IPD commands, and debug variables are shorthand versions of strings used in IPD commands. This allows you to create convenient shortcuts to commands you use most commonly. To include a semicolon or a pound sign in aliases, they must be prefixed with a backslash.

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 using the **exec** command. In addition, you can create a special file containing commands that are to be executed whenever you start 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 illustrates how a set of commands can be defined to load and do setup for an application being debugged. Once the command file is executed, you can proceed with entering more debugger commands to analyze the application. The sample command file, *picf*, 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) >
```

On-line 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. If core file analysis is being performed, the summary list of IPD commands shows only those commands that are used with core files.

Controlling Program Execution

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

- Running and halting through programs.
- Setting breakpoints, tracepoints, and watchpoints.
- Defining the signal set that IPD responds to.

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.

A keyboard interrupt (<Ctrl-C>) is required to terminate the **wait** command prior to all processes reaching a stopped state. It is important to be aware that executing processes are allowed to write to *stdout* and *stderr* in only two situations:

- After each command completes, and before each IPD prompt is returned.
- During execution of a **wait** command.

In order to see output from the application while it is executing, you must enter the **wait** command or press <Return> to cause the prompt to be redisplayed. To provide input to an executing application, you must enter the **wait** command and then enter the program input.

Breakpoints, Tracepoints, and Watchpoints

IPD allows you to set and remove breakpoints, tracepoints, and watchpoints. You can set breakpoints and tracepoints at the entry point to a procedure, 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 a process type 0 on node 0 only.

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

The following example sets a watchpoint on write to the address *0x0401b7a8*. The **-write** switch specifies that execution should halt only when this address is written to.

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

Defining the Signal Set

You can use the **signal** command to choose which UNIX signals IPD should report that the application has received. Initially, all signals except for SIGALRM and SIGCHLD are reported. The **signal** command does not change how an application responds to the signals, it only affects whether or not IPD reports receipt of the signal.

The following example uses the **signal** command to add SIGALRM to the set of signals that are reported:

```
(all:0) > signal -on SIGALRM
```

This causes the debugger to keep a process that receives a SIGALRM in a stopped state and report the signal. If this signal had not been added to the list for reporting, the arrival of the SIGALRM would have only awoken the sleeping process and it would have continued without intervention by the debugger.

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.

A *load module* is an executable object module that you have loaded onto the system with the IPD **load** command. The **instrument** command requires the context to include only processes running the same load module.

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

```
ipd > load my_app  
*** reading symbol table for /home/ianboyd/bin/gauss... 100%  
*** loading program...  
*** initializing IPD for parallel application...  
*** load complete  
(all:0) > instrument -prof  
(all:0) > run; wait
```

When the application runs to completion and exits, the prompt returns and the performance data is written out.

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 used in conjunction with the **step** command to display each source line you are stepping through.

```
(0:0) > run;wait
Context          State      Reason    Location    Procedure
=====
*(0:0)          Breakpoint C Bp 6     Line 188    shadow()

(0:0) > step; list,1
Context          State      Reason    Location    Procedure
=====
*(0:0)          Stepped
***** (0:0) *****
File: ./gauss.f
192>      leftnode = iam - 1

(0:0) > step; list
Context          State      Reason    Location    Procedure
=====
*(0:0)          Stepped
***** (0:0) *****
File: ./gauss.f
193>      rightnode = iam + 1
```

The **source** command specifies a list of directories in which to search for a source file. By default, it contains only the current working directory. If your executable consists of source files from multiple directories and/or is not located in the directory with the current source files, you need to use this prior to the **list** command.

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 entry to procedure *shadow()*.

```
(all:0) > disa shadow(),16
***** (all:0) *****
gauss.f{}shadow() + 0x0
00011500: ec1f0401 orh      0x401, r0, r31
00011504: e7ffed00 or       0xed00, r31, r31
00011508: 1fe01801 st.l    fp, 0(r31)
0001150c: a3e30000 mov     r31, fp
00011510: 1fe00805 st.l    r1, 4(r31)
00011514: 1c7f87fd st.l    r16, -4(fp)
00011518: 1c7f8ff9 st.l    r17, -8(fp)
0001151c: 1c7f97f5 st.l    r18, -12(fp)
00011520: 1c7f9ff1 st.l    r19, -16(fp)
00011524: 1c7fa7ed st.l    r20, -20(fp)
00011528: 1c7fafa9 st.l    r21, -24(fp)
0001152c: 1c7fb7e5 st.l    r22, -28(fp)
gauss.f{}shadow()#188
00011530: 147cffe9 ld.l    -24(fp), r28
00011534: 147efffd ld.l    -4(fp), r30
00011538: 139d0001 ld.l    r0(r28), r29
0001153c: 13d00001 ld.l    r0(r30), r16
(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 (in bytes)
(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) > rcvq
*** Unsatisfied receives posted in (all:0)
      Recv Posted   For Msg           Msg Length
Call Type   By           From           Msg Type      (in bytes)   Handler
=====
CRECV      (0:0)         (2:0)         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. These commands are used together to find message-passing errors, such as messages sent to the wrong nodes, or receives posted for the wrong type.

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

```
(0:0) > display leftid
***** (0:0) *****

** gauss.f{}shadow()#193 leftid **
leftid = 0
```

The following example shows the display of the register named "r1". The value in parentheses off to the right is the decimal equivalent to the hexadecimal value on the left (identified by the leading "0x").

```
all:0) > disp -r1
***** (all:0) *****
r1      0x000109c0          ( 68032 )
```

If a memory address is specified on the display command line, a hexadecimal dump of 352 bytes of memory is displayed beginning with the address specified. The ASCII conversion of the values in memory is displayed on the right. A dot (.) represents a non-printable character.

```
(all:0) > disp 0x04018bd8
***** (all:0) *****

** gauss.f{}shadow()#193 67210200 **

0x04018bd8: 0x00000041 0x6c6c6548 0x726f576f 0x0000646c *A...HelloWorld...*
0x04018be8: 0x00000000 0x00000000 0x20202020 0x20202020 *.....*
```

```

0x04018bf8: 0x20202020 0x20202020 0x20202020 0x20202020 * *
0x04018c08: 0x20202020 0x20202020 0x20202020 0x20202020 * *
0x04018c18: 0x20202020 0x20202020 0x20202020 0x20202020 * *
0x04018c28: 0x20202020 0x20202020 0x20202020 0x20202020 * *
0x04018c38: 0x20202020 0x20202020 0x20202020 0x20202020 * *
0x04018c48: 0x20202020 0x20202020 0x20202020 0x20202020 * *
0x04018c58: 0x20202020 0x20202020 0x20202020 0x20202020 * *
0x04018c68: 0x20202020 0x20202020 0x20202020 0x20202020 * *
0x04018c78: 0x20202020 0x20202020 0x20202020 0x20202020 * *
0x04018c88: 0x20202020 0x20202020 0x20202020 0x20202020 * *
0x04018c98: 0x20202020 0x20202020 0x20202020 0x20202020 * *
0x04018ca8: 0x20202020 0x20202020 0x20202020 0x20202020 * *
0x04018cb8: 0x00000000 0x00000000 0x00000000 0x00000001 * ..... *
0x04018cc8: 0x00000002 0x00000003 0x00000004 0x00000005 * ..... *
0x04018cd8: 0x00000006 0x00000007 0x00000008 0x00000009 * ..... *
0x04018ce8: 0x0000000a 0x0000000b 0x0000000c 0x0000000d * ..... *
0x04018cf8: 0x0000000e 0x0000000f 0x00000010 0x00000011 * ..... *
0x04018d08: 0x00000012 0x00000013 0x00000014 0x00000015 * ..... *
0x04018d18: 0x00000016 0x00000017 0x00000018 0x00000019 * ..... *
0x04018d28: 0x0000001a 0x0000001b 0x0000001c 0x0000001d * ..... *

```

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* for just one process (3:0).

```

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

***** (0..2:0) *****

** gauss.f{}shadow()#193 nbrnodes **
nbrnodes = 1

***** (3:0) *****

** gauss.f{}shadow()#193 nbrnodes **
nbrnodes = 3

```

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

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

***** (3:0) *****

** gauss.f{}shadow()#188 iam **
iam = 2
```

Debugging Threaded Applications

IPD allows viewing of information from multiple threads in a process. To do this, use the **threads** command with the **-on** switch. When threads mode is on, commands like **display**, **frame**, and **process** print information for each thread in each process in the current context. Following is an example of displaying a stack traceback before and after turning threads mode on:

```
(all:0) > frame
***** (all:0) *****
func1() [myhello.c{}#8]
main() [myhello.c{}#31]
_crt0_start() [crt0.c{}0x000101fc]
(all:0) > threads -on
(all:0) > frame
== (all:0) =====
***** (all:0) thread 0 *****
func1() [myhello.c{}#8]
main() [myhello.c{}#31]
_crt0_start() [crt0.c{}0x000101fc]
***** (all:0) thread 1 *****
_mach_msg_trap() [unknown1.s{}0x00024ae8]
? mach_msg() [mach_msg.c{}0x0002487c]
mach_msg_receive() [mach_msg_receive.c{}0x000249f0]
_nx_port_rcv_thread() [nx_port.c{}0x00023fd4]
***** (all:0) thread 2 *****
_mach_msg_trap() [unknown1.s{}0x00024ae8]
? mach_msg() [mach_msg.c{}0x0002487c]
mach_msg_receive() [mach_msg_receive.c{}0x000249f0]
_nx_port_rcv_thread() [nx_port.c{}0x00023fd4]
(all:0) >
```

IPD does not allow execution control of individual threads. Thus, commands like **run**, **step**, and **break** cannot be applied to an individual thread. If a breakpoint is set and execution started, all threads for the process execute until one thread hits the breakpoint, causing all threads to halt. Upon continuing execution, the breakpoint may be hit again as a different thread encounters it again, halting all of the other threads.

When threads mode is off (the default) IPD displays information on the thread that was active at the time the process was halted. Generally, this is the main user thread (thread ID 0), but if a different thread caused the process to halt a ">" is printed in the first column of that process's **process** command output.

The thread ID number seen in the command's output is not bound to a specific thread. Each time execution is resumed a thread's ID number may change. Thread 0 is the only exception to this; it is created first and remains constant throughout the execution of the application. The **frame** command is useful for determining the origin of a thread.

Applications compiled with the **-nx** option have several threads generated for it automatically. The first is used by the handler specified in one of the *hrecv()* family of calls. The second is used for virtual memory paging. Since both these threads are created automatically and live throughout the execution of a process, their thread IDs (thread 1 and 2, respectively) remain constant.

If a call is made to a PFS function, 1 or 2 threads are generated at that time for use by the parallel file system's I/O handling.

The compiler switch **-Mconcur** also causes 1 or 2 threads to be created. It also has the side effect of forcing an **-O2** optimization level, which reduces the amount of debug information available.

Debugging MPI Applications

An application that successfully executes a call to *MPI_Init()* is recognized as being an MPI application by IPD. This causes IPD to change the manner in which a process is named in context displays, from the NX node/ptype pair to the MPI communicator/rank pair. Once *MPI_Finalize()* is executed the context displays revert to the NX style. The **msgstyle** command with the **-nx** switch can be used to force context displays to use the NX style prior to encountering the *MPI_Finalize()* function. Use the **-mpi** switch to return to MPI mode.

Communicators are assigned handles (names) as they are created within the application. The initial set of communicator handles consists of COMMWORLD and COMMSELF0, ..., COMMSELF $n-1$ where n is the group size of COMMWORLD. These are handles for MPI_COMM_WORLD and MPI_COMM_SELF (one per node), respectively. As additional communicators are created within the application, intracommunicators are assigned handles of the form COMM1, COMM2, etc. and intercommunicators are assigned handles of the form ICOMM1, ICOMM2, etc. These handles persist until the corresponding communicator is freed by the application.

The **commshow** command without arguments lists all of the communicator handles available for use in a context specification. To find the communicator handle for a particular communicator in the application, specify the variable name of the desired communicator as an argument to the **commshow** command. Below is an example of an MPI application that was run on a six-node partition and stopped after it created a communicator. IPD assigned the communicator the handle COMM1 and commshow indicates that it has five processes in its group composed of the MPI_COMM_WORLD processes with ranks 1 through 5.

```
(COMMWORLD:all) > commshow
Intracommunicators:
Name          Size   Rank (in COMMWORLD)
=====
COMMWORLD     6     0..5
COMM1         5     1..5

COMMSELF[0..5]

Intercommunicators:
Name          Intracommunicator Pair
=====
```

The next example shows that the communicator handle for the variable “other” (of type `MPI_Comm`) is `COMM1` for `COMMWORLD` ranks 1 through 5. Note that an error was returned for the process in `COMMWORLD` with rank 0, because the communicator was not defined for that process.

```
(COMMWORLD:all) > commshow other
***** (COMMWORLD:0) *****

** commany.c{ }main()#68 other **

ERROR: cannot get communicator information
***   Null pointer argument

***** (COMMWORLD:1..5) *****

** commany.c{ }main()#68 other **
other = COMM1
```

Communicator handles and ranks can be used in context specifications to restrict the set of processes that commands apply to just like `node/ptypes` do. Thus, if you only wanted to see message passing that was occurring within the communicator “other” in your program, you would find the handle associated with that variable as shown above and set the default context accordingly. Then the commands `msgqueue` and `recvqueue` will only report on messages sent or receives posted within the context of that communicator. To view all messages without regard for the context in which they were sent or posted you must add the `-all` switch to these commands. The `-all` switch causes any messages sent via `NX` message passing calls to be included in the display as well as all `MPI` messages.

```
(COMMWorld:all) > context(comm1:all)
(COMM1:all) > msgq
*** Unreceived messages in (COMM1:all)
```

Source	Destination	Msg Tag	Msg Length (in bytes)
(COMM1:0)	(COMM1:0)	23	10
(COMM1:0)	(COMM1:1)	23	10
(COMM1:0)	(COMM1:2)	23	10
(COMM1:0)	(COMM1:3)	23	10
(COMM1:0)	(COMM1:4)	23	10

```
(COMM1:all) > recvq
```

```
*** Unsatisfied receives posted in (COMM1:all)
```

Recv Posted By	For Msg From	Msg Tag	Msg Length (in bytes)
(COMM1:0)	(COMM1:1)	ANY	12
(COMM1:1)	(COMM1:ANY)	22	12
(COMM1:1)	(COMM1:1)	ANY	12
(COMM1:2)	(COMM1:ANY)	22	12
(COMM1:2)	(COMM1:1)	ANY	12
(COMM1:3)	(COMM1:ANY)	22	12
(COMM1:3)	(COMM1:1)	ANY	12
(COMM1:4)	(COMM1:ANY)	22	12
(COMM1:4)	(COMM1:1)	ANY	12

As seen in the preceding example, the **msgqueue** and **recqueue** commands output is slightly different for MPI messages. The Call Type column does not appear, because this information is not available to the debugger. There is a Msg Tag column rather than a Msg Type and this contains the word ANY if the MPI receive used the tag wildcard MPI_ANY_TAG. The "For Msg From" column contains the word ANY if the rank wildcard MPI_ANY_SOURCE was specified in the receive call.

If the communicator used in sending a message or posting a receive no longer exists, COMMUNKNOWN will be given as the communicator handle.

Examining Core Files

The **coreload** command loads one or more core files for examination. After a core file is loaded, all IPD commands that are not related to executing code are available for viewing the state of the process and the contents of the data. The **help** command will show only the subset of commands that applies to core files is entered after a **coreload** command.

IPD can provide symbolic information on where the error occurred and the calling sequence which brought you to that location, using a stack (or frame) traceback. IPD can also display register values. If the application was compiled for debug when it faulted, line number and variable data can be displayed as well. The contents of variables can only be displayed if the type of a core file is FULL.

Following is an example session using IPD to display information contained in a core directory. Only faulting processes are loaded by default.

```
ipd > coreload
*** reading symbol table for my_app... 100%
*** scanning core files...
*** coreload complete
      Context                State      Reason      Location      Procedure
=====
*(0,1:0)                Signaled   SIGSEV      Line 15      sub1()
(0,1:0) > frame
***** (all:0) *****
    sub1()    [myapp.c{}#15]
    main()    [myapp.c{}#26]
    _crt0_start() [crt0.c{}0x000101e0]
(0,1:0) > list #10
***** (all:0) *****
File: ./myapp.c
10:
11>void
12:sub1( ptr )
13:long *ptr;
14:{
15>  *ptr = 5;
16;}
(0,1:0) > disp ptr
***** (all:0) *****

** myapp.c{}sub1()#15 ptr **
ptr = 0x00000000

(0,1:0) >
```

The example shows that a null pointer was being used in an assignment at line #15 in function *subI()*. This example shows output for an application which was compiled for debug. If it had not been, the line number information would have been replaced by addresses and display of the symbolic name "ptr" would not have been possible. Thus, you could only narrow the problem down to the routine in which it occurred, unless you were familiar with assembly language and used disassemble and **display -registers** to get further clues to the problem.

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 );
```

Breakpoints and tracepoints must be set on the *first* line of a multi-line C++ call.

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.'
```

The **list** command displays a ">" character in front of lines that may be specified for a breakpoint.

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	INTEGER*1 var <---
logical*1 var(x)	INTEGER*1 var(x) <---
logical*1 var(x,y)	INTEGER*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. Note that processes may be left running if you use this method to interrupt IPD.

Using IPD in a Sample Session

This section shows how to use IPD in an example debug session. The program example is a variation of the Gauss-Seidel method for solving LaPlace's equation. The example uses an iterative algorithm in which each cycle requires that each array element is averaged with its nearest neighbors. The example decomposes the problem by columns. It distributes blocks of columns to each processor used in the computation.

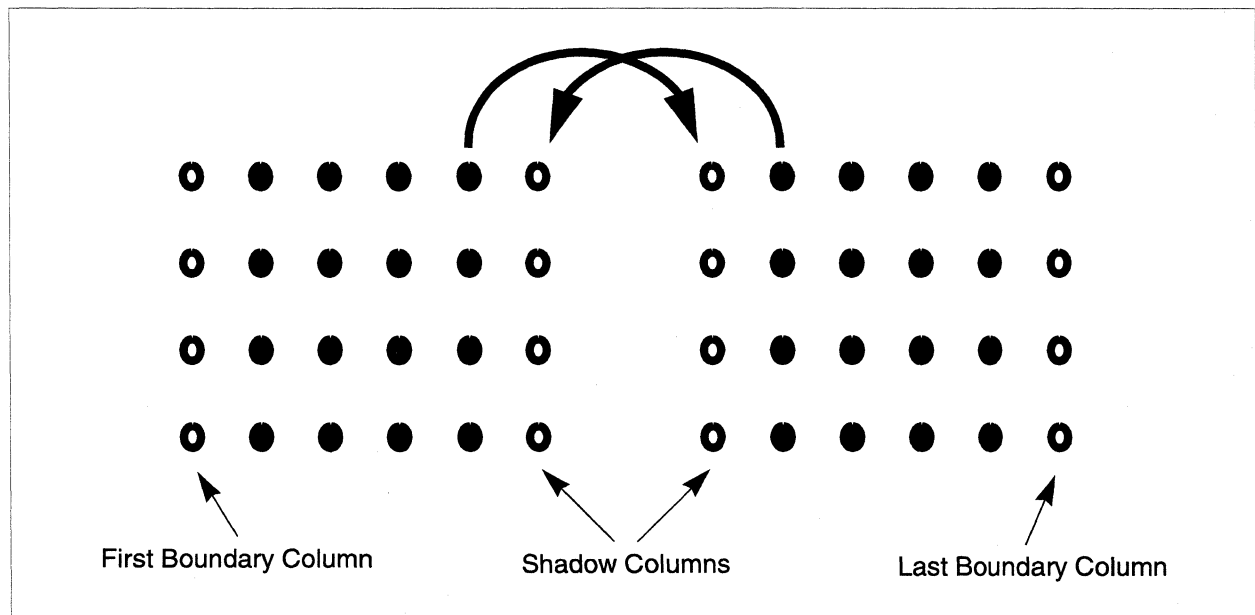


Figure 2-1. Shadow Columns in the Gauss-Seidel Example

Figure 2-1 shows that each node owns a block of columns. Only the outer column of each of these blocks must be shared with another node. To reduce communication between the nodes, each processor maintains a “shadow” buffer that contains a copy of the neighboring column from the previous iteration.

Creating the Example Program

Copy the contents of `/usr/share/examples/ipd` to your own area. Compile and link the example by running `make` or with the following command:

```
if77 -o gauss -g gauss.f -nx
```

This command creates the executable file `gauss` to run on multiple nodes. It is compiled with the `-g` switch to create debug information.

Editing the `.ipdrc` File

Before starting IPD, you can create and edit the `.ipdrc` file to set up several alias definitions for IPD commands. Create a file called `.ipdrc` in your home directory, containing the following commands:

```
alias c continue\; wait  
alias s step
```

This example defines aliases for the commonly used command pair `continue` and `wait`, and for the `step` command. When you start IPD, it reads the `.ipdrc` file and stores the aliases you specify.

Starting IPD

Type the following command to start IPD. The example assumes that you run IPD from the same directory that contains the example's source and executable files.

```
% ipd  
  
*** IPD Parallel Debugger, Paragon Release 1.4  
*** Copyright (c) 1990,1991,1992,1993,1994,1995 Intel Corporation  
  
ipd >
```

Loading the Program

Load the program for debugging. It is loaded onto four nodes in the default partition, as specified by the **-sz** switch argument.

```
ipd > load gauss -sz 4
*** reading symbol table for /home/myacct/gauss... 100%
*** loading program...
*** initializing IPD for parallel application...
*** load complete
(all:0) >
```

After loading the program, the prompt changes to the default context (**all:0**). The context defines the nodes and processes affected by the command.

You can change the default context with the **context** command. Specifying the context in a command itself overrides the default context, for that command only.

Getting Help

You can get a list of IPD commands by entering the **help** command. Use the abbreviation **h** or enter a question mark (?), as follows.

```
(all:0) > help
Commands are grouped functionally. The information provided for each command is: command name, shortest acceptable abbreviation, and brief description.
```

Enter 'help <command>' to get detailed information for each command.

Program load and termination:

load	loa	Load programs
coreload	cor	Load core files for examination
kill	k	Terminate processes

Program execution control and state:

continue	conti	Continue stopped processes
run	ru	Start process execution from the beginning
rerun	rer	Same as run, but do not reuse previous argument list
process	p	Display the current state of processes
frame	fr	Display stack traceback from current execution point
stop	sto	Halt process execution
wait	wai	Wait for processes to stop
break	b	Set or display breakpoints
trace	tr	Set or display tracepoints
watch	wat	Set or display watchpoints
remove	rem	Remove break/trace/watchpoints

step	ste	Execute the next source statement or instruction
signal	si	Modify or display IPD signal reporting

Program performance analysis data collection:

instrument	i	Instrument program for collecting performance data
flush	fl	Set flush policy for event trace buffers

Program data display and modification:

assign	as	Assign a new value to a variable or address
display	disp	Display the value of an expression, address, or register
type	ty	Display the type of an expression

Program message queue display:

msgqueue	ms	Display the queue of messages sent but not received
recvqueue	rec	Display the queue of receives posted but not satisfied

Program code display:

list	li	List source code
disassemble	disa	Display an assembly listing of program code
source	so	Set or display source directory search paths

IPD control and information:

context	conte	Change default list of processes to apply commands to
commshow	com	Display communicator handles assigned by debugger
exec	exe	Read debugger commands from a file
exit	exi	Exit IPD - same as quit
help or ?	h	Display command summary or details
log	log	Record the debug session in a file
more	mo	Turn terminal scrolling on or off
msgstyle	msgs	Set or display context format
alias	al	Set or display command aliases
unalias	una	Delete aliases
set	se	Set or display debug variables
unset	uns	Delete debug variables
status	sta	Display version number and control values
system or !	sy	Execute a UNIX shell command
threads	th	Set or display threads mode
quit	q	Exit IPD - same as exit

For more specific help on a given command, enter **help** or **?**, followed by the name of the command on which you wish help. This displays a brief description of the function, followed by the syntax. For example, you can request help on **break** as follows:

```
(all:0) > ? break
```

```
Display Breakpoint information:
```

```
break [context] [-full]
```

```
Set Breakpoint at procedure:
```

```
break [context] [file{}]procedure([type[,type]...]) [-after count]
```

```
Set Breakpoint at source line number:
```

```
break [context] #line [-after count]
```

```
break [context] file{}#line [-after count]
```

```
break [context] [file{}]procedure([type[,type]...])#line [-after count]
```

```
Set Breakpoint at instruction address:
```

```
break [context] address [-after count]
```

The **break** command allows you to set code breakpoints or display current breakpoints. The count value specifies the number of times the breakpoint is encountered before the break occurs (default is 1). Breakpoints are defined in the current context (either default or specified).

Without any switches, this command displays the current breakpoints for the default or specified context. The "-full" switch generates a "long form" display with more room for long file and function names.

This command is not allowed when examining core files.

Setting Up the Debugging Environment

It may be useful to have a log file containing a record of your debugging session. Use the **IPD log** command to specify this. In the following command, you turn on logging and name the file *gauss.log*.

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

Now list IPD's current status:


```
(all:0) > status
```

```
IPD version number: Paragon Release 1.4
```

```
Debug mode: Runtime process analysis
```

```
Application partition info: .compute.mypartition
```

USER	GROUP	ACCESS	SIZE	FREE	RQ	EPL
myacct	usr	777	4	0	SPS	0

```
Message style: NX
```

```
More: on
```

```
Threads: off
```

```
Log file: gauss.log
```

```
Source search paths for /home/myacct/gauss:
```

The **status** command returns the debug mode, partition information, the status of the **more** command, the name (if any) of the current log file, and the source search paths. The debug mode indicates whether you are examining a running executable or a core file. The IPD **more** facility controls screen scrolling; when you are listing something that fills up more than one screen it pauses when the screen is full and waits for you to enter a keystroke. During interactive debugging, **more** is set to "on" by default.

Unless all of your source files are in your current directory, you need to add to the source search paths by using the **source** command so IPD can find the source files.

To display the aliases that are set, enter the **alias** command.

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

Alias	Command String
=====	=====
c	continue; wait
s	step

The command displays the aliases you set in the *ipdrc* file. For example, using the single keystroke **c** executes both the **continue** and **wait** commands.

Another way to set up your debug environment is to use the **set** command to define a variable representing commands that you enter numerous times. For example, you could define the string **ALL** to represent the context string (*all:0*), and another string **S** to represent the name of a procedure you use many times.

```
(all:0) > set ALL (all:0)
(all:0) > set S shadow()
(all:0) > set
Variable      Substitution String
=====
ALL           (all:0)
S             shadow()
```

Once you have defined these variables, you can use them on command lines in place of the original string; a dollar sign (\$) is required in front of the variable name. For example, you could use the **display** command to display the variable *iam* on all nodes. Attempting to do this display now will fail, because the variable *iam* is local to *shadow()* and does not exist yet. If the application were stopped within a call to *shadow()*, the output would be as follows:

```
(all:0) > disp $S iam

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

Running the Program

Enter the **run** command, followed by the **wait** command, to run the program until all processes are complete, or some other event occurs (such as a breakpoint).

```
(all:0) > run; wait
```

After a few seconds, you see no sign that the program is going to return, so you correctly conclude that it is hung up somewhere. Press **** or **<Ctrl-C>** to cause control to return to the debugger. This allows you access to the IPD commands, but does not halt execution, as you can see by entering the **process** command:

```
^C
(all:0) > process
      Context                State      Reason      Location      Procedure
=====
*(all:0)                Executing
```

The asterisk in front of the context indicates that the status of the program has changed, since the last time you displayed the process state.

Tracking the Fault

To begin to analyze the problem, halt execution with the **stop** command. You cannot use commands to examine the state of a process unless it is in a stopped state.

```
(all:0) > stop
```

If you enter the **process** command now, more information is available. (Since no other commands begin with p, you can abbreviate the **process** command.)

```
(all:0) > p
Context                State      Reason    Location    Procedure
=====
* (0:0)                Interrupted 0x00029d44 _flick()
* (1:0)                Interrupted 0x000295e4 _flick()
* (2:0)                Interrupted 0x00029d44 _flick()
* (3:0)                Interrupted 0x00029598 _flick()
* (4:0)                Interrupted 0x00020f18 _csend()
```

The asterisk in front of the context display indicates that the process status has changed again. The display also tells you that three processes are stopped in the *flick()* routine and one process is in the *csend()* routine. This is an indication that the program is blocked somehow. To find out where the problem is, use the **frame** command to trace the stack to provide a history of the routines called, starting with the most recent.

```
(all:0) > frame
***** (0:0) *****
_flick()    [_flick.c{}0x00028598]
_crecv()    [nxlib.c{}0x0002111c]
crecv()     [crecv.c{}0x0002e028]
gopfb_tree() [_gops.c{}0x0002ab4c]
_gopfb()    [_gops.c{}0x00029918]
_gdhigh()   [_gops.c{}0x000287f8]
gdhigh_()   [gdhigh_.c{}0x0001c38c]
gauss()     [gauss.f{}#87]
main()      [pgfmain.c{}0x000109bc]
_crt0_start() [crt0.c{}0x000101e0]
***** (1:0) *****
_flick()    [_flick.c{}0x000285e4]
_crecv()    [nxlib.c{}0x0002111c]
crecv()     [crecv.c{}0x0002e028]
gopfb_tree() [_gops.c{}0x0002aa34]
_gopfb()    [_gops.c{}0x00029918]
_gdhigh()   [_gops.c{}0x000287f8]
gdhigh_()   [gdhigh_.c{}0x0001c38c]
```

```

gauss()      [gauss.f{}#87]
main()      [pgfmain.c{}0x000109bc]
_crt0_start() [crt0.c{}0x000101e0]
**** (2:0) ****
_flick()    [_flick.c{}0x00028574]
_crecv()    [nxlib.c{}0x0002111c]
crecv()     [crecv.c{}0x0002e028]
gopfb_tree() [_gops.c{}0x0002ad70]
_gopfb()    [_gops.c{}0x00029918]
_gdhigh()   [_gops.c{}0x000287f8]
gdhigh_()   [gdhigh_.c{}0x0001c38c]
gauss()     [gauss.f{}#87]
main()     [pgfmain.c{}0x000109bc]
_crt0_start() [crt0.c{}0x000101e0]
**** (3:0) ****
_csend()    [nxlib.c{}0x00020f18]
csend_()    [csend_.c{}0x0001c30c]
shadow()    [gauss.f{}#222]
gauss()     [gauss.f{}#67]
main()     [pgfmain.c{}0x000109bc]
_crt0_start() [crt0.c{}0x000101e0]

```

The names followed by the double parentheses () indicate the routines that the program has called. The names followed by the double braces {} indicate the source modules in which these routines reside. The routines that show line numbers were compiled for debug and generally are user routines. Those with memory addresses were not compiled for debug and are typically system library routines.

The stack trace tells you that nodes 0 through 2 went to the system routine *_flick()* from the user-called routine *gdhigh()*, which was called at line 87 in *gauss.f*. All three nodes have similar trace records, with only the location of the calls to some routines and the point at which they stopped inside *_flick()* differing among them. Node 3, on the other hand, is in the *_csend()* routine, which it reached by way of a call to the *shadow()* routine, called at line 67 in *gauss.f*. Knowing that *gdhigh()* is a global operation library call, which involves all nodes, we can guess that node 3 is the hold-up. Since it is not doing what the other nodes are, we can focus our attention on node 3 for now.

The **msgqueue** and **recvqueue** commands allow you to look at the message send and receive queues for misdirected messages.

```
(all:0) > msgq
```

```
*** Unreceived messages in (all:0)
```

Source	Destination	Msg Type	Msg Length (in bytes)
=====	=====	=====	=====

```
(all:0) > recvq
```

```
*** Unsatisfied receives posted in (0:0)
```

Call Type	Recv Posted By	For Msg From	Msg Type	Msg Length (in bytes)	Handler
=====	=====	=====	=====	=====	=====
CRECV	(0:0)	(-1:-1)	1000004002	8	
CRECV	(1:0)	(-1:-1)	1000004004	8	
CRECV	(2:0)	(-1:-1)	1000004800	8	
IRECV	(3:0)	(-1:-1)	300	144	

This tells you that there are no messages waiting to be received on any of the nodes, as indicated by the empty **msgq** display. However, all of the nodes have receives posted. Nodes 0 through 2 have posted blocking receives (*crecv*). Node 3 posted an asynchronous receive (*irecv*) for a message of type 300, which has not come in, and proceeded on. The frame output tells us that it is blocked in the *csend()* routine, called at line 222.

The **list** command lists the source code from the current execution point, a procedure, or a source line number. List the whole *shadow* routine, which contains line 222:

```
(all:0) > list shadow()
```

```
***** (0:0) *****
```

```
File: ./gauss.f
```

```
180>      subroutine shadow(ndim,n,totdim, iam,nbrnodes,range,a)
181:c
182:c      Shadow performs the exchange of neighbor columns into the shadow buffers
183:c
184:      integer          fromleft, fromright
185:      parameter (fromleft= 200, fromright = 300)
186:      integer          ndim,n,totdim
187:      integer          iam, nbrnodes, range
188>      double precision a(ndim,range+2)
189:
190:      integer          length,leftnode,rightnode,lefttid,righttid
191:
192>      leftnode  = iam - 1
193>      rightnode = iam + 1
194>      length    = (totdim+2)*8
195:
196>      if(iam.eq.0) then
197:c
198:c      If I am the leftmost node of the array (node 0) then only exchange
199:c      with the right (to the left is a boundary of the array)
200:c
```

```

201>         rightid = irecv(fromright, a(1,range+2), length)
202>         call csend(fromleft, a(1,range+1), length, rightnode,0)
203>         call msgwait(rightid)
204:
205>         else if (iam .eq. nbrnodes) then
206:c
207:c   If I am the rightmost node of the array (highest numbered node) then
208:c   only exchange with the node to the left.
209:c
210>         leftid = irecv(fromleft, a(1,1), length)
211>         call csend(fromright, a(1,2), length, leftnode,0)
212>         call msgwait(leftid)
213:
214>         else
215:c
216:c   Otherwise I am a node in the middle, so exchange with nodes to either
217:c   side.
218>         leftid = irecv(fromleft, a(1,1), length)
219>         rightid = irecv(fromright,a(1,range+2), length)
220:
221>         call csend(fromright, a(1,2), length, leftnode,0)
222>         call csend(fromleft, a(1,range+1), length, rightnode,0)
223:
224>         call msgwait(leftid)
225>         call msgwait(rightid)
226:
227>     endif
228> end

```

This routine deals with the “shadow” columns that each node must share with the adjacent nodes, as described earlier. The variable *iam* is the current node number. If *iam* is node 0, it sends and receives only to the right (higher-numbered nodes) because there are no nodes to the left. If the node is the highest-numbered node (the right-most), it communicates only to the left. All other nodes receive from and send to nodes on either side, both left and right. This application was run on a 4-node partition, numbered 0..3, so node 3 is the highest numbered node. It should be executing the middle block of the *if* statement, but the **frame** command indicated that node 3 was halted at line #222, in the last block. It is blocked in *csend()*, which tries to send a message to a node on its right, which doesn't exist.

You probably want to examine the variables to see what happened. With the **display** command, you can display the value of any variable or memory address. Look at the variables *iam* and *nbrnodes* to find out why the middle block is not being executed. Set the context with the **context** command so that you are only looking at node 3. Notice that in both these **display** commands (which use the **disp** abbreviation) the variable name is qualified with the name of the routine in which they reside. That is necessary here because node 3 is not in actually in the *shadow()* routine, and there could be different local variables of the same name in the routine it is in. We can avoid confusion by being explicit concerning the scope of the variables we are interested in.

```
(all:0) > context (3:0)
(3:0) > disp shadow()iam

***** (3:0) *****

** gauss.f{}shadow()#188 iam **
iam = 0

(3:0) > disp shadow()nbrnodes

(3:0) > disp shadow()nbrnodes
***** (3:0) *****

** gauss.f{}shadow()#188 nbrnodes **
nbrnodes = 4
```

The **display** output shows that *iam* is not equal to *nbrnodes*. You begin to suspect that *nbrnodes* should, in fact, be *nbrnodes-1*, since you want to check for the maximum node id number, which is always one less than the total number of nodes.

IPD also gives you the **type** command, so you can quickly see the type of any variable. (Here, we make use of the shorthand provided by the variable we created earlier. We could have used it in the previous two **display** commands in place of “shadow()” as well.)

```
(3:0) > type $S nbrnodes
** gauss.f{}shadow()#188 nbrnodes **
***** (0:0) *****
INTEGER
```

To prove your suspicion about the cause of the fault in the program, you decide to use IPD's ability to set breakpoints and assign values to variables. The **break** command allows you to display current breakpoints or set breakpoints at procedures, source line numbers and instruction addresses. In this case, you reset the context to include all processes and set a breakpoint at line #192, the first executable line of the shadow procedure. Display the breakpoint using **b**, an abbreviation for **break**.

```
(3:0) > context $ALL
(all:0) > break shadow()#192
(all:0) > b
(all:0)
Bp #   File name      Procedure   Breakpoint Condition      Bp context
====  =====
1    gauss.f          shadow     Line 192                    (all:0)
```

Now enter **run** followed by **wait**. The **run** command starts the program from the beginning. Since this results in the killing of the current set of processes, you are asked to confirm that you want to do this. The **wait** command prevents the prompt from returning until the breakpoint is hit and process information is displayed.

```
(all:0) > run
* This command will destroy all processes under debug.
  Are you sure you want to do this (y/n)? y
*** initializing IPD for parallel application...
```

```
(all:0) > wait
Context                State      Reason    Location    Procedure
=====
*(all:0)                Breakpoint C Bp 1     Line 192    shadow()
```

Now, once again, display the current value of *nbrnodes*, and then, on node 3, use the **assign** command to temporarily reassign the value of *nbrnodes* to 3, instead of 4. Qualification of the scope of the variables is unnecessary here, since the application is actually stopped within the *shadow()* routine itself.

```
(all:0) > disp nbrnodes
***** (all:0) *****

** gauss.f{}shadow()#192 nbrnodes **
nbrnodes = 4
```

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

** gauss.f{}shadow()#192 nbrnodes **
nbrnodes = 3
```

Using the **step** command, you can single step through the next several steps. The following example uses the **s** alias:

```
(3:0) > s
Context                State      Reason    Location    Procedure
=====
*(3:0)                Stepped                    Line 193    shadow()
(3:0) > s
Context                State      Reason    Location    Procedure
=====
*(3:0)                Stepped                    Line 194    shadow()
```

Now, for the next two steps, if you add the **list** command with a count of 1, (**list,1**), you can display the lines you are stepping through. Note that the **list** command's count value needs only to be specified once. IPD retains the count previously used.


```
(3:0) > s ; list,1
Context          State      Reason      Location      Procedure
=====
*(3:0)          Stepped          Line 196     shadow()
***** (3:0) *****
File: ./gauss.f
196>      if(iam.eq.0) then
```

```
(3:0) > s ; list
Context          State      Reason      Location      Procedure
=====
*(3:0)          Stepped          Line 201     shadow()
***** (3:0) *****
File: ./gauss.f
201>      rightid = irecv(fromright, a(1,range+2), length)
```

The program is now executing the correct block of the **if** statement. You can remove the breakpoint with the **remove** command, specifying the breakpoint number displayed by the previous break command. However, you should first reset the context to all processes, since the breakpoint was set for all processes. If you did not, only the breakpoint on node 3 would be removed and all other processes would encounter it again and stop. You may then continue executing the program with the **continue** command. The **continue** command, unlike the **run** command, continues from the current point, and retains the reassigned values of any variables.

```
(3:0) > context $ALL
(all:0) > rem 1
(all:0) > c
```

```
C 0 99. 99. 99. 99. 99. 99. 99. 99. 99. 99. 99. 99. 99. 99. 99. 99. 99.
C 1  0. 49. 69. 78. 83. 86. 88. 89. 89. 89. 89. 88. 86. 83. 78. 69. 49.  0.
C 2  0. 30. 49. 62. 69. 74. 77. 79. 80. 80. 79. 77. 74. 69. 62. 49. 30.  0.
C 3  0. 21. 37. 49. 58. 64. 68. 70. 72. 72. 70. 68. 64. 58. 49. 37. 21.  0.
C 4  0. 16. 30. 41. 49. 56. 60. 63. 64. 64. 63. 60. 56. 49. 41. 30. 16.  0.
C 5  0. 13. 25. 35. 43. 49. 54. 57. 58. 58. 57. 54. 49. 43. 35. 25. 13.  0.
C 6  0. 11. 22. 31. 39. 45. 49. 52. 54. 54. 52. 49. 45. 39. 31. 22. 11.  0.
C 7  0. 10. 20. 29. 36. 42. 47. 49. 51. 51. 49. 47. 42. 36. 29. 20. 10.  0.
C 8  0. 10. 19. 27. 35. 41. 45. 48. 49. 49. 48. 45. 41. 35. 27. 19. 10.  0.
C 9  0. 10. 19. 27. 35. 41. 45. 48. 49. 49. 48. 45. 41. 35. 27. 19. 10.  0.
C10 0. 10. 20. 29. 36. 42. 47. 49. 51. 51. 49. 47. 42. 36. 29. 20. 10.  0.
C11 0. 11. 22. 31. 39. 45. 49. 52. 54. 54. 52. 49. 45. 39. 31. 22. 11.  0.
C12 0. 13. 25. 35. 43. 49. 54. 57. 58. 58. 57. 54. 49. 43. 35. 25. 13.  0.
C13 0. 16. 30. 41. 49. 56. 60. 63. 64. 64. 63. 60. 56. 49. 41. 30. 16.  0.
C14 0. 21. 37. 49. 58. 64. 68. 70. 72. 72. 70. 68. 64. 58. 49. 37. 21.  0.
C15 0. 30. 49. 62. 69. 74. 77. 79. 80. 80. 79. 77. 74. 69. 62. 49. 30.  0.
C16 0. 49. 69. 78. 83. 86. 88. 89. 89. 89. 89. 88. 86. 83. 78. 69. 49.  0.
```

The "continue; wait" alias was used, however, and the program cannot complete execution, because you have altered the value of *nbrnodes* midway through the execution, and only on one node. However, you have now identified your problem, and can fix it in the source code and recompile. At this point, enter **<Ctrl-C>** to return to an IPD prompt.

```
^C
(all:0) >
```

Exiting IPD

You have found the problem and assured yourself that the program will run correctly after you fix the bug and recompile, so you can exit IPD with either the **quit** or the **exit** command.

```
(all:0) > quit
*** IPD exiting
%
```

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™ system 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" on page 3-14.

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 <i>count</i> argument.
commshow	full	Fully supported.
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.
msgstyle	full	An option in the message queue display.
process	partial	No -change or -loadfile .
quit	full	Used to exit.
recvqueue	partial	No -type option
remove	partial	No -all option.
rerun	none	Not used.
run	full	Used to restart applications.

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

Command	Support	Explanation
set	none	Not used.
source	partial	No -add or -remove options.
status	none	Not used.
step	partial	no <i>count</i> parameter.
stop	full	Fully supported.
system	full	Fully supported.
threads	full	Fully supported.
trace	partial	Set everywhere a program is loaded. Can not be set for an address. No <i>count</i> 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 <i>count</i> argument.

Starting XIPD

To run XIPD on the Paragon system do the following:

1. Enter the following command on your workstation:

```
% xhost +paragon_system
```

where *paragon_system* is the name of the Paragon system on which you are going to run XIPD.

2. Log onto the Paragon system.
3. Set the *DISPLAY* environment variable to your workstation as in the following example:

```
% setenv DISPLAY machine_name : 0
```

where *machine_name* is the name of your workstation.

4. Check to be sure you have */usr/bin/X11* in your search path.

5. Start XIPD.

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

```
xipd [session_name] [session_option] [control_option] [display_option] [resource_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 with which XIPD should be used.

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.

-core Causes XIPD to go directly to the *Core Analysis* dialog instead of the *Load Application* dialog.

-coreDir *directory* Causes XIPD to go directly to the *Core Analysis* dialog and load the contents of the specified core directory.

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**.

resource_option can be any of the following:

-pn *partition_name* Sets the **partitionName** resource to *partition_name*.

{-program | -prog | -app} *application_name*
Sets the **applicationName** resource to *application_name*.

X Toolkit parameters

The standard parameters supported by the X Toolkit (refer to the *X Toolkit Intrinsic 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 start 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.

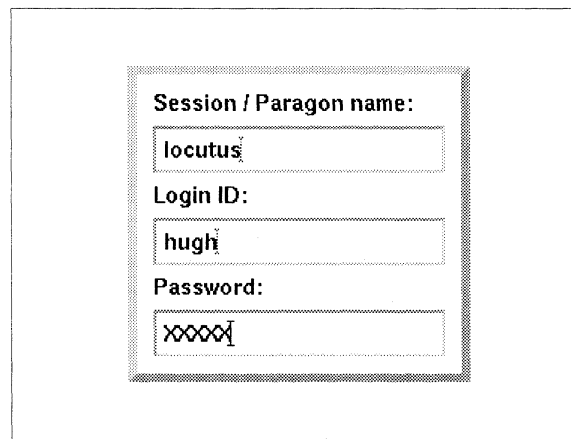
If you prefer to locate your session files in another directory, you can define the environment variable *XIPDHOME* to a directory to store the session files. For example, if you defined *XIPDHOME* to */home/username/misc/sessions*, the session *Galaxy* would be created with the file name */home/username/misc/sessions/.Galaxy*.

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.



The figure shows a rectangular dialog box with a dotted border. Inside the dialog box, there are three vertically stacked input fields. The first field is labeled "Session / Paragon name:" and contains the text "locutus". The second field is labeled "Login ID:" and contains the text "hugh". The third field is labeled "Password:" and contains five "x" characters. The dialog box is centered on the page.

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.

A password is required only if you are using a different account, or if XIPD is running on a remote workstation. This field can remain blank if you are running XIPD on the Paragon system you are going to use for debugging.

Control Buttons

<i>OK</i>	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.
<i>Reset</i>	Resets the entries in the start-up dialog to their original settings.
<i>Help</i>	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.

Host address

The entry in 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.

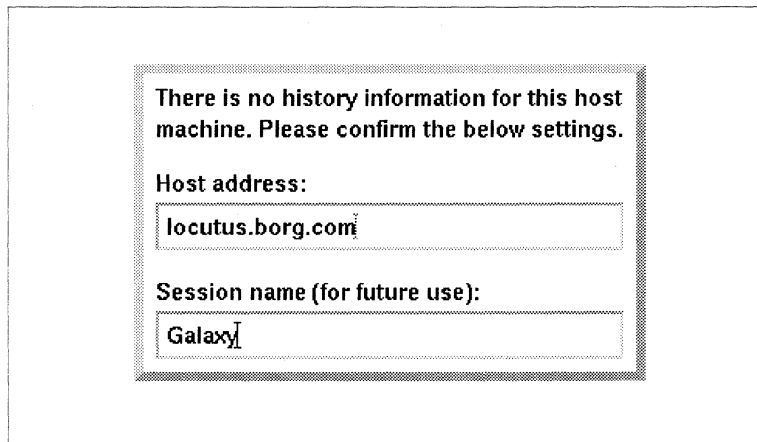


Figure 3-2. Session Name Dialog

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

- | | |
|--------------|---|
| <i>OK</i> | Confirms the dialog entries and dismisses the dialog. The <i>Host address</i> field must be a valid machine address and the <i>Session name</i> 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. |
| <i>Reset</i> | Undoes all the changes to the field contents. |
| <i>Help</i> | Displays help text about the session name dialog. |

Loading a Program

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, XIPD displays the *Load Application* dialog. This dialog allows you to select a partition, select which nodes within the partition compose the mesh for loading programs, and specify the program to load. When you select *OK*, XIPD loads the program into the mesh. The section "Load Application Dialog" on page 3-17 provides a complete description of features of the *Load Application* dialog.

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. After you have entered all your applications, select *OK* to start loading.

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 Process Type* button on the *Load Application* dialog. 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 the process type menu on the *Load Application* dialog lets you switch between the different process types.

Once the load is completed, XIPD displays the main window. Figure 3-3 shows the main window.

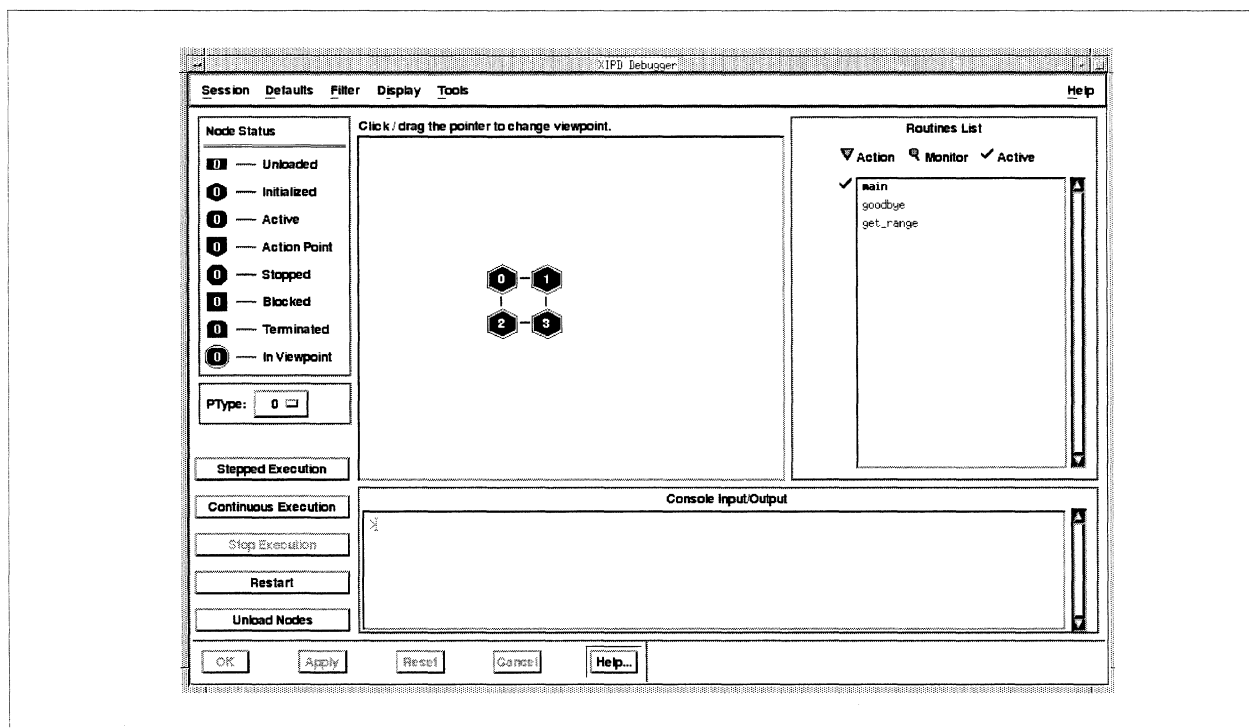


Figure 3-3. XIPD Main Window

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

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 for NX applications. 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. See the Process Type Selection topic later in this section for more information.

For MPI applications, the *Meshes* menu changes from process type selection to communicator group selection once MPI is active. See the Communication Selection topic later in this section for more information.

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 *Create Performance Data* 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 optionally 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 **run** 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.

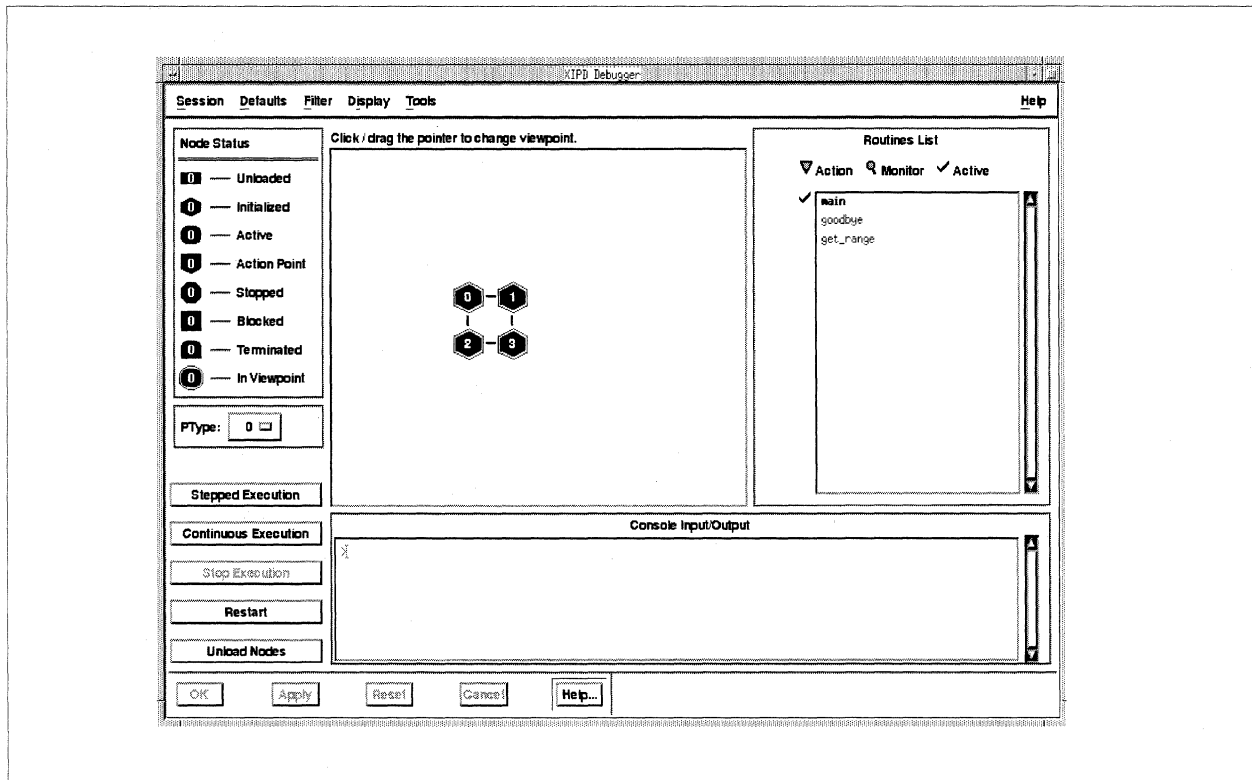


Figure 3-4. Main Window

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 or MPI communicator 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 start XIPD with the **-noanalysis** command line option, and the *Stepped Execution* button is not present when you start XIPD with the **-nodebug** command line option

Session Menu

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

- loading a program
- setting display preferences
- locating the source code for a loaded program
- executing a system command directly
- exiting XIPD

Figure 3-5 shows the session menu.

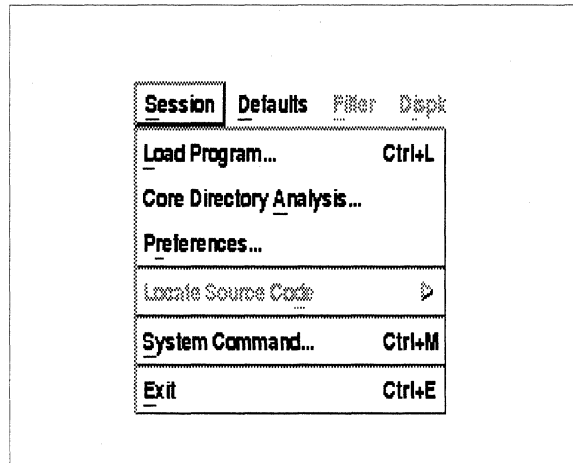


Figure 3-5. Session Menu

Load Program

Load Program displays the *Load Application* 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 <Ctrl-L>. The *Load Application* dialog is described in the section “Load Application Dialog” on page 3-17.

Core Directory Analysis

Core Directory Analysis displays the *Core Analysis* dialog. The *Core Analysis* dialog is described in the section “Core Analysis Dialog” on page 3-27.

Preferences

Preferences displays the XIPD preferences dialog. *Preferences* is always enabled. The preferences dialog is described in the section “Preferences Dialog” on page 3-33.

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 code 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. The code location dialog is described in the section "Code Location Dialog" on page 3-38.

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 <Ctrl-M>. The *System Command* dialog is described in the section "System Command Dialog" on page 3-39.

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 <Ctrl-E>.

Load Application Dialog

The *Load Application* dialog allows you to load an application. You can bring up the *Load Application* dialog with the *Load Program* item from the *Session* menu. The *Load Application* dialog contains the following regions:

- Application Information
- Application Options
- Partition Selection
- Load Options
- Load Status Messages
- Control Buttons

Figure 3-6 shows the *Load Application* dialog.

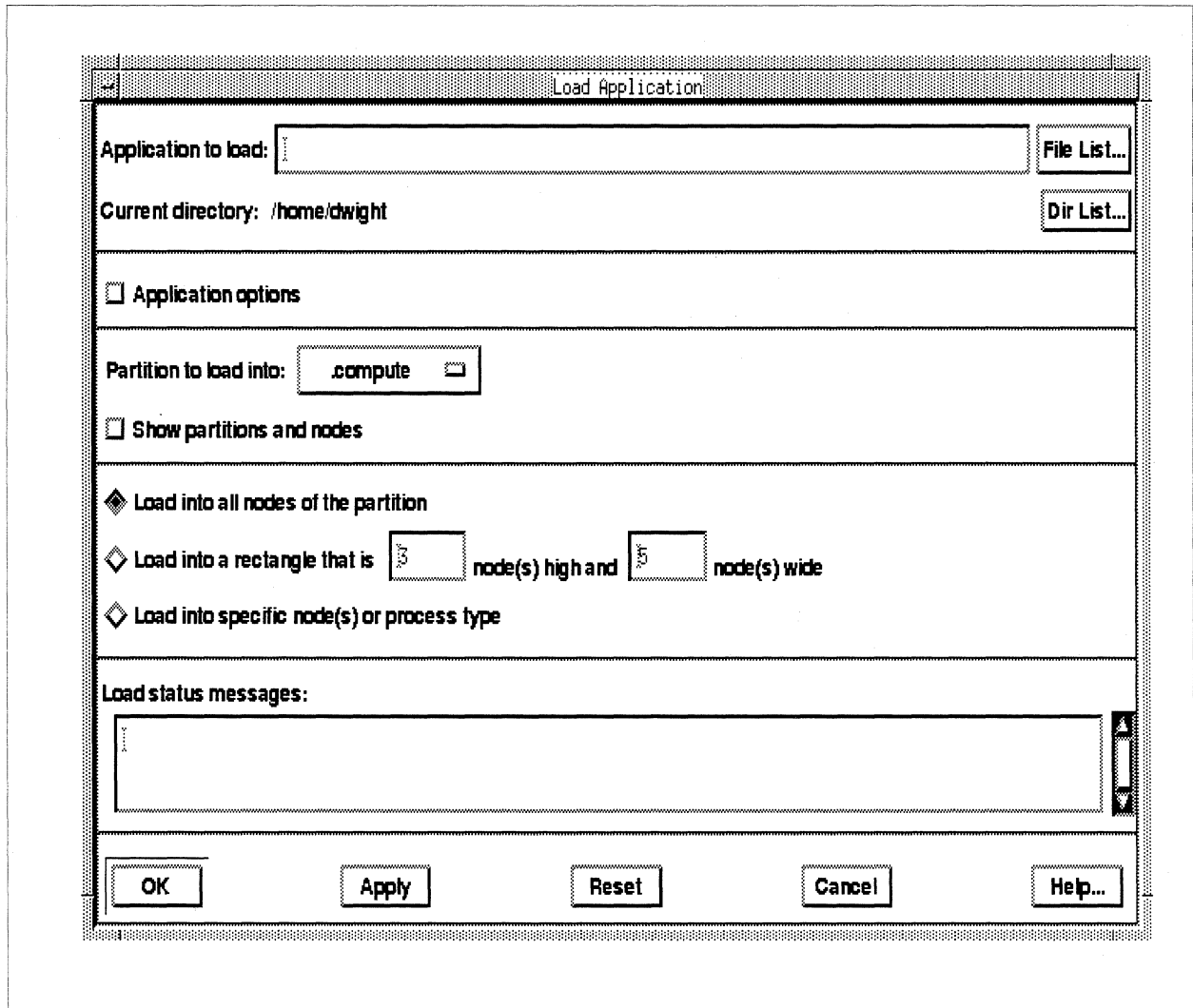


Figure 3-6. Load Application Dialog

Application Information

The application information area of the *Load Application* dialog allows you to do the following:

- specify an application to load
- identify the current directory

Application to Load

The *Application to Load* text field allows you to enter the name of the application you wish to load. Next to this text field is the *File List* button. Selecting this button displays the file selection dialog. If you select a file from the file selection dialog, the entry in the *Application to Load* text field is updated. The file selection dialog is described in the section “File Selection” on page 3-71.

Current Directory

The *Current Directory* field lists the current directory. Next to this field is the *Dir List* button. Selecting this button displays the file selection dialog. If you select a directory from the file selection dialog, the *Current Directory* field is updated. The file selection dialog is described in the section “File Selection” on page 3-71.

Application Options

The Application Options area of the *Load Application* dialog is an expandable area that allows you to do the following:

- specify command line arguments
- specify an input file for redirection
- specify an output file for redirection
- specify if the application is a controlling process

When the *Load Application* dialog appears, the Application Options section only contains a toggle button. When you set this button, an area containing the application options opens under the toggle button. If you reset the button, this area is collapsed and only the toggle button is visible.

Figure 3-7 shows the expanded Application Options section.

Application options

Arguments:

Input file: File List...

Output file: File List...

Controlling / service application

Figure 3-7. Application Options Section of Load Application Dialog

Arguments

The *Arguments* text field allows you to specify command line arguments to be passed to the application being loaded.

Input file

The *Input File* text field allows you to specify the name of a file to be used for input redirection. If you do not want to use input redirection, this field should be left blank. Next to the *Input File* text field is a *File List* button. Selecting this button displays the file selection dialog. If you select a file from the file selection dialog, the entry in the Input File text field is updated. The file selection dialog is described in the section "File Selection" on page 3-71.

Output file

The *Output File* text field allows you to specify the name of a file to be used for output redirection. If you do not want to use output redirection, this field should be left blank. Next to the *Output File* text field is a *File List* button. Selecting this button displays the file selection dialog. If you select a file from the file selection dialog, the entry in the Output File text field is updated. The file selection dialog is described in the section "File Selection" on page 3-71.

Controlling/service application

The *Controlling/service application* toggle button should be set if the application being loaded is a controlling process.

By default, XIPD 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 set the *Controlling/service application* toggle button to suppress passing these options.

Partition Selection

The Partition Selection area of the *Load Application* dialog allows you to select in which partition the application should run. This area allows you to do the following:

- specify the partition to load into
- show partitions and nodes

Partition to Load Into

Partition to Load Into contains the name of the currently selected partition and a pull-down menu that includes the names of all partitions in the *.compute* partition. Selecting a name from the menu updates the currently selected partition.

Show Partitions and nodes

If you set the *Show partitions and nodes* toggle button in the Partition Selection area, a graphical display of the partition tree appears under the toggle button. This display allows you to do the following:

- select a partition
- view the partition location

Figure 3-8 shows the expanded Partition Selection section.

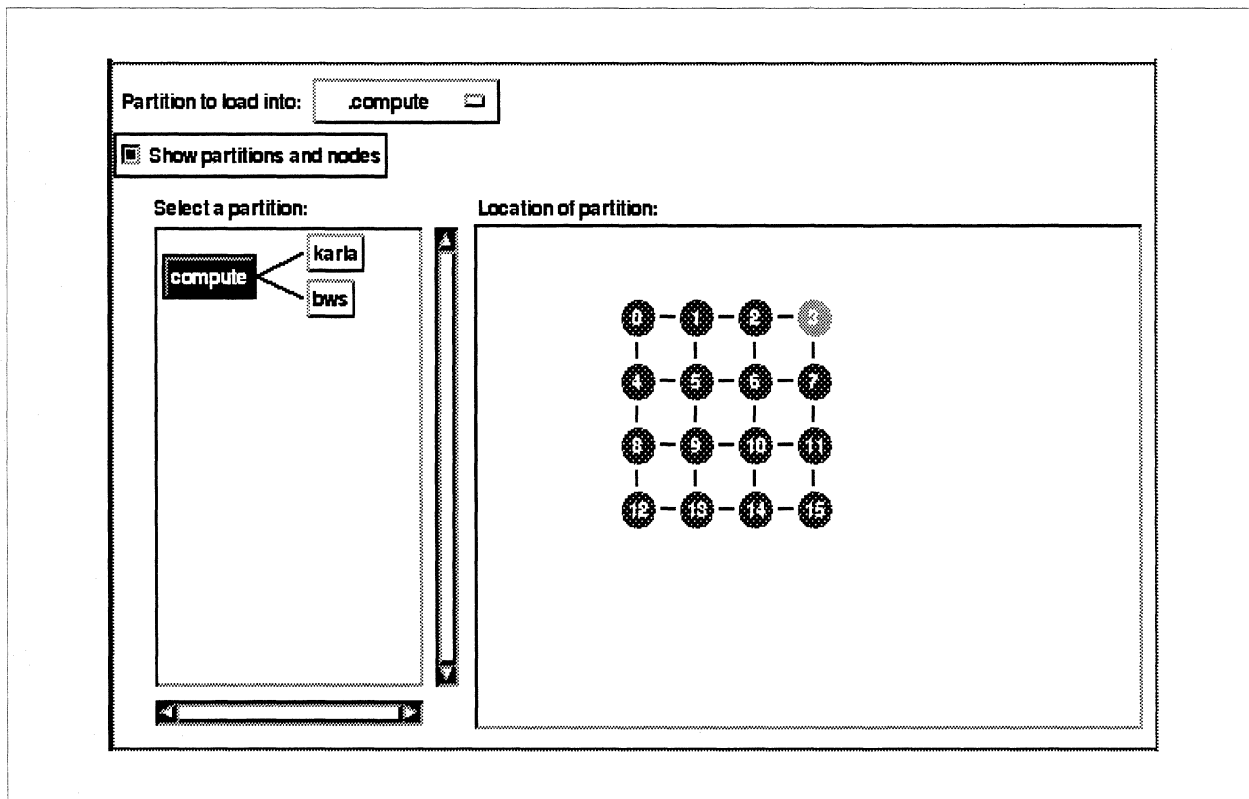


Figure 3-8. Partition Selection Section of Load Application Dialog

Select a Partition

This area shows a graphical display of the partitions allocated under the *.compute* partition. The currently-selected partition is highlighted. When you select a partition, the *Partition to Load Into* field is updated.

Location of Partition

This area shows a graphical representation of the physical mesh, highlighting the nodes in the currently-selected partition. The colors and patterns used to highlight the nodes are display-specific.

Load Options

The Load Options area of the *Load Application* dialog allows you to specify which nodes in the selected partition should be loaded with the application. You can do one of the following:

- load into all nodes
- load into a rectangle of nodes
- load into specific nodes

Initially, this area contains only three radio buttons for the three load options, and the load into all nodes option is set.

Load into All Nodes

If this radio button is set, the application is loaded onto all nodes in the selected partition.

Load into Rectangle

The *Load into a rectangle* option allows you to specify a rectangle of nodes into which the application is loaded. You select this option by setting the radio button and specifying the height and width of the rectangle in the adjacent boxes. You must be sure that the dimensions you specify for the rectangle are valid for the selected partition. If you specify an invalid rectangle, you will receive an allocator error message when XIPD attempts to load your application into the rectangle.

Load into Specific Nodes

If you set the radio button to load into specific nodes, the Load Options area expands to include the following:

- a node panel to select specific nodes
- a process type option menu
- a new process type push button
- buttons to set load mesh height and width
- a button to select all nodes
- a button to unselect all nodes

Figure 3-9 shows the expanded Load Options section.

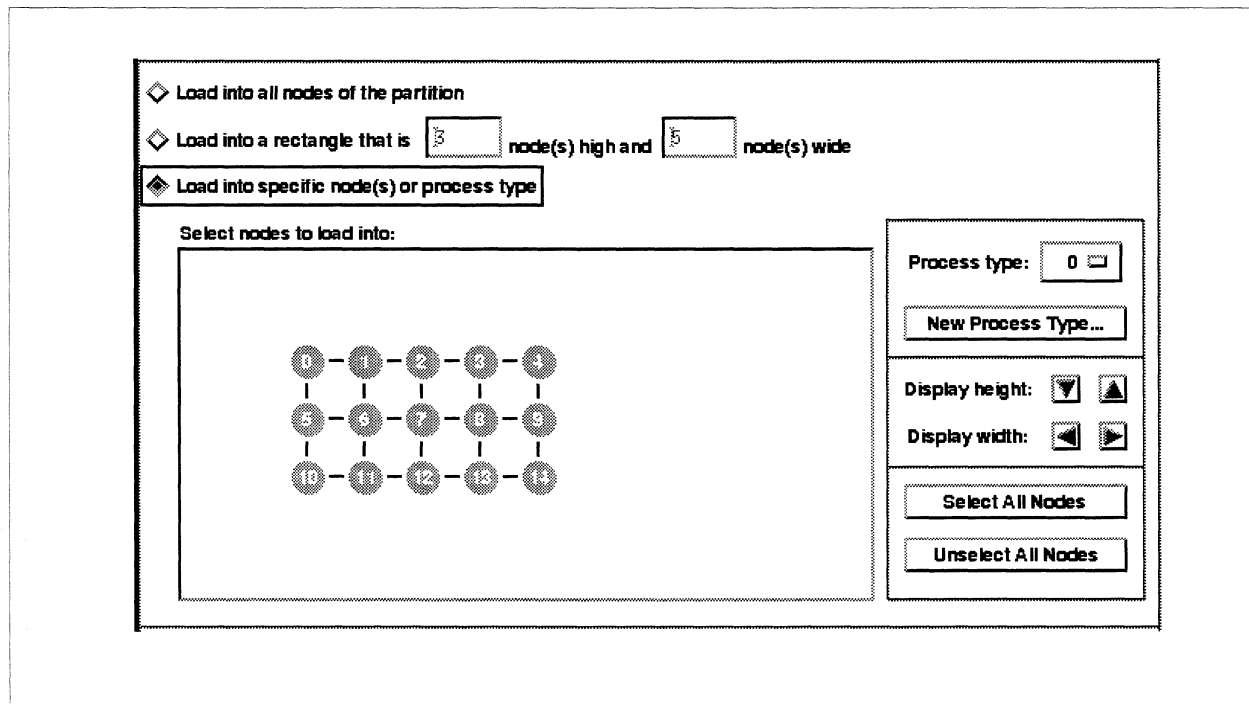


Figure 3-9. Load Options Section of Load Application Dialog

Node Selection

The node panel allows you to select specific nodes to load your application into. The node panel contains a representation of the nodes in the selected partition. The nodes are displayed as a mesh. The height and width of the mesh is logical (based on the number of nodes in the mesh), since the nodes in the partition may actually be scattered in a non-rectangular pattern. When you select specific nodes for loading, your selection must start with the first node in the specified partition, and the nodes selected must be contiguous.

Nodes in the display are in one of three states:

Unselected	Can be loaded into
Selected	Will be loaded at the next OK or Apply
Loaded	Already loaded and can't be loaded again

The colors and patterns used to render the state of the nodes are display-specific.

You can select a node by clicking on it. You can select a group of nodes by holding down the first mouse button and dragging the pointer over the group. Clicking on a selected node deselects the node.

Process Type Selection

The *Process type* menu allows you to choose a process type. Each process type allocated with the *New Process Type* push button is given a specific set of nodes for loading. When you select a process type from the *Process type* menu, the node panel is updated to reflect the status of the nodes associated with that process type.

New Process Type

When you select the *New Process Type* push button, a dialog appears to allow you to specify a new process type. Enter a number in the dialog box and press the dialog's *OK* button to allocate a new process type for loading into. This new process type is then incorporated onto the *Process type* menu.

Figure 3-10 shows the *New Mesh Process Type* dialog.

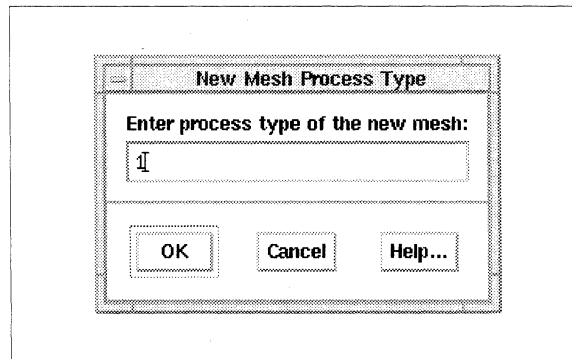


Figure 3-10. New Mesh Process Type Dialog

Display Height / Width

The *Display height* and *Display width* arrows have the following effects on the load mesh:

Display height down	decreases the height and increases the width
Display height up	increases the height and decreases the width
Display width left	decreases the width and increases the height
Display width right	increases the width and decreases the height

If it is not possible to do the requested change on the load mesh, a beep will sound.

Select All Nodes

The *Select All Nodes* push button changes all nodes in the partition that are not currently in the *Loaded* state to the *Selected* state.

Unselect All Nodes

The *Unselect All Nodes* push button changes all nodes in the partition that are not currently in the *Loaded* state to the *Unselected* state.

Load Status Messages

The Load Status Messages area of the *Load Application* dialog is a scrolling text area that contains messages about the current state of the application load process. This area clears every time the dialog appears. You can not edit the text in this area, but you can select it and paste it into another X client.

Control Buttons

The control buttons on the *Load Application* dialog have the following functions:

OK

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

- No file name is given.
- No nodes are selected and *Controlling/service application* is not set.
- Nodes are selected and *Controlling/service application* is set.

If there are no errors, XIPD starts IPD and sends IPD a **load** command.

Apply

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

- No file name is given.
- *Controlling/service application* is set.
- No nodes are selected and *Controlling/service application* is not set.

Reset

Reset eliminates all changes since the last *Apply*.

Cancel

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

Help

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

Core Analysis Dialog

When you elect to do core file analysis, XIPD displays the *Core Analysis* dialog. The *Core Analysis* dialog allows you to do the following:

- Specify a core directory.
- Display the core directory contents.
- Specify selection and display options.
- Specify partition and node options.

Figure 3-12 shows the *Core Analysis* dialog.

Core Directory

The *Core Directory* field contains the relative or full path name of a core file directory from which you can choose core files. You can type in the core file directory name and press <Return> to load the contents of the directory into the core directory contents field.

Next to the Core Directory field is a Directory List button. If you select this button, XIPD displays the file selection dialog. If you choose a core directory from the dialog, it is copied into the *Core Directory* field.

Core Directory Contents

The *Core Directory Contents* field contains a scrolling list of files in the specified core directory. The list contains the following information for each entry:

- The executable's path.
- The node number.
- The ptype number.
- The exit signal.
- The type of core dump.
- Whether or not the entry is selected.

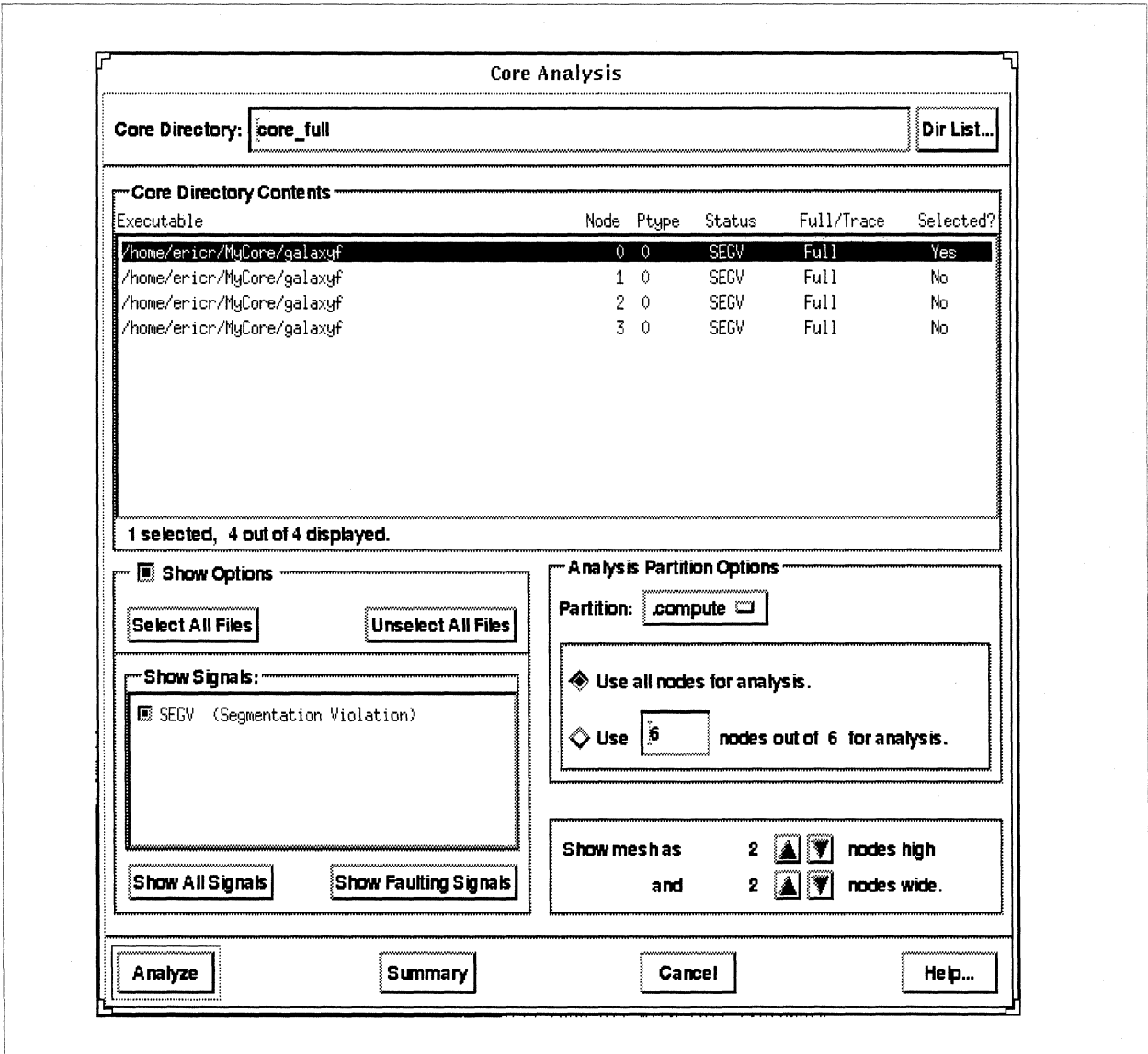


Figure 3-11. Core Analysis Dialog

Below the *Core Directory Contents* field is a status line that provides the following information:

- How many files are selected.
- How many files are in the list.
- How many files are in the core directory.

Show Options

If the *Show Options* button is set, this area of the dialog expands to include the following options:

- Select All Files
- Unselect All Files
- Show Signals
- Show All Signals
- Show Faulting Signals

When the Show Options button is not selected, this area collapses and only the *Show Options* button is visible.

Select All

If you select the *Select All* push button, XIPD selects all remaining unselected entries in the *Core Directory Contents* window. If you want to select only the faulting processes, you must first select the *Show Faulting Signals* button and then select the *Select All* button.

Unselect All

If you select the *Unselect All* push button, XIPD unselects all selected entries in the *Core Directory Contents* window.

Show Signals

The *Show Signals* window contains an entry for each type of signal contained in the loaded core directory. Next to each signal type is a toggle button. If the toggle button is set, all core files that terminated with that signal type are included in the *Core Directory Contents* list. If the toggle button for a signal type is not set, all core files that terminated with that signal type are removed from the *Core Directory Contents* list.

Show All Signals

The *Show All Signals* button sets all of the toggle buttons in the *Show Signals* window so all core files are displayed in *Core Directory Contents* list.

Show Faulting Signals

The *Show Faulting Signals* button sets all of the toggle buttons for faulting signals and unsets the toggle buttons for non-faulting signals in the *Show Signals* window. This causes only core files that terminated with faulting signals to be displayed in the *Core Directory Contents* list.

Partition and Node Options

This section of the dialog contains items to do the following:

- Select a partition for core file analysis.
- Use all nodes for analysis.
- Use some nodes for analysis.
- Display and set mesh height and width.

Partition

The *Partition* button pulls displays a menu with the names of all partitions in the compute area of the mesh. You can select a partition from this menu for core file analysis. The default partition is determined by the following:

XIpd.partitionName	If this application resource is not blank, it becomes the default selected partition.
\$NX_DFLT_PART	If XIpd.partitionName is blank or does not exist, the value of this environment variable becomes the default selected partition.
.compute	If neither XIpd.partitionName or \$NX_DFLT_PART are defined, the .compute partition becomes the default selected partition.

Use All Nodes

If the Use all nodes toggle button is set, all nodes in the selected partition are used for core file analysis.

Use Some Nodes

If the *Use num nodes* toggle button is set, you can specify how many nodes in the specified partition you want to use for core file analysis. You enter the number in the text field. The number can range from one to the total number of nodes in the specified partition.

Set Mesh Dimensions

This area of the dialog displays the height and width of the mesh, and provides arrow buttons for redefining the height and width. The product of the height and width equals the number of nodes being used for core file analysis. If you attempt to define an invalid mesh height or width, XIPD beeps.

Control Buttons

The *Core Analysis* dialog contains the following control buttons:

<i>Analyze</i>	Starts interactive analysis of the selected core files. If no files are selected, an error dialog is displayed.
<i>Summary</i>	Displays a summary traceback report of all selected core files in a separate summary dialog. If no files are selected, an error dialog is displayed. The summary dialog contains a text area for the summary report, a <i>Done</i> button to dismiss the dialog, and a <i>Help</i> button to display help text about the summary dialog.
<i>Cancel</i>	Dismisses the Core Analysis dialog.
<i>Help</i>	Displays help text for the <i>Core Analysis</i> dialog.

NOTE

After a core directory load, you may add or remove files from the load to refine the current session. When *Analyze* is selected, the load is adjusted. Selecting a new core file directory ends the current analysis.

Core Load and Load are exclusive—to switch between them you must exit the current dialog.

Summary Report

The *Core File Summary* report is a brief analysis of the selected core files. The scrollable text area of the dialog contains the following:

- The result of IPD scanning the core file for analysis. If *Show Threads* is turned on, a threads -on command is sent to IPD.
- The output of **frame (all:all)**, which is a stack traceback for all of the selected core files. This shows how the node reached the point it was at when the core file was generated.
- The output of **status**, which provides information about the environment the original faulting application ran within.

Selecting the *Done* button dismisses the dialog.

Figure 3-12 shows a Core File Summary report.

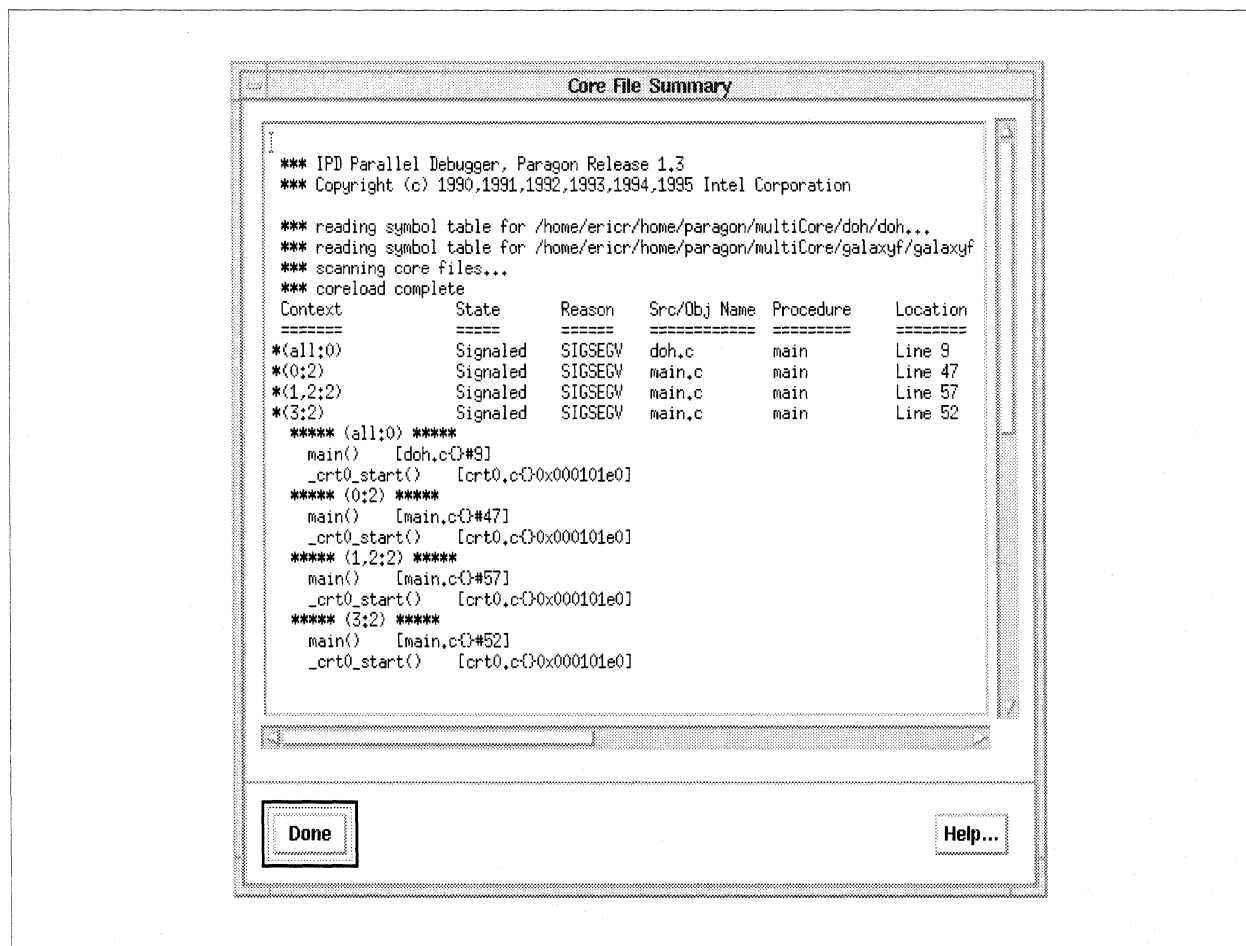


Figure 3-12. Core File Summary

Figure 3-12 shows a Core File Summary including thread information.

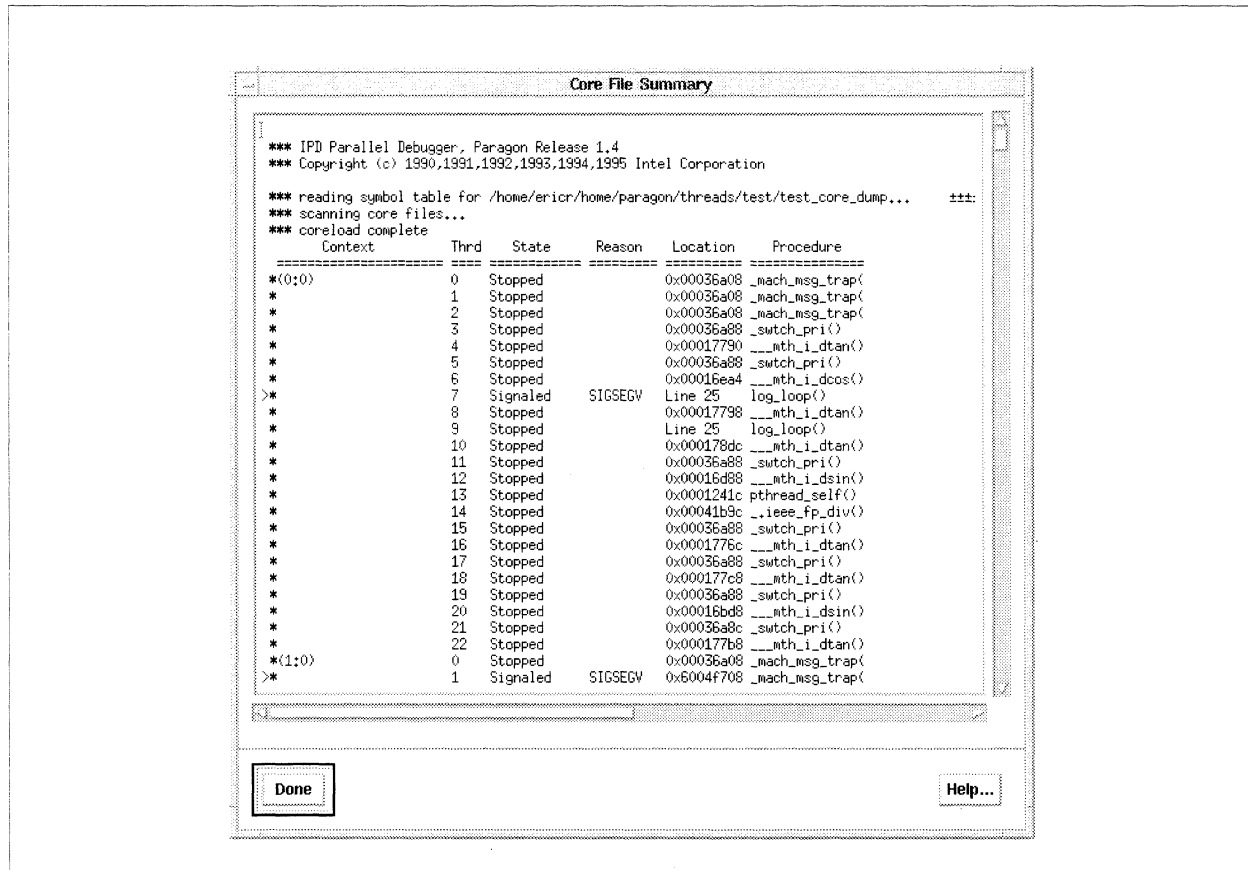


Figure 3-13. Core File Summary With Thread Information

Preferences Dialog

The *preferences* dialog configures the appearance of XIPD. Changes are written to your *XIPd* resource file. Figure 3-14 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

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:

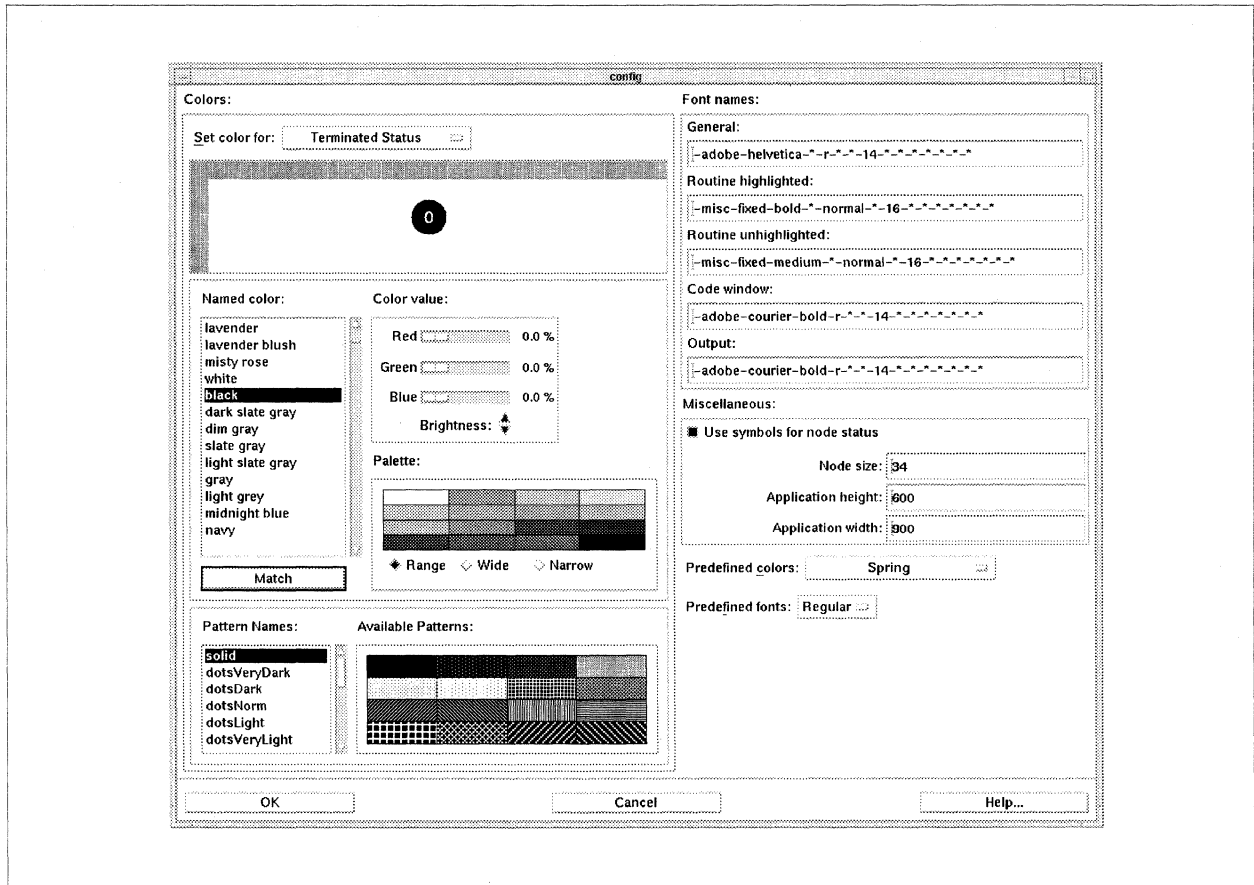


Figure 3-14. Preferences Dialog

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 field contains 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:

<i>Range</i>	Modifies the palette to cover the entire spectrum range.
<i>Wide</i>	Modifies the palette to cover a wide range of colors around the current color.
<i>Narrow</i>	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:

<i>General</i>	Used for most XIPD interface text, such as labels and menu names.
<i>Routine highlighted</i>	Used for routine list entries that are highlighted.
<i>Routine unhighlighted</i>	Used for routine list entries that are not highlighted.
<i>Code window</i>	Uses in the code window. This should be a fixed-width font.
<i>Output</i>	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.

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 3-15 shows the code location dialog.

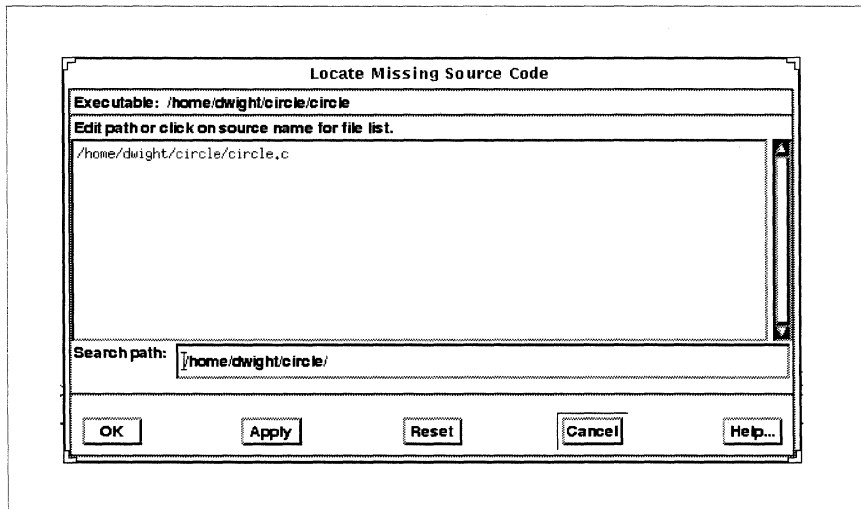


Figure 3-15. Code Location Dialog

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. The file selection dialog is described in the section "File Selection" on page 3-71.

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

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

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 3-16 shows the *System Command* dialog.

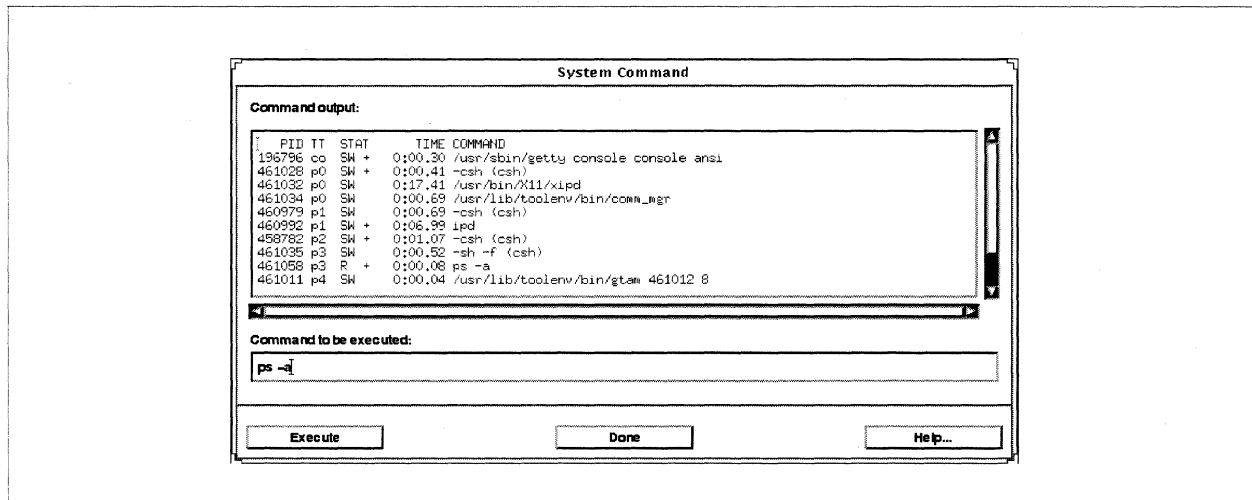


Figure 3-16. System Command Dialog

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

<i>Execute</i>	Sends the contents of the <i>Command to be executed</i> field to IPD, prefixed by the IPD system command. All output, up to the next IPD prompt, is put into <i>Command output</i> .
<i>Done</i>	Dismisses the dialog.
<i>Help</i>	Displays help topic text about the 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 3-17 shows the *Defaults* menu.

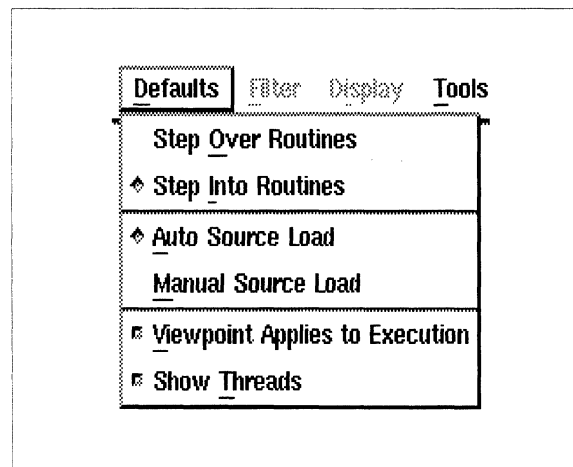


Figure 3-17. Defaults Menu

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.

Show Threads

When *Show Threads* is set, thread information is displayed in the node viewpoint, traceback, variable, and core summary report displays. When *Show Threads* is not set, only information about the active thread is shown. The default is *off*.

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 3-18 shows the *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.

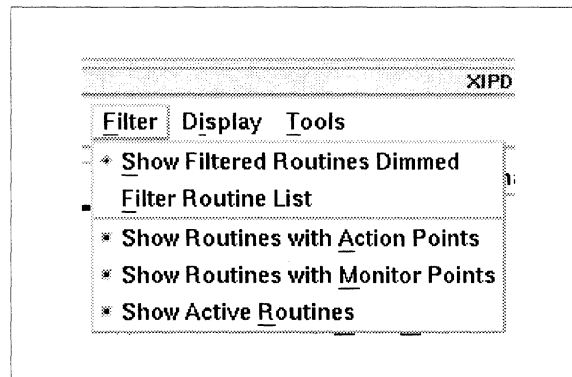


Figure 3-18. Filter Menu

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 3-19 shows the *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 <Ctrl-S>.

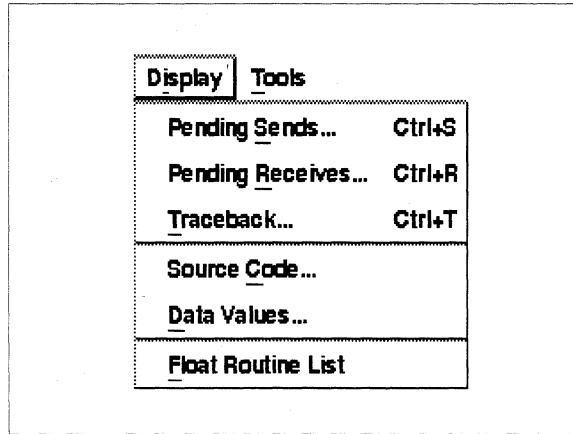


Figure 3-19. Display Menu

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 <Ctrl-R>.

Traceback

Traceback displays the *Traceback* dialog. The default keyboard accelerator for this item is <Ctrl-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.

Float Routine List

Float Routine List allows you to change the main window routine list into a separate floating dialog. For a description of the routine list, refer to the section "Routine List" on page 3-67.

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 3-20 shows the *Pending Sends* dialog.

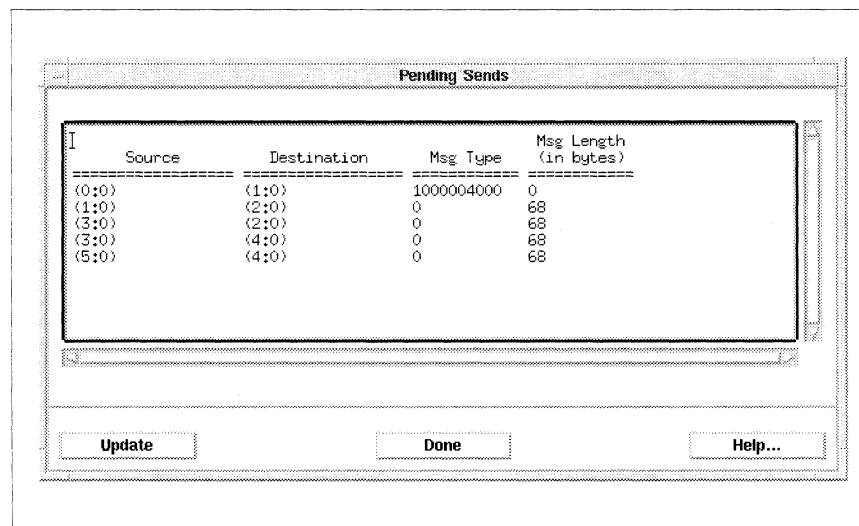


Figure 3-20. Pending Sends Dialog

Dialog Buttons

- | | |
|---------------|---|
| <i>Update</i> | 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. |
| <i>Done</i> | Dismisses the dialog. |
| <i>Help</i> | Displays help topic text about the dialog. |

When MPI is active, a frame area in the dialog includes two additional toggle buttons:

All: Show all MPI and NX based messages Sends a **-all** switch along with the IPD msgqueue command.

NX: Show message queue in NX style Sends a **-nx** switch along with the IPD msgqueue command.

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 3-21 shows the *Pending Receives* dialog.

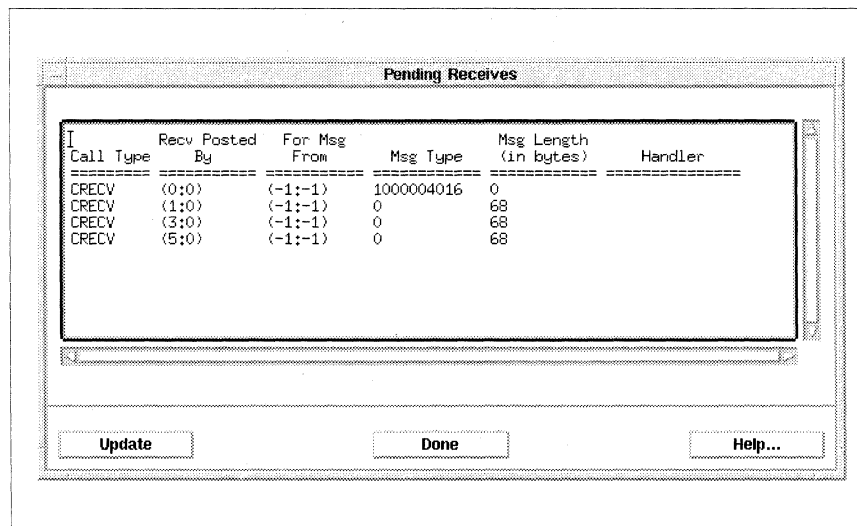


Figure 3-21. Pending Receives Dialog

Dialog Buttons

<i>Update</i>	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.
<i>Done</i>	Dismisses the dialog.
<i>Help</i>	Displays help topic text about the dialog.

When MPI is active, a frame area in the dialog includes two additional toggle buttons:

<i>All: Show all MPI and NX based messages</i>	Sends a -all switch along with the IPD recvqueue command.
<i>NX: Show message queue in NX style</i>	Sends a -nx switch along with the IPD recvqueue command.

Traceback Dialog

The *Traceback* dialog is displayed when you select the *Traceback* menu item. This dialog contains the output of the **frame** command. If *Show Threads* is turned on the output includes frame output for all threads in the viewpoint's selected nodes. 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 calling routine's location, and so on.

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

Figure 3-22 shows the *Traceback* dialog.

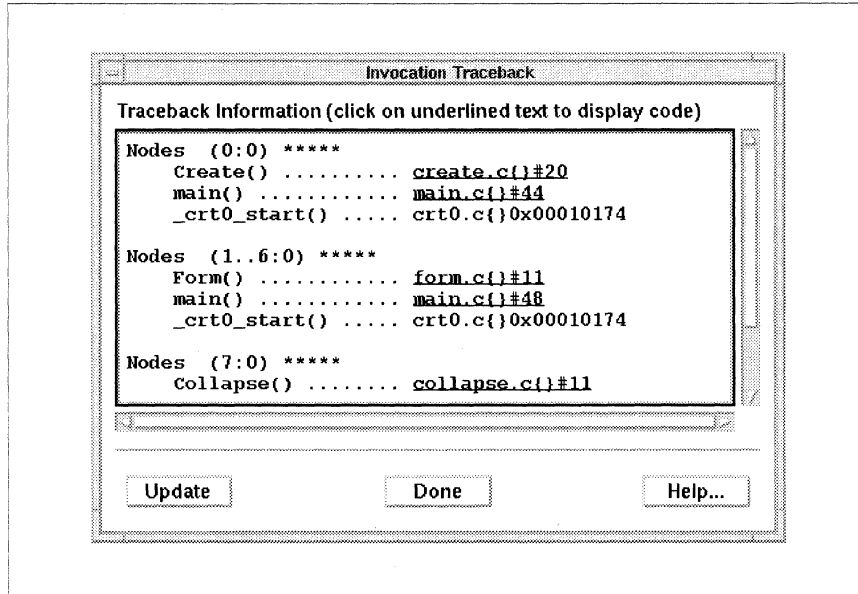


Figure 3-22. Traceback Dialog

Figure 3-22 shows a *Traceback* dialog that includes frame information from multiple threads.

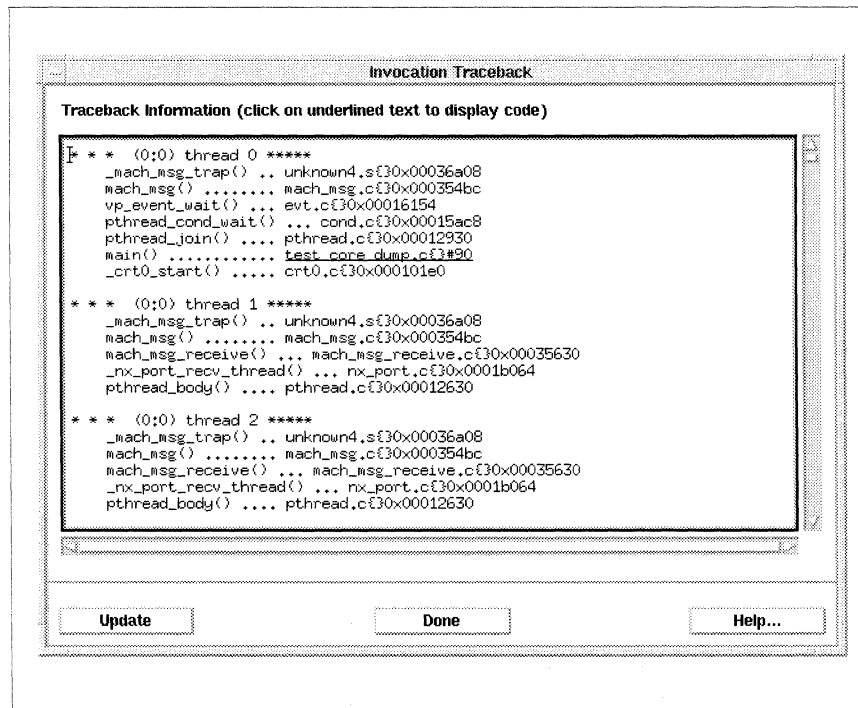


Figure 3-23. Traceback Dialog With Multiple Threads

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

<i>Update</i>	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.
<i>Done</i>	Dismisses the dialog.
<i>Help</i>	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 *Create Performance Data* dialog. This menu is not present if you started XIPD with the **-noanalysis** command line option. Figure 3-24 shows the tools menu.

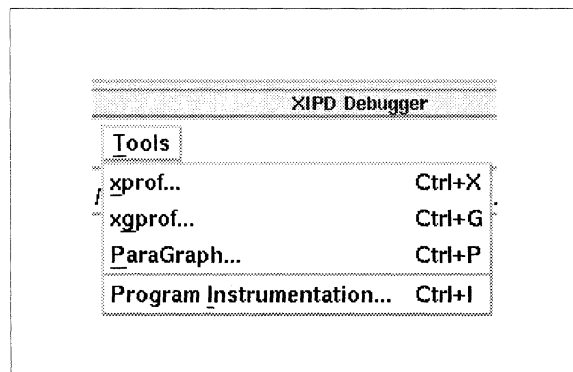


Figure 3-24. Tools Menu

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 **<Ctrl-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 <Ctrl-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 <Ctrl-P>.

Instrument Code

Instrument Code displays the *Create Performance Data* 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 <Ctrl-I>.

XProf Dialog

Figure 3-25 shows the *XProf* dialog.

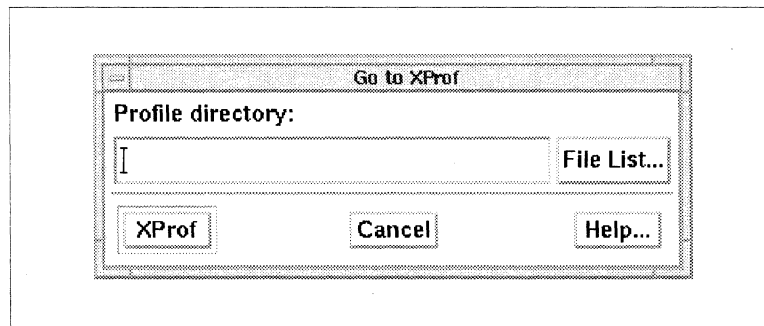


Figure 3-25. XProf Dialog

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

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

XGprof Dialog

Figure 3-26 shows the *XGprof* dialog.

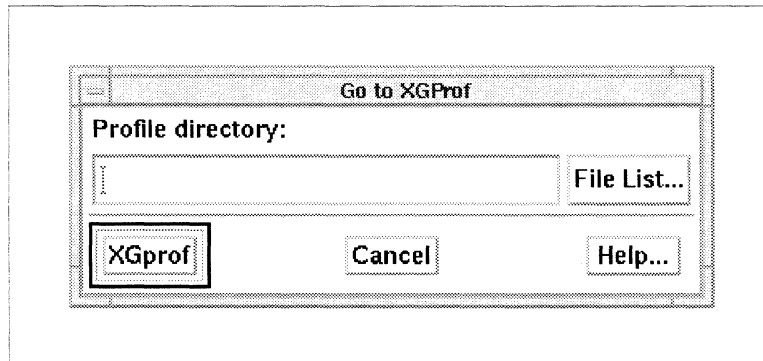


Figure 3-26. XGprof Dialog

Profile directory

This field contains the name of the **gprof** output directory (for example, */gmon.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

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

ParaGraph Dialog

Figure 3-27 shows the *ParaGraph* dialog.

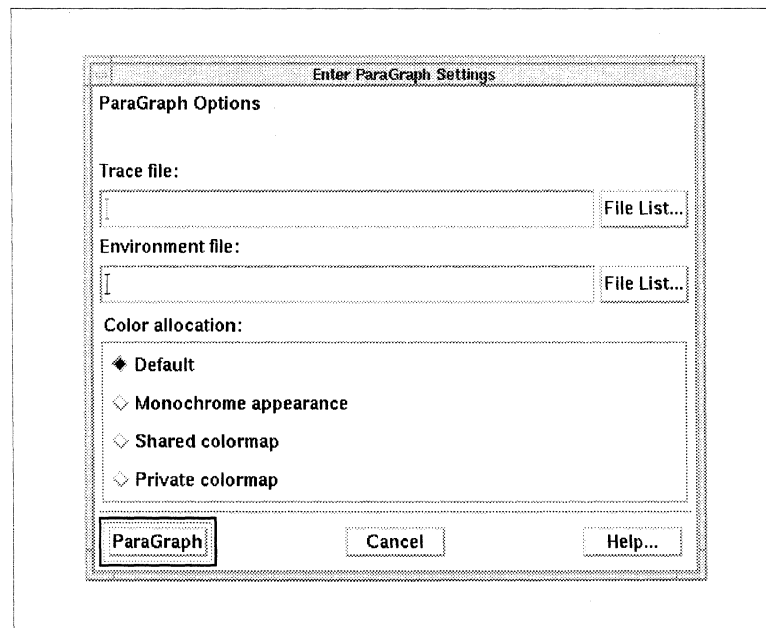


Figure 3-27. Enter ParaGraph Settings Dialog

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

<i>Default</i>	ParaGraph defaults to its own color selection scheme.
<i>Monochrome</i>	ParaGraph appears in black and white.
<i>Shared colormap</i>	ParaGraph uses the shared colormap, which might result in ParaGraph not having enough colormap space for it to display all the colors it needs.
<i>Private colormap</i>	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

<i>ParaGraph</i>	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 starts ParaGraph with the given options. If a given file name is not valid, XIPD displays an error dialog.
<i>Cancel</i>	Dismisses the dialog and discards any changes to the dialog.
<i>Help</i>	Displays help text for the dialog.

Create Performance Data Dialog

Use the *Create Performance Data* 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 3-28 shows the *Create Performance Data* dialog.

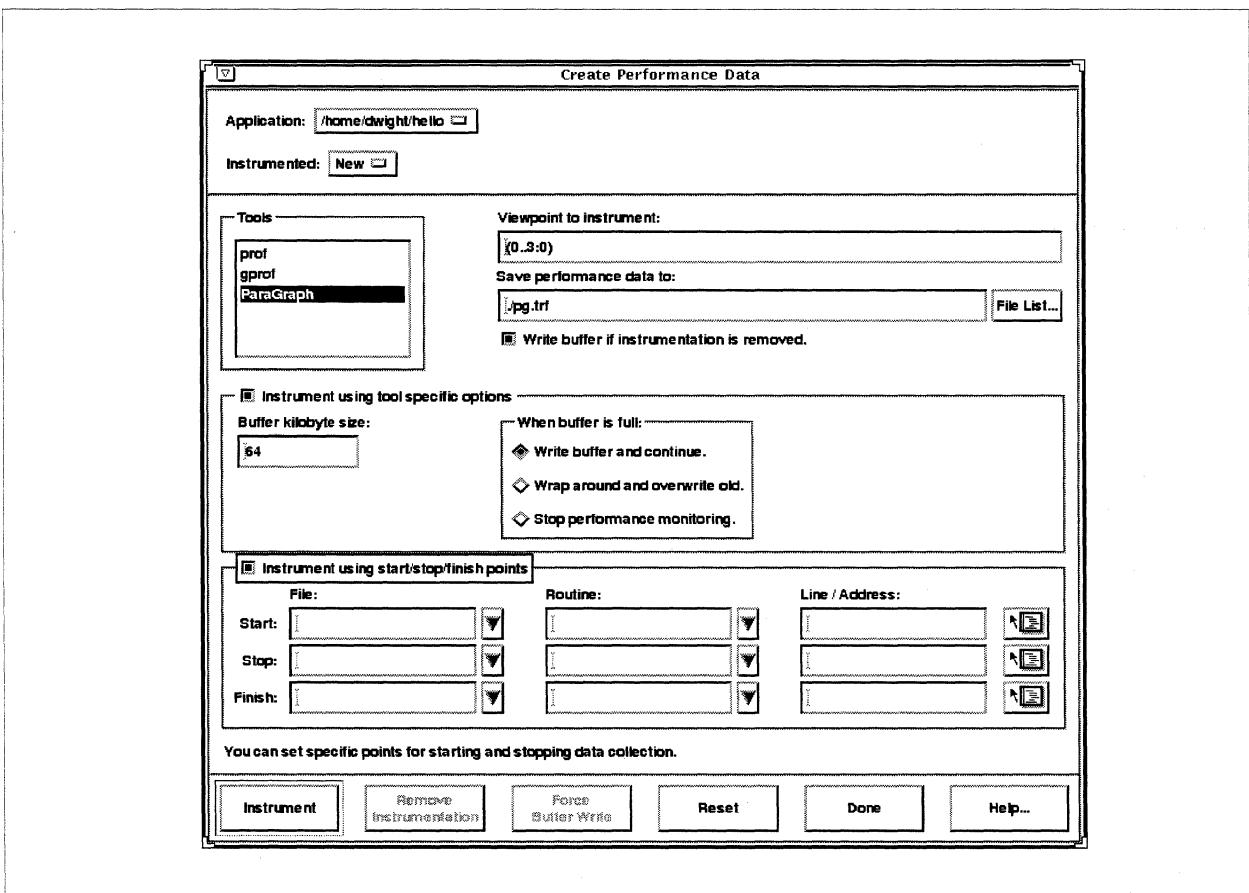


Figure 3-28. Create Performance Data Dialog

Application

The *Application* list option menu contains the names of all the loaded applications, or [empty] if no applications are loaded. When you select an application from the list, it becomes the selection of the dialog. The option menu selection defaults to the first executable loaded.

Instrumented

The *Instrumented* option menu initially contains only the item *New*. As you instrument contexts, the contexts are added to this menu. When you select an item from the *Instrumented* option menu, the fields of the dialog are updated to show how that context was instrumented.

When *New* is selected, you can edit all fields of the dialog, and the *Remove Instrumentation* and *Force Buffer Write* buttons are disabled. When you select a context from the option menu, most of the dialog fields become non-editable, and the *Remove Instrumentation* and *Force Buffer Write* buttons become enabled.

Tools

This radio box contains the names of all the valid analysis tools. This includes *prof*, *gprof*, and *ParaGraph*. The default tool selection is *prof*.

When you select a tool, the default file name for that tool appears in the *Save performance data to* field, and any tool-specific options appear in the *Instrument using tool-specific options* section of the dialog (if that section is expanded).

Viewpoint to Instrument

This text field reflects the viewpoint to be instrumented (if *New* is selected) or the viewpoint that has already been instrumented (if an existing instrumented context is selected).

Save Performance Data to

This text field contains the name of the directory or file to which performance data is saved. Whenever an analysis tool is selected, the default file name for that tool appears in this field.

The *File list* push button next to this field displays the file selection dialog. If you select a file from the dialog, the selection is put into the *Save Performance Data to* text field.

Write buffer if instrumentation is removed

If this option is set when instrumentation is removed, IPD is instructed to save 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.

Tool-Specific Options

This section of the *Create Performance Data* dialog is collapsible. You can expand and collapse it by clicking on the *Instrument using tool specific options* push button. This section of the dialog provides items to do the following:

- Define the trace event buffer size for ParaGraph.
- Define what to do when the buffer is full for ParaGraph.

Buffer Kilobyte Size

This is a text field that contains the size of the trace event buffer to be used to collect performance data. The default size is 64 kilobytes. This field is enabled only when ParaGraph is selected.

When buffer is full

This radio box is enabled when ParaGraph is chosen as the performance tool. The contents control what is done when the data collection buffer is full. The following options are available:

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

Point-Specific Options

This section of the *Create Performance Data* dialog is collapsible. You can expand and collapse it by clicking on the *Instrument using start/stop/finish points* push button.

This section of the dialog provides items to do the following:

- Specify file names for start, stop, and finish points.
- Specify routine names for start, stop, and finish points.
- Specify line numbers or addresses for start, stop, and finish points.
- Link to a source code window for setting start, stop, and finish points.

A *start point* is a point where performance data collection is turned on during execution of the specified application.

A *stop point* is a point at which performance data collection is suspended during execution. Performance data collection can be resumed by a following start point.

A *finish point* is a point at which performance data collection is stopped during execution. A finish point differs from a stop point because performance data collection can not be resumed following a finish point.

File

The *File* column contains text fields to specify file names for start, stop, and finish points for instrumenting the current application. You can enter a file name in the text field or select the button to the right of the field to display a popup menu of all known file names for the application. The file name popup menu is displayed only if there is debug information about the current executable. Selecting an entry from the popup menu copies the file name from the menu to the *File* text field. If there are no known source files for the current executable, an information dialog is displayed.

Routine

The *Routine* column contains text fields to specify routine names for start, stop, and finish points for instrumenting the current application. You can enter a routine name in the text field or select the button to the right of the field to display a popup menu of all known routines for the file specified in the adjacent *File* text field. The routine popup menu is displayed only if there is debug information about the file specified in the adjacent *File* text field. Selecting an entry from the popup menu copies the routine name from the menu to the *Routine* text field. If there are no known routines in the file specified in the adjacent *File* text field, or if the adjacent *File* text field is empty, an information dialog is displayed.

Line/Address

The *Line/Address* column contains text fields to specify line numbers (for the specified source file) or addresses for start, stop, and finish points for instrumenting the current application. If an address is specified, the field must start with 0x or 0X.

Source Code Link

The icons to the right of the *Line/Address* column allow you to set start, stop, and finish points from source code in a code window. If you select a source code link icon, the icon blinks. If you click with the pointer on a line in a code window while the icon is blinking, the file name, routine name, and line number of where you clicked are copied into the text fields of the row adjacent the icon.

Selecting a source code link icon does not open a code window. A code window must already be open, or you can open one after you select the icon.

Status Messages

The area between the point-specific options and the control buttons is used for the display of status messages about the *Create Performance Data* dialog.

Control Buttons

The *Create Performance Data* dialog has the following dialog buttons:

<i>Instrument</i>	Starts instrumentation based on the entries in the <i>Create Performance Data</i> dialog. As much error checking as possible is done before the start of instrumentation. XIPD displays a working dialog during instrumentation.
<i>Remove Instrumentation</i>	Removes instrumentation from the current selection. If the removal is not successful, an error dialog is displayed. If successful, the context is removed from the <i>Instrumented</i> menu.
<i>Force Buffer Write</i>	Sends an instrument -write command to IPD. You should use this if a program abnormally terminates before writing its performance buffer.
<i>Reset</i>	Undoes all changes made to the dialog's fields.
<i>Done</i>	Dismisses the dialog.
<i>Help</i>	Displays help topic text for the <i>Create Performance Data</i> 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 3-29 shows the 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.

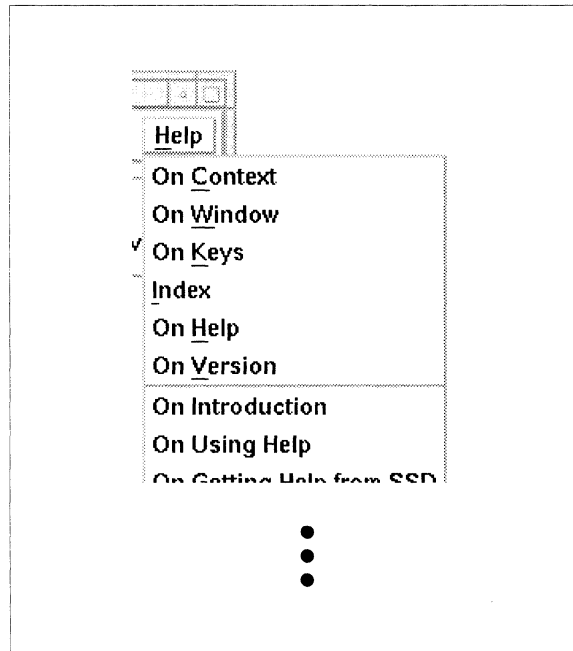


Figure 3-29. Help Menu

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 3-30 shows the help index dialog.

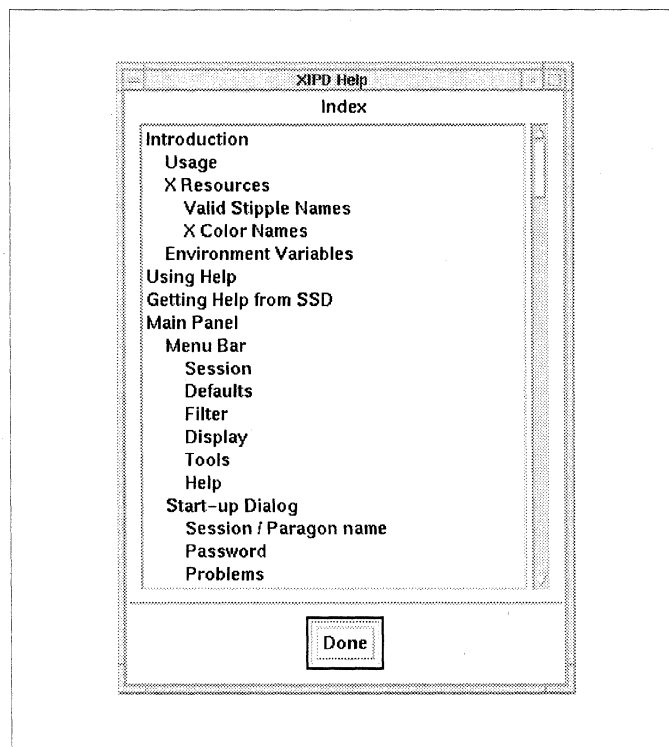


Figure 3-30. Help Index Dialog

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 3-31 shows the help topic dialog.

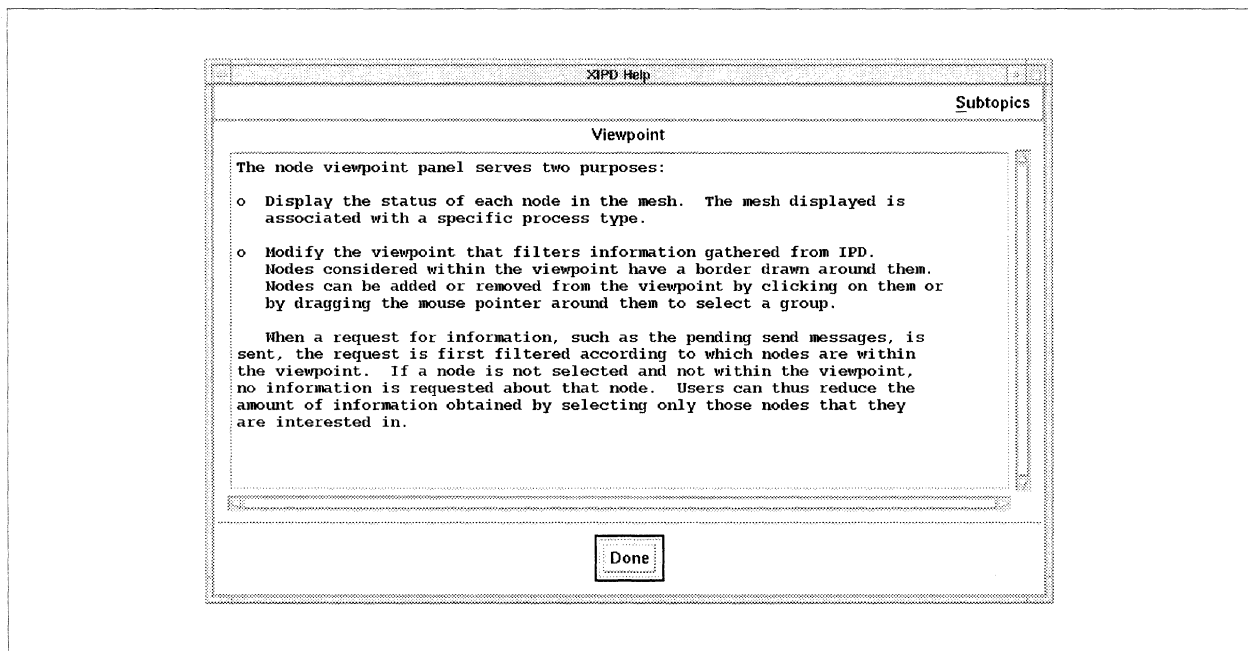


Figure 3-31. Help Topic Dialog

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 3-32 shows the main window legend.

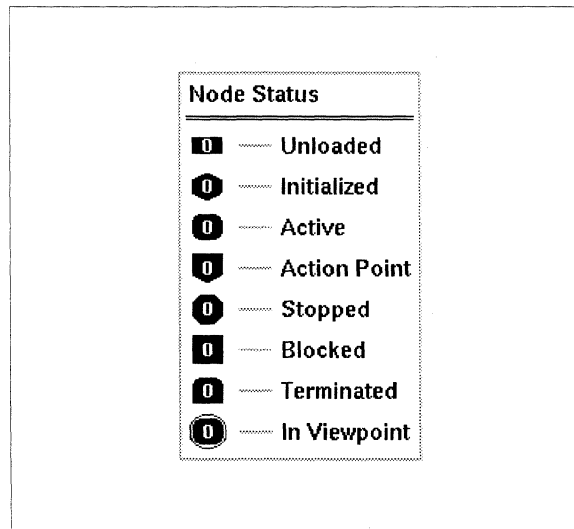


Figure 3-32. Main Window Legend

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:

- | | |
|--------------------|--|
| <i>Unloaded</i> | Nothing is loaded into the node. |
| <i>Initialized</i> | A program is loaded and stopped at its first executable statement, ready to begin execution. |

<i>Active</i>	The node is executing.
<i>Action Point</i>	The node has stopped due to an action point, such as a user-set breakpoint or a data based watchpoint.
<i>Stopped</i>	The node is stopped. This could be due to executing a step command or selecting the <i>Stop Execution</i> button. The node might be stopped at an instruction address.
<i>Blocked</i>	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() .
<i>Terminated</i>	The program has finished execution.
<i>In Viewpoint</i>	The node is part of the viewpoint.

Process Type Selection

For NX applications, the *Ptype* option menu in the main window changes the viewpoint to a different process type. When you select *Ptype*, 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.

Communicator Selection

For MPI applications, the *Communicator* menu in the main window changes the viewpoint to different communicator types once MPI is active. When you select *Communicator*, a menu containing all the communicators is popped up. When you select a communicator type, XIPD updates the viewpoint to reflect the node status for the communicator type.

The pull-down menu contains these areas:

- COMM_WORLD
- COMM n intracommunicator groups
- ICOMM n intercommunicator groups (which may not be present)
- COMMSELF communicators

Each numbered item (COMM1, COMM2, etc.) is sorted within its group. Items are removed when communicators cease to exist.

When a selected communicator ceases to exist, the select goes back to COMM_WORLD. When COMM_WORLD no longer exists (indicating that MPI is no longer active), control reverts to process-type selection, with process type 0 selected.

Nodes displayed for COMM_WORLD or an intracommunicator group are numbered by rank, rather than by node number.

Selecting an ICOMM intercommunicator displays the number of nodes in the intercommunicator, all of which stay selected within the viewpoint.

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 3-33 shows the execution control buttons.

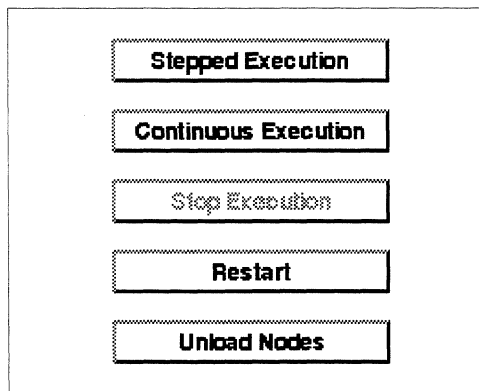


Figure 3-33. Execution Controls

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

The viewpoint filters the 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 3-34 shows how a node appears in and out of viewpoint

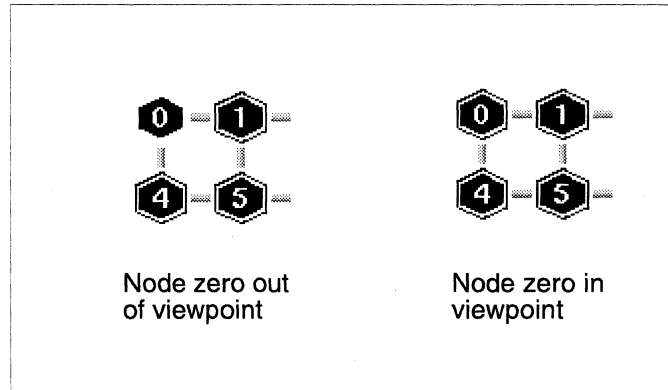


Figure 3-34. Node Viewpoint

Identifying Nodes With Multiple Threads

If *Show Threads* is set, nodes with multiple threads active are displayed with a thick border. Figure 3-34 shows the difference in the way that nodes are displayed.

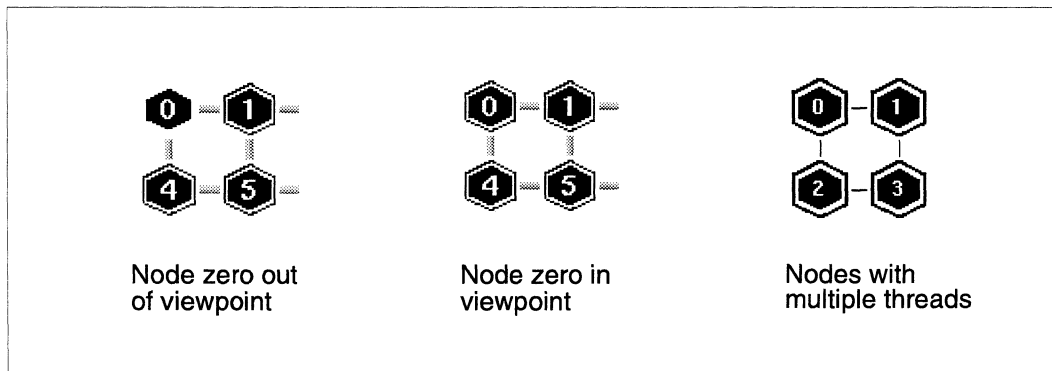


Figure 3-35. Nodes With and Without Multiple Threads

Displaying a Node's Status

You can display a pop-up information panel about an individual node by holding down the third 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. If *Show Threads* is off, the panel includes a notation about other active threads (if any) for the node:ptype, and looks like Figure 3-36. If additional ptypes are active, the next process types's information follows.

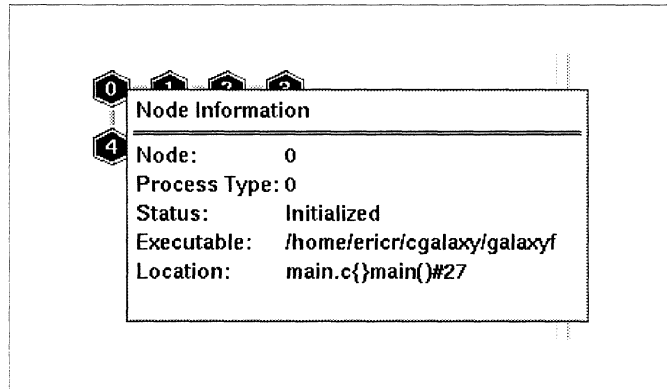


Figure 3-36. Node Information Panel

If *Show Threads* is on, the node information panel includes information about each active thread on the specified node. Figure 3-36 shows a node information panel with *Show Threads* turned on.

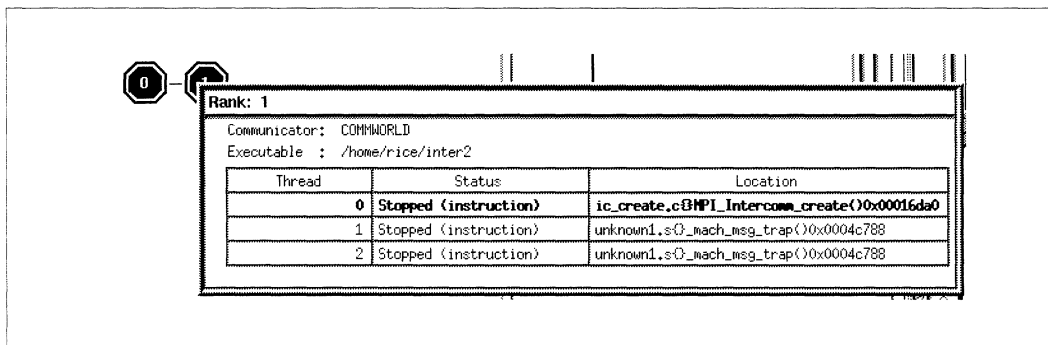


Figure 3-37. Node Information Panel With Show Threads Enabled

Displaying the Source Code a Node is Executing

Clicking on a node with the second 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.

If there are multiple source files, the routines within each source file are displayed indented underneath the file name. If there are multiple executables, the files and routines are listed indented underneath their executable. C++ class methods are listed underneath their class name. Nested classes are displayed within the class they are nested in.

Clicking on a routine will display the source code window for that routine. Clicking on a file, class, or executable name will compress the routine list such that the contents indented underneath the item clicked on will be removed. A ... will be appended to the item, reflecting that if the item is clicked upon again the nested items will reappear.

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. A green check mark appears next to active routines that at least one node is stopped within. A green check mark also appears to any file name, class name, or executable name listed that the active routine belongs to.

A yellow yield sign appears next to routines that have either a break point or trace point set. A yellow yield sign also appears next to any file name, class name, or executable name listed that the routine belongs to.

A grey magnifying glass appears next to routines that have an instrumentation start, stop, or write point set. A magnifying glass also appears next to any file name, class name, or executable name listed that the routine belongs to.

Figure 3-38 shows a routine list.

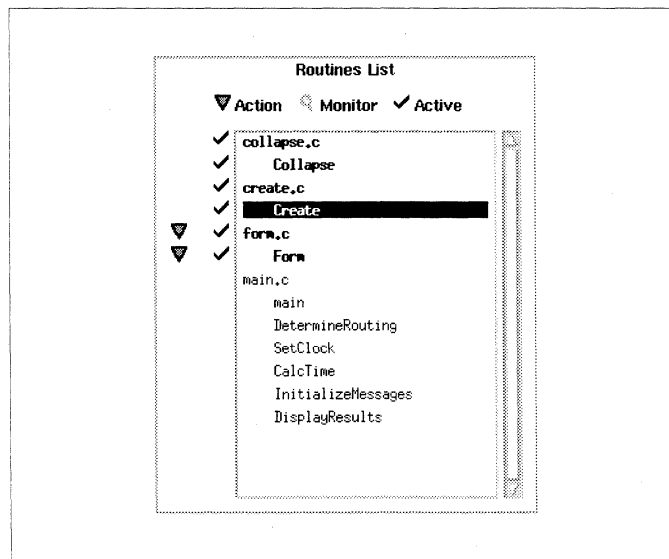


Figure 3-38. Routine List

The routine list in Figure 3-38 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 3-38). 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 3-38 appears as shown in Figure 3-39.

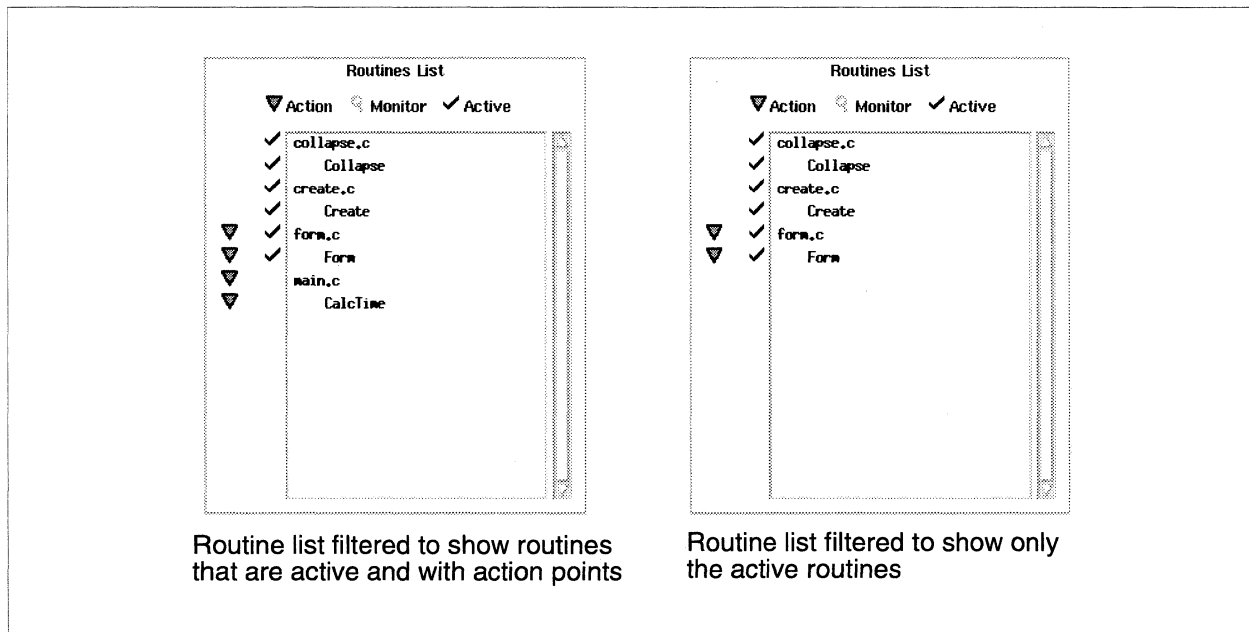


Figure 3-39. Filtered Routine List

Floating the Routine List

The routine list can be “floated” into a separate dialog by pulling down the *Display* menu and selecting *Float Routine List*. Selecting this menu item switches the routine list between floating and fixed, so selecting the menu item when the routine list is floating returns the routine list to the main panel.

Selecting the *Done* button dismisses the routine list and causes the fixed routine list to appear in the main panel.

Selecting the *Help* button displays help text about floating the routine list.

Console Input/Output

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

Program output is added to the scrollable text region as it occurs. You can select this text and paste it into other clients. You can also enter text in this region. When the <RETURN> key is pressed the typed text is sent to the application as input—the text is not used as an IPD command.

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
- Load Application

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 3-40 shows the file selection dialog.

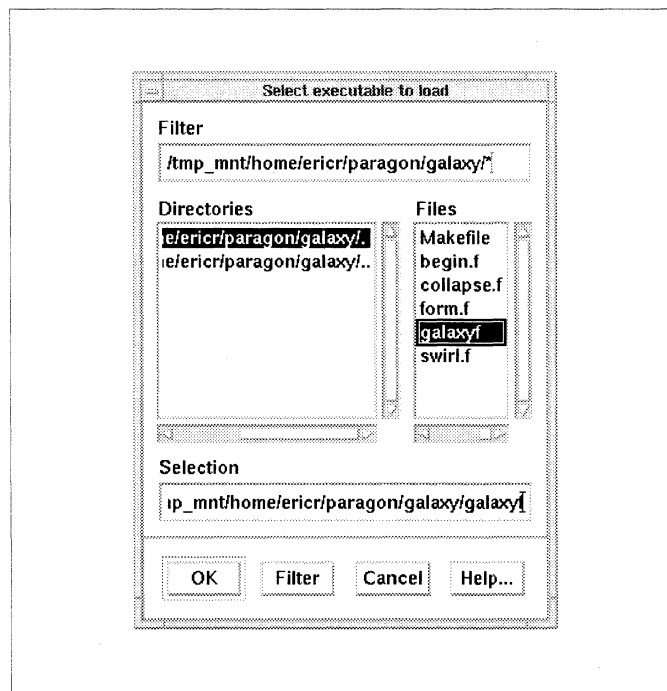


Figure 3-40. File Selection Dialog

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 starts *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

<i>OK</i>	Passes the entry in <i>Selection</i> to the originating dialog and dismisses the file selection dialog.
<i>Filter</i>	Obtains a new file list, and possibly a new directory list.
<i>Cancel</i>	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 3-41 shows the code window.

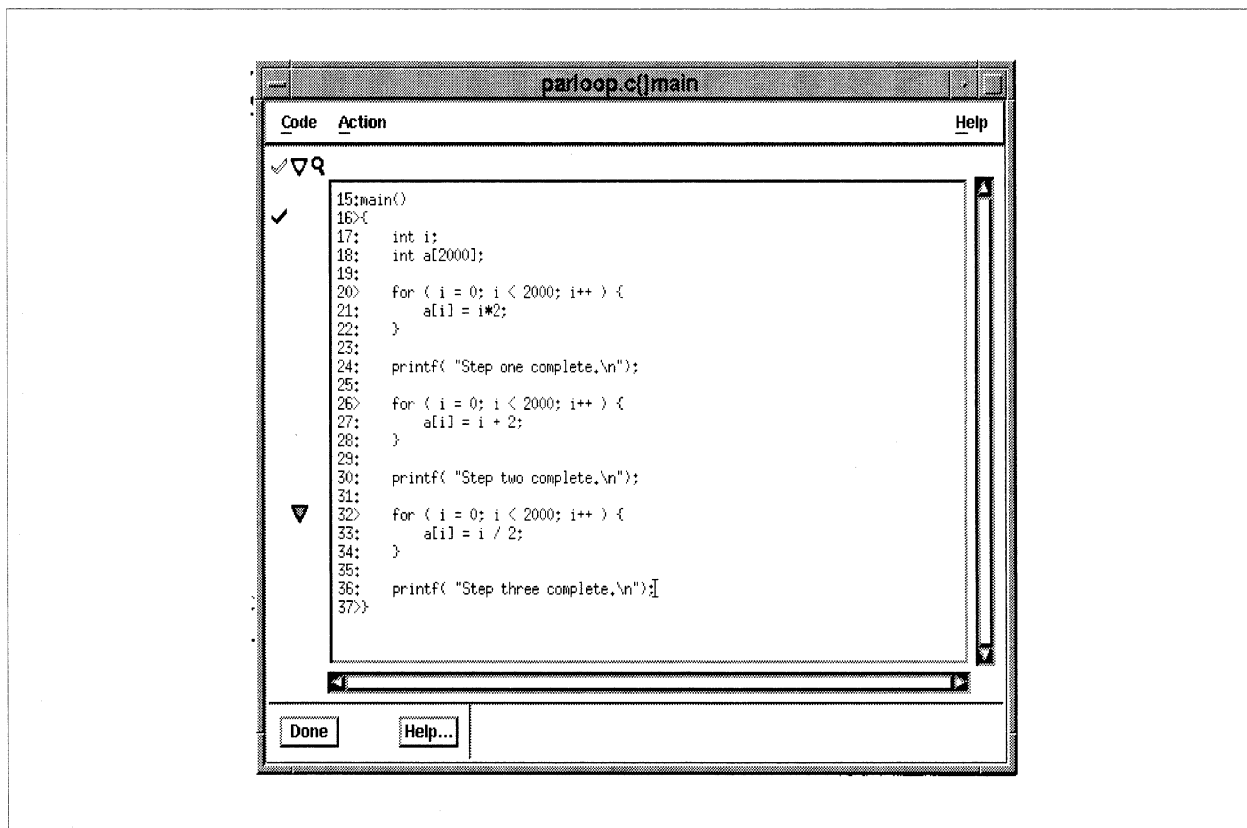


Figure 3-41. Code Window

The contents of the code window change if you start 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 3-42 shows the code menu.

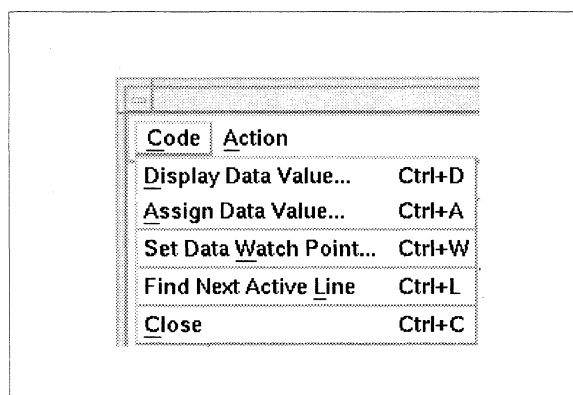


Figure 3-42. Code Menu (NX Applications)

<i>Display Data Value</i>	Displays the data display dialog for the current code window.
<i>Assign Data Value</i>	Displays the data modification dialog for the current code window.
<i>Set Data Watch Point</i>	Displays the data watchpoint dialog for the current routine's program.
<i>Find Active Lines</i>	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 <i>Find Active Lines</i> advance to the next one. If the last executing line is being displayed, the next click on <i>Find Active Lines</i> scrolls back to the first executing line.
<i>Close</i>	Dismisses the code window..

Figure 3-44 shows the code menu that is used with MPI applications. It contains this additional entry:

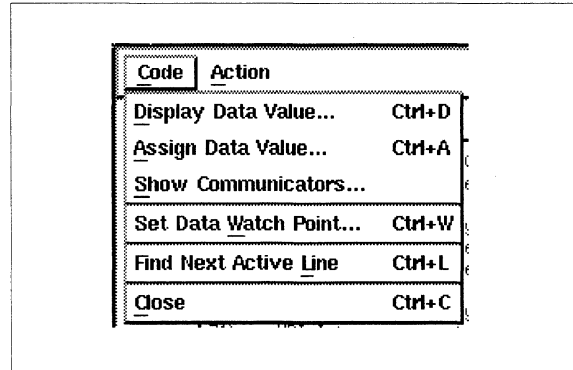


Figure 3-43. Action Menu (MPI Applications)

Show Communicators Opens a dialog that displays the communicators for a communicator variable.

Action Menu

The action menu selects the kind of action point to be set for a line. Figure 3-44 shows the action menu.

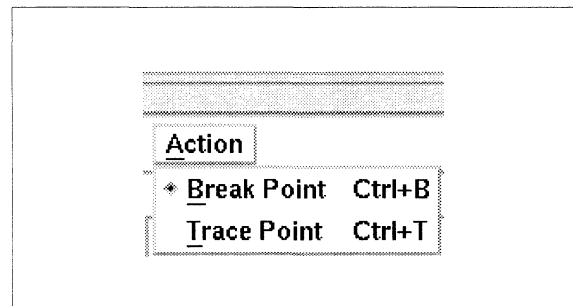


Figure 3-44. Action Menu

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 3-45 shows the 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.

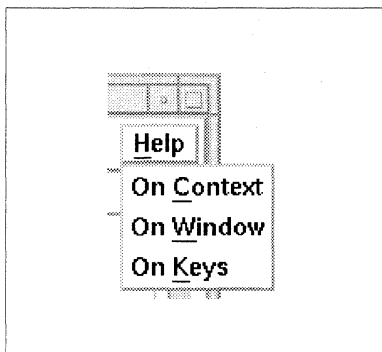


Figure 3-45. Code Window Help Menu

- 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.

Executable Lines

Executable lines are identified with a greater-than symbol following the line number (“>”). Non-executable line are identified with a colon symbol (“:”).

NOTE

Code 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

<i>Done</i>	Dismisses the code window.
<i>Help</i>	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. If *Show Threads* is set, the output includes values for all threads that have the expression active. A data display dialog is associated with a particular code window.

Error messages will be interspersed in the output for threads that do not have the expression active. An error dialog will pop up when an expression cannot be evaluated for any thread on any of the nodes in the selected viewpoint.

Figure 3-46 shows the 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 or expression to be displayed.

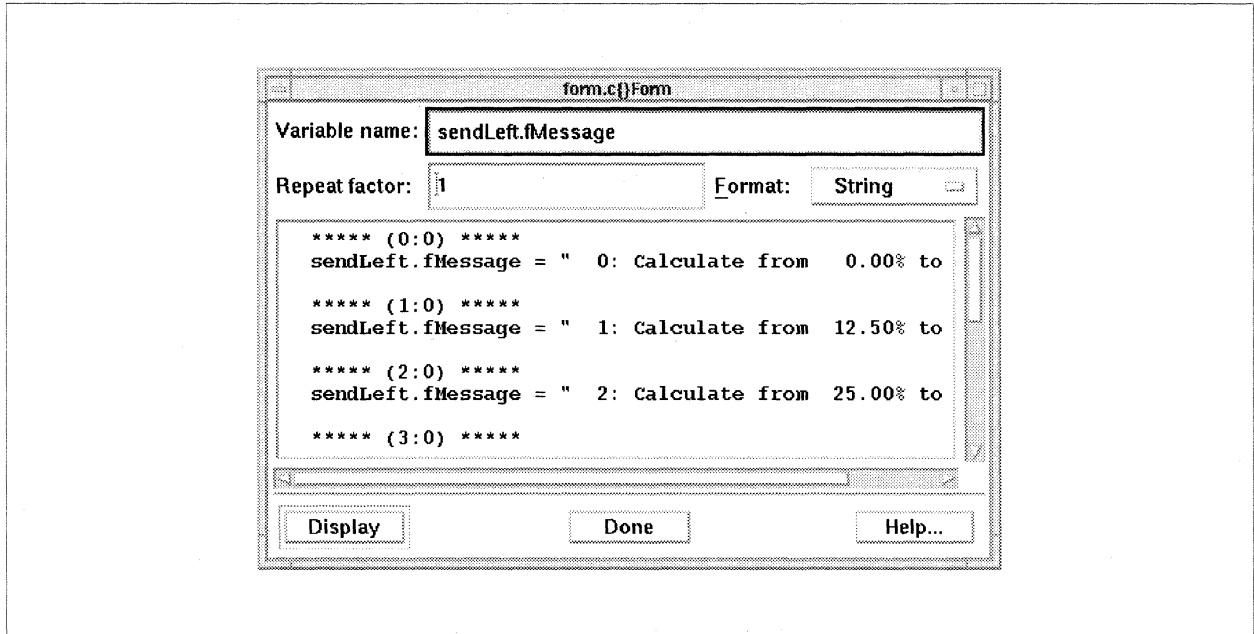


Figure 3-46. Data Display Dialog

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

<i>Display</i>	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.
<i>Done</i>	Dismisses the data display dialog.
<i>Help</i>	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 3-47 shows the data modification dialog.

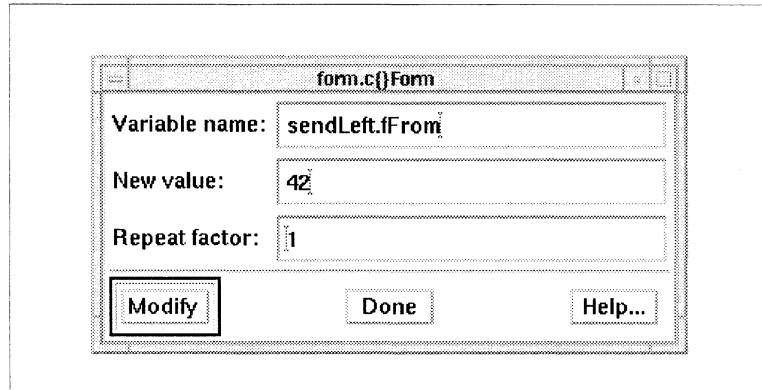


Figure 3-47. Data Modification Dialog

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

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

Show Communicator

The show communicator dialog, shown in Figure 3-44, allows you to display the communicators associated with a specified communicator variable.

Show Communicator For

The name of the communicator variable that you want to show communicators for.

Dialog Buttons

<i>Modify</i>	IPD modifies the given variable to the value specified in <i>New Value</i> . The dialog is disabled until IPD completes modifying the variable.
---------------	---

- Done* Dismisses the show communicator dialog.
- Help* Displays help topic text about the dialog.

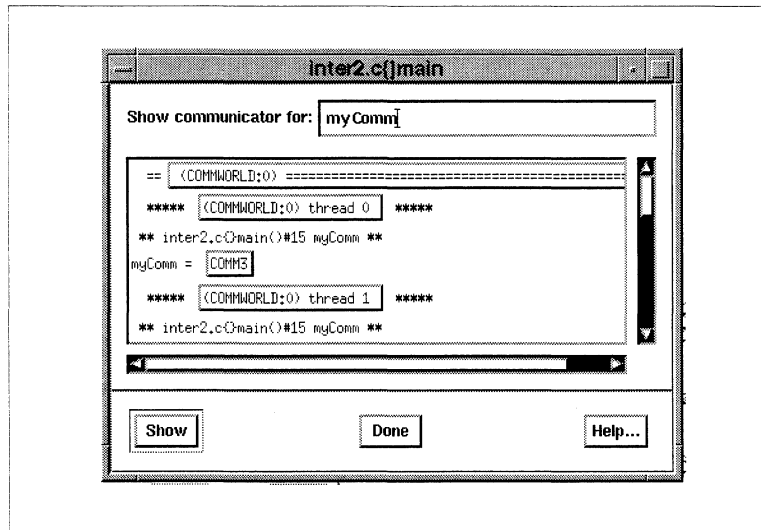


Figure 3-48. Show Communicator Dialog

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 3-49 shows the data watch point dialog.

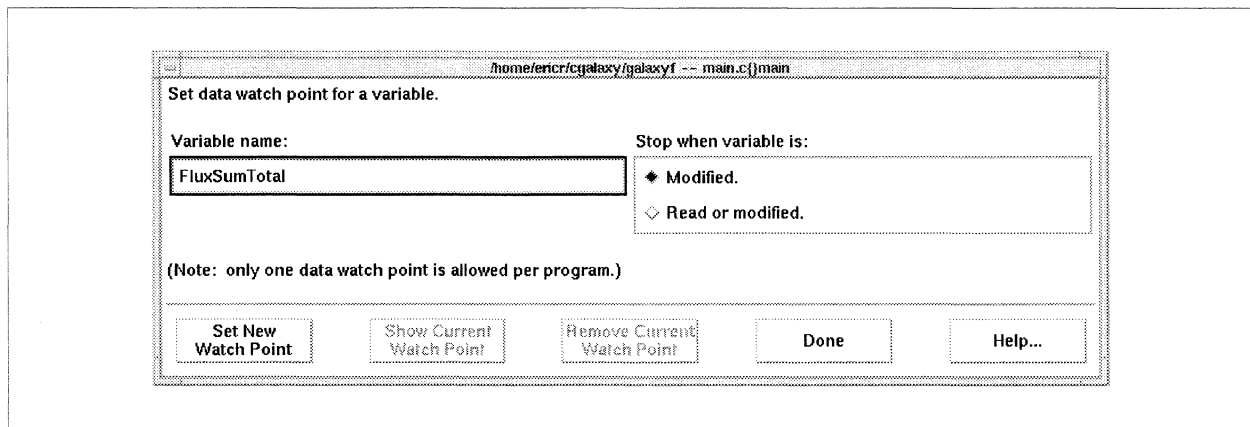


Figure 3-49. Data Watchpoint Dialog

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:

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

Dialog Buttons

<i>Set New Watch Point</i>	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.
<i>Show Current Watch Point</i>	Sets <i>Variable name</i> and <i>Stop when variable is</i> to the current watchpoint settings. This button is disabled if no watchpoint is currently in effect for the program associated with the code window.
<i>Remove Current Watch Point</i>	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.
<i>Done</i>	Dismisses the data watchpoint dialog.
<i>Help</i>	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, you can use the XIPD application resources listed in the following tables to configure XIPD.

Table 3-2 lists the XIPD color resources.

Table 3-2. XIPD Color Resources (1 of 2)

Resource	Purpose
XIpd.appForeground	A color name that defines the default foreground color for interface elements. This resource and <i>XIpd*foreground</i> should be the same.
XIpd.appBackground	A color name that defines the default background color for interface elements. This resource and <i>XIpd*background</i> 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.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 <i>XIpd.normColor</i> and <i>XIpd.darkColor</i> 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 <i>XIpd.brightColor</i> and <i>XIpd.darkColor</i> as well.

Table 3-2. XIPD Color Resources (2 of 2)

Resource	Purpose
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 <i>XIpd.brightColor</i> and <i>XIpd.normColor</i> 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 <i>Preferences</i> 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 <i>Preferences</i> dialog sets this the same color as unloaded status.
XIpd.activeForeground	A color name that defines the color used for active code icons. The <i>Preferences</i> dialog sets this the same color as active status.

Table 3-3 lists the XIPD pattern resources.

Table 3-3. XIPD Pattern Resources

Resource	Purpose
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.

Table 3-4 lists the XIPD font resources.

Table 3-4. XIPD Font Resources

Resource	Purpose
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. <i>This should be a fixed width font.</i>
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. <i>This should be a fixed width font.</i>
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.

Table 3-5 lists the XIPD size resources.

Table 3-5. XIPD Size Resources

Resource	Purpose
XIpd.nodeHeight	An integer that defines the height, in pixels, that XIPD draws nodes (this and <i>XIpd.nodeWidth</i> should be set to the same value).
XIpd.nodeWidth	An integer that defines the width, in pixels, that XIPD draws nodes (this and <i>XIpd.nodeHeight</i> should be set to the same value).
XIpd.geometry	A geometry string that controls the initial size and placement of XIPD.

Table 3-6 lists other XIPD resources.

Table 3-6. Other XIPD Resources

Resource	Purpose
XIpd.showSymbols	A boolean (True / False) value. If this resource is <i>True</i> , XIPD uses symbols as well as color to display the status of nodes in the viewpoint panel. If <i>False</i> , XIPD uses filled circles.
XIpd.doDebug	A boolean (True / False) value. If this resource is <i>True</i> , debugging elements of XIPD are present.
XIpd.doProfile	A boolean (True / False) value. If this resource is <i>True</i> , performance analysis elements of XIPD are present.
XIpd.doRemoteLogin	A boolean (True / False) value. If this resource is <i>True</i> , the XIPD startup dialog is always displayed.
XIpd.doCoreAnalysis	A boolean (True / False) value. If this resource is <i>True</i> , XIPD displays the <i>Core Analysis</i> dialog first, instead of the <i>Load Application</i> dialog.
XIpd.coreDirectoryName	A character string that defines the name of the default core directory.
XIpd.coreSelect	A character string that can be one of the following: Fault When a core directory is successfully scanned, all files from faulting processes are selected. This is the default. Non-Fault When a core directory is successfully scanned, all files from non-faulting processes are selected. All When a core directory is successfully scanned, all files are selected.
XIpd.partitionName	A character string that defines the default partition name for the <i>Load Application</i> dialog.
XIpd.applicationName	A character string that defines the default application name for the <i>Load Application</i> dialog.

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.
lineHoriz	Horizontal lines.
lineVert	Vertical lines.

Default Configuration

Table 3-7 lists the default XIPD resource settings.

Table 3-7. Default XIPD Resource Settings (1 of 3)

Resource	Purpose
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)

Table 3-7. Default XIPD Resource Settings (2 of 3)

Resource	Purpose
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	<i>Null-String</i> (color / grayscale) "dotsNorm" (monochrome)
XIpd.initializedStatusStipple	<i>Null-String</i> (color / grayscale) "dotsVeryDark" (monochrome)
XIpd.activeStatusStipple	<i>Null-String</i> (color / grayscale) "lineDiagLeft" (monochrome)
XIpd.breakpointStatusStipple	<i>Null-String</i> (color / grayscale) "lineHatch" (monochrome)
XIpd.stoppedStatusStipple	<i>Null-String</i> (color / grayscale / monochrome)
XIpd.blockedStatusStipple	<i>Null-String</i> (color / grayscale / monochrome)
XIpd.terminatedStatusStipple	<i>Null-String</i> (color / grayscale) "lineHoriz" (monochrome)
XIpd.generalFontName	variable
XIpd.codeWindowFontName	fixed

Table 3-7. Default XIPD Resource Settings (3 of 3)

Resource	Purpose
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
XIpd.doDebug	True
XIpd.doProfile	True
XIpd.doRemoteLogin	False
XIpd.partitionName	Empty String
XIpd.applicationName	Empty String

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:

<i>NX_DFLT_PART</i>	Name of the default partition that XIPD selects automatically within the mesh configuration dialog. If not defined (or invalid), XIPD selects the <i>.compute</i> partition.
<i>XAPPLRESDIR</i>	XIPD looks in this directory for the resource file <i>XIpd</i> when you modify XIPD resource settings through the <i>Preferences</i> dialog. If not defined, XIPD looks in your home directory.
<i>XIPDHOME</i>	Alternative home directory for XIPD session files. If not defined, XIPD stores the session files in your home directory.
<i>CORE_FILE_PATH</i>	The directory path that XIPD will search for core dump directories.



Program Profiling: prof and gprof

4

This chapter describes **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™ System 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.

If an application has been compiled with the **-Mconcur** switch and is running on a node with multiple arithmetic processors (an MP node), the profile data will include enhanced data that describes performance on individual user threads. The display will show an additional field (%Parallel) that contains the sum of thread execution times. The enhanced data may be suppressed with the **-u** command line switch in **prof**.

If a system contains both GP and MP nodes, **prof** displays information about GP nodes and MP nodes separately.

Routines from the *libpm* library are now displayed under a single entry titled "overhead."

The underbar that's added to routine names by the linker is now stripped off in the **prof** output.

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

address_option can be one of the following

- o** Displays each symbol address in octal.
- x** Displays each symbol address in hexadecimal.

display_options can be any of the following:

- g** Includes non-global symbols (static functions).
- z** Includes all symbols in the profile range, even if associated with zero number of calls and zero time.
- h** Suppresses the heading normally displayed on the report. This is useful if the report is to be processed further.
- s** Displays a summary of several of the monitoring parameters and statistics on the standard error output.
- u** Suppresses the display of enhanced performance data (the %Parallel field) for applications running on nodes with two or more arithmetic CPUs.

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

-m profile_file Uses 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

executable_name is the name of the executable file, *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 Correlates 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

Standard 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
0.0 0.00 0.01 1 0.0 _memcpy
0.0 0.00 0.01 6 0.0 _main
0.0 0.00 0.01 6 0.0 _NCisshift
0.0 0.00 0.01 6 0.0 _NLchrlen
0.0 0.00 0.01 1 0.0 __doprnt
0.0 0.00 0.01 1 0.0 _exit
0.0 0.00 0.01 1 0.0 __xflsbuf
0.0 0.00 0.01 1 0.0 _fwrite
0.0 0.00 0.01 1 0.0 _memchr
0.0 0.00 0.01 1 0.0 _printf
0.0 0.00 0.01 2 0.0 _profil
0.0 0.00 0.01 1 0.0 _write
```

If more than one routine name is associated with the same piece of code, **prof** prints out the additional routine names on the same line, as shown in the following example:

```
0.0    0.01    25.69    3204    0.003    ___mth_i_dsin    [_sin]
```

Both `_sin` and `___mth_i_dsin` represent the same piece of code, so **prof** prints both names. All matches are listed after the first name, within brackets and separated by commas (if there is more than one name in brackets).

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.

The special name “<overhead>” is assigned to routines that reside within the *libpm* library. All information about “overhead” library routines is condensed into one entry.

Function names are now shown without the underbar that the linker adds.

Enhanced (Multi-Thread) Output

The profile data for MP nodes, which have multiple arithmetic processors, reports execution times for all user threads. The enhanced display contains the following additional field:

%Parallel	The percentage of time that a function is executed by a thread that is running in parallel with another thread on a different CPU. This field is only displayed if the application is executed on a node with two or more arithmetic CPUs.
------------------	--

The following example shows the result of profiling enhanced data:

```

paragon> prof hello
Executable: /home/rice/hello
prof data: /home/rice/mon.out/hello.1048654.0.0
%Time Seconds Cumsecs #Calls msec/call %Parallel Name
0.0 0.00 0.00 11 0.0 0 pthread_self
0.0 0.00 0.00 11 0.0 0 spin_lock
0.0 0.00 0.00 11 0.0 0 stack_self
0.0 0.00 0.00 11 0.0 0 vp_self
0.0 0.00 0.00 1 0. 0 pthread_mutex_lock
0.0 0.00 0.00 1 0. 0 pthread_mutex_unlock
0.0 0.00 0.00 1 0. 0 exit
0.0 0.00 0.00 1 0. 0 _xflsbuf
0.0 0.00 0.00 1 0. 0 printf
0.0 0.00 0.00 2 0. 0 _mprofil
0.0 0.00 0.00 4 0. 0 rec_mutex_lock
0.0 0.00 0.00 4 0. 0 rec_mutex_unlock
0.0 0.00 0.00 1 0. 0 _write
0.0 0.00 0.00 1 0. 0 _doprnt
0.0 0.00 0.00 2 0. 0 ferror
0.0 0.00 0.00 1 0. 0 fileno
0.0 0.00 0.00 1 0. 0 unlocked_fwrite
0.0 0.00 0.00 1 0. 0 _memchr
0.0 0.00 0.00 6 0. 0 NCisshift
0.0 0.00 0.00 6 0. 0 NLchrlen

```

The times for all threads are summed and used as a denominator when computing percent of total execution time. The percentages always total 100%.

Overhead Routines

The following example shows how routines from the *libpm* library are combined into a single entry:

```
Executable: /home/ericr/fft/fft2d
prof data: /home/ericr/fft/mon.out/fft2d.1048580.0.0
```

%Time	Seconds	Cumsecs	#Calls	msec/call	Name
23.2	5.54	5.54	1484800	0.0037	_hypot
21.5	5.13	10.67	93381	0.0549	_ccopy_
13.4	3.19	13.86			fft2d_860
9.8	2.34	16.20			<overhead>
6.0	1.44	17.64	1272842	0.0011	___mth_i_dsqrt [_sqrt]
6.0	1.42	19.06	1485411	0.0010	___mth_i_qidiv
5.9	1.40	20.46	7696	0.182	_kcfftb_pi_
5.3	1.27	21.73	7696	0.165	_kcfftb_pf_
1.5	0.36	22.09			_flick
1.5	0.35	22.44			___mth_i_cabs
0.8	0.20	22.64			_csend
0.8	0.18	22.82			___builtin_va_arg
0.7	0.16	22.98	3204	0.050	___mth_i_dsin [_sin]
0.5	0.12	23.10	16	7.5	_kcfftd_pxf_

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.

All routines that reside in the *libpm* library are regarded as “overhead”. In the call graph, all such routines are listed as <overhead>, along with an index number into the index table. In the flat profile, all overhead routines are combined into one line, with multiple index entries that identify the routines that were included. In the symbol index table, overhead routines are enclosed in angle brackets “<>”.

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 used, 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 the **-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*** Computes the normal call-graph profile but excludes *function_name* when printing the profile. Suppresses printing the graph profile entry for routine *function_name*'s descendants, unless they have other ancestors that are not suppressed. More than one **-e** options can be given. Only one *function_name* may be given with each option.
- E *function_name*** Assigns a zero run-time to *function_name* (thus excluding the time spent in *function_name* and its descendants from the total and percentage time computations). Suppresses printing the graph profile entry for *function_name*. The fact that the function name was excluded from the profile is indicated by printing the index of the function in round rather than square brackets. More than one **-E** option may 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, you would specify **-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 with the following form:

executable_name.pid.node.ptype

executable_name is the name of the executable file, *pid* is the process id, *node* is the node number as given in the *INFO* file, and *ptype* is the process type. 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 specified, 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 example 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

index	%time	self	descendents	called/total	parents	index
				called+self called/total	name children	
[1]	0.0	0.00	0.00	11/11	stack_self	[4]
		0.00	0.00	11	get_stack_pointer	[1]

[2]	0.0	0.00	0.00	1/11	pthread_mutex_unlock	[17]
		0.00	0.00	2/11	pthread_mutex_lock	[16]
		0.00	0.00	4/11	rec_mutex_lock	[8]
		0.00	0.00	4/11	rec_mutex_unlock	[9]
		0.00	0.00	11	pthread_self	[2]
		0.00	0.00	11/11	vp_self	[5]

[3]	0.0	0.00	0.00	11/11	11	vp_self [5] spin_lock [3]
[4]	0.0	0.00	0.00	11/11	11	vp_self [5] stack_self [4] get_stack_pointer [1]
[5]	0.0	0.00	0.00	11/11	11	pthread_self [2] vp_self [5] spin_lock [3] stack_self [4]
[6]	0.0	0.00	0.00	6/6	6	NLchrln [7] NCisshift [6]
[7]	0.0	0.00	0.00	6/6	6	_doprnt [685] NLchrln [7] NCisshift [6]
[8]	0.0	0.00	0.00	1/4	4	printf [13] fileno [12] ferror [10] rec_mutex_lock [8] pthread_self [2] pthread_libmutex_lock [14]
[9]	0.0	0.00	0.00	1/4	4	printf [13] fileno [12] ferror [10] rec_mutex_unlock [9] pthread_self [2] pthread_libmutex_unlock [15]

		0.00	0.00	1/2	printf [13]
		0.00	0.00	1/2	_doprnt [685]
[10]	0.0	0.00	0.00	2	ferror [10]
		0.00	0.00	2/4	rec_mutex_lock [8]
		0.00	0.00	2/4	rec_mutex_unlock [9]
		0.00	0.00	1/1	_crt0_start [734]
[11]	0.0	0.00	0.00	1	exit [11]
		0.00	0.00	1/1	_xflsbuf [688]
[12]	0.0	0.00	0.00	1	fileno [12]
		0.00	0.00	1/4	rec_mutex_lock [8]
		0.00	0.00	1/4	rec_mutex_unlock [9]
		0.00	0.00	1/1	main [307]
[13]	0.0	0.00	0.00	1	printf [13]
		0.00	0.00	1/4	rec_mutex_lock [8]
		0.00	0.00	1/1	_doprnt [685]
		0.00	0.00	1/2	ferror [10]
		0.00	0.00	1/4	rec_mutex_unlock [9]
		0.00	0.00	1/1	rec_mutex_lock [8]
[14]	0.0	0.00	0.00	1	pthread_libmutex_lock [14]
		0.00	0.00	1/1	pthread_mutex_lock [16]
		0.00	0.00	1/1	rec_mutex_unlock [9]
[15]	0.0	0.00	0.00	1	pthread_libmutex_unlock [15]
		0.00	0.00	1/1	pthread_mutex_unlock [17]
		0.00	0.00	1/1	pthread_libmutex_lock [14]
[16]	0.0	0.00	0.00	1	pthread_mutex_lock [16]
		0.00	0.00	2/11	pthread_self [2]

[17]	0.0	0.00	0.00	1/1	pthread_libmutex_unlock [15]
		0.00	0.00	1	pthread_mutex_unlock [17]
		0.00	0.00	1/11	pthread_self [2]

[18]	0.0	0.00	0.00	1/1	_doprnt [685]
		0.00	0.00	1	unlocked_fwrite [18]
		0.00	0.00	1/1	_memchr [686]
		0.00	0.00	1/1	_xflsbuf [688]

[685]	0.0	0.00	0.00	1/1	printf [13]
		0.00	0.00	1	_doprnt [685]
		0.00	0.00	6/6	NLchrln [7]
		0.00	0.00	1/1	unlocked_fwrite [18]
		0.00	0.00	1/2	ferror [10]

[686]	0.0	0.00	0.00	1/1	unlocked_fwrite [18]
		0.00	0.00	1	_memchr [686]

[687]	0.0	0.00	0.00	1/1	_xflsbuf [688]
		0.00	0.00	1	_write [687]

[688]	0.0	0.00	0.00	1/1	unlocked_fwrite [18]
		0.00	0.00	1	_xflsbuf [688]
		0.00	0.00	1/1	fileno [12]
		0.00	0.00	1/1	_write [687]

flat profile:

% time	the percentage of the total running time of the program used by this function.
cumulative seconds	a running sum of the number of seconds accounted for by this function and those listed above it.
self seconds	the number of seconds accounted for by this 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 ms/call	the average number of milliseconds spent in this function per call, if this function is profiled, else blank.
total ms/call	the average number of milliseconds spent in this 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

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
0.0	0.00	0.00	11	0.00	0.00	get_stack_pointer [1]
0.0	0.00	0.00	11	0.00	0.00	pthread_self [2]
0.0	0.00	0.00	11	0.00	0.00	spin_lock [3]
0.0	0.00	0.00	11	0.00	0.00	stack_self [4]
0.0	0.00	0.00	11	0.00	0.00	vp_self [5]
0.0	0.00	0.00	6	0.00	0.00	NCisshift [6]
0.0	0.00	0.00	6	0.00	0.00	NLchrlen [7]
0.0	0.00	0.00	4	0.00	0.00	rec_mutex_lock [8]
0.0	0.00	0.00	4	0.00	0.00	rec_mutex_unlock [9]
0.0	0.00	0.00	2	0.00	0.00	ferror [10]
0.0	0.00	0.00	1	0.00	0.00	_doprnt [685]
0.0	0.00	0.00	1	0.00	0.00	_memchr [686]
0.0	0.00	0.00	1	0.00	0.00	_write [687]
0.0	0.00	0.00	1	0.00	0.00	_xflsbuf [688]
0.0	0.00	0.00	1	0.00	0.00	exit [11]
0.0	0.00	0.00	1	0.00	0.00	fileno [12]
0.0	0.00	0.00	1	0.00	0.00	printf [13]
0.0	0.00	0.00	1	0.00	0.00	pthread_libmutex_lock [14]
0.0	0.00	0.00	1	0.00	0.00	pthread_libmutex_unlock [15]
0.0	0.00	0.00	1	0.00	0.00	pthread_mutex_lock [16]
0.0	0.00	0.00	1	0.00	0.00	pthread_mutex_unlock [17]
0.0	0.00	0.00	1	0.00	0.00	unlocked_fwrite [18]

Index by function name

[6] NCisshift	[12] fileno	[8] rec_mutex_lock
[7] NLchrlen	[1] get_stack_pointer	[9] rec_mutex_unlock
[685] _doprnt	[13] printf	[3] spin_lock
[686] _memchr	[14] pthread_libmutex_lo	[4] stack_self
[687] _write	[15] pthread_libmutex_un	[18] unlocked_fwrite
[688] _xflsbuf	[16] pthread_mutex_lock	[5] vp_self
[11] exit	[17] pthread_mutex_unloc	
[10] ferror	[2] pthread_self	

If more than one routine name is associated with the same piece of code, **gprof** prints out the additional routine names in a table at the end of the output, as shown in the following example:

6 functions have additional matching symbol(s):

```
[379] __ieee_fp_div      [__mth_i_ddiv]
      [75] __mth_i_alog   [_logf]
      [381] __mth_i_dcos  [_cos]
      [70] __mth_i_dsin  [_sin]
      [7]  __mth_i_dsqrt [_sqrt]
      [44] dzopy_loop101 [dzopy_loop10]
```

If an application uses routines in the *libpm* library, those routines are combined into a single "overhead" entry in the call graph, as shown in the following example entries:

```
[4]      9.4      4.19      0.00      <overhead> [4]
-----
                                     <spontaneous>
[5]      6.9      3.08      0.00      <overhead> [5]
```

In the flat profile, the overhead routines from the *libpm* library are combined into a single entry with multiple index references, one index reference for each routine that's included. The following sample is an example:

% time	cumulative seconds	self seconds	self calls	self ms/call	total ms/call	name
51.2	22.71	22.71				<overhead> [1, 2, 4, 5]
8.9	26.67	3.96				fft2d_860 [3]
6.1	29.36	2.69				ccopy_loop100 [7]
4.7	31.43	2.07				case2 [6]
3.6	33.01	1.58				dcopy_loop10 [9]
3.3	34.49	1.48				x_gt [8]
3.0	35.83	1.34				LAB1 [10]

The last example shows a routine index that contains routines from the *libpm* library. It shows how those entries are identified with angle brackets.

Index by function name

```
[2] <__PMA                > [33] _memcpy          [153] fr_read_record
[1] <_record_arc          > [892] _memset         [154] fr_readnum
[4] <__PMTS                > [84]  _msgdone         [176] free
[5] <__PMVE                > [882] _msgmerge       [100] ftn_allocate
```



This chapter describes XProf, a graphical front end to **prof**. For a description of **prof**, refer to Chapter 4, *Program Profiling: prof and gprof*. Using **prof** from the command line can be complicated in the Paragon™ system 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 system 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” on page 5-5.

Invoking XProf

To invoke XProf on the Paragon system do the following:

1. Enter the following command on your workstation:

```
% xhost + paragon_system
```

where *paragon_system* is the name of the Paragon system on which you are going to run XProf.

2. Log onto the Paragon system.
3. Set the *DISPLAY* environment variable to your workstation as in the following example:

```
% setenv DISPLAY machine_name : 0
```

where *machine_name* is the name of your workstation.

4. Check to be sure you have */usr/bin/X11* in your search path.
5. Invoke 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:

- l** Include local (static-declared) symbols. This corresponds to the **prof -g** option.
- 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. The default for *profdir* is *mon.out*.

X Toolkit parameters

The standard parameters supported by the X Toolkit (refer to the *X Toolkit Intrinsic 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 with 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 Runtime 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" on page 5-19.

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.

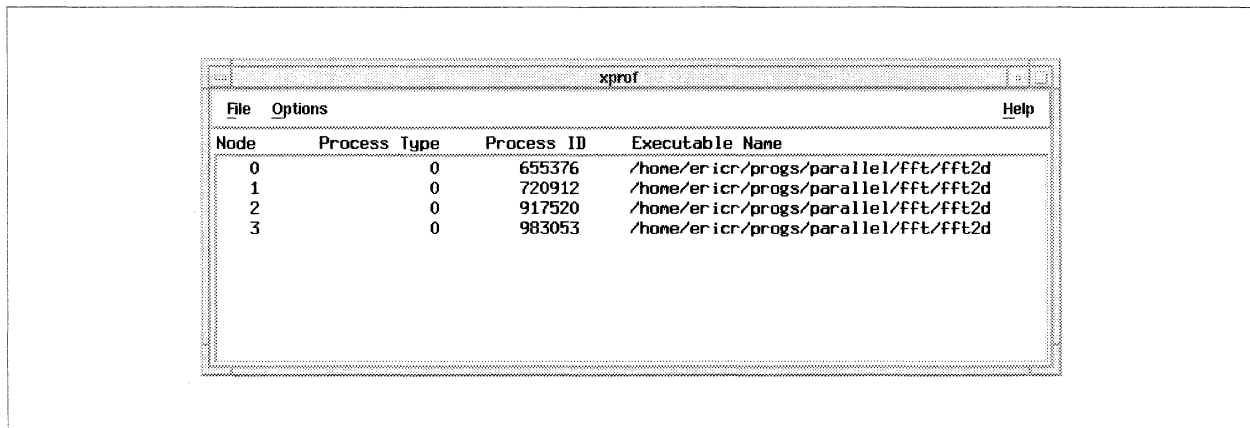


Figure 5-1. XProf Main Window

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

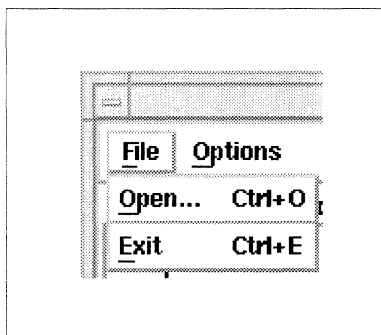


Figure 5-2. File Menu

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 <Ctrl-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 <Ctrl-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.

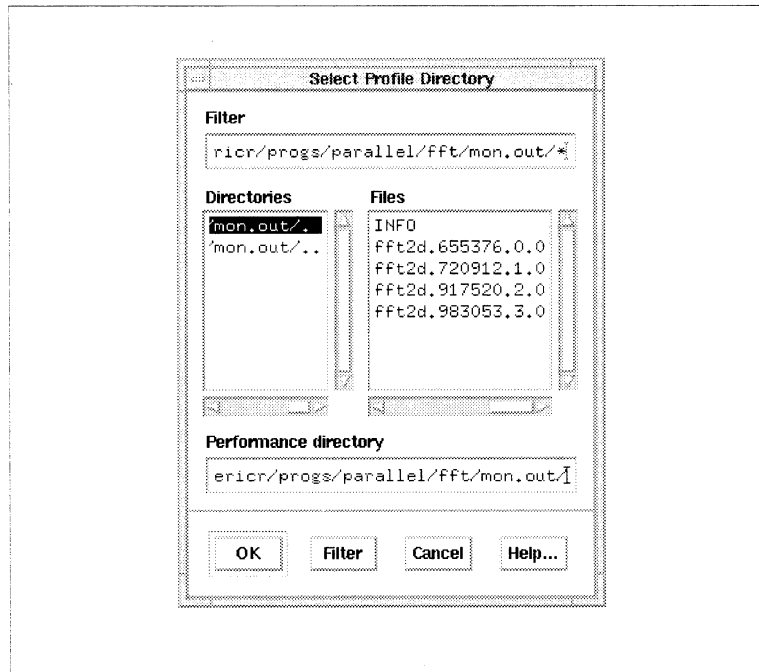


Figure 5-3. Select Profile Directory Dialog

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 an asterisk (*) 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* 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.

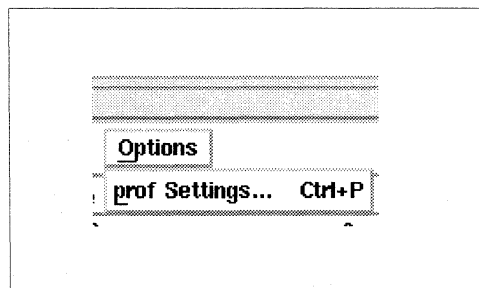


Figure 5-4. Options Menu

prof Settings

This selection displays the *Enter prof Settings* dialog. Changes to this dialog affect subsequent **prof** generated reports. The default keyboard accelerator for this item is <Ctrl-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.

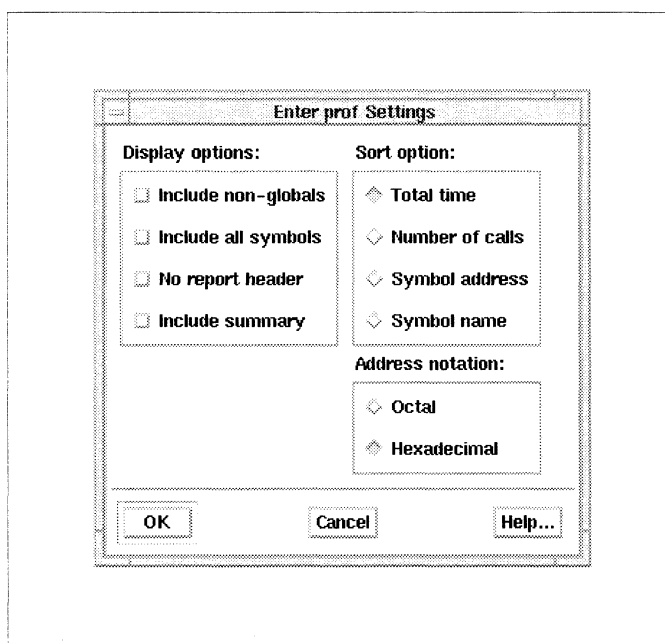


Figure 5-5. Enter prof Settings Dialog

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" on page 5-2) or with entries in the XProf resource file. The XProf resource file is described in the section "Configuring XProf" on page 5-19.

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:

<i>Include non-globals</i>	Include local (static declared) routines in the prof output.
<i>Include all symbols</i>	Include all symbols, including symbols with zero-use, in the prof output.
<i>No report header</i>	Omit the prof output header.
<i>Include summary</i>	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:

<i>Total time</i>	Sort by decreasing amount of time spent in a routine.
<i>Number of calls</i>	Sort by decreasing number of calls to a routine.
<i>Symbol address</i>	Sort by ascending symbol address.
<i>Symbol name</i>	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:

<i>Octal</i>	Print addresses in base eight.
<i>Hexadecimal</i>	Print addresses in base sixteen.

Dialog Buttons

<i>OK</i>	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 <i>XProf</i> resource file (refer to the section "Configuring XProf" on page 5-19).
<i>Cancel</i>	Dismisses the <i>Enter prof Settings</i> dialog. Any changes to the dialog are discarded.
<i>Help</i>	Displays help text for the <i>Enter prof Settings</i> 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.

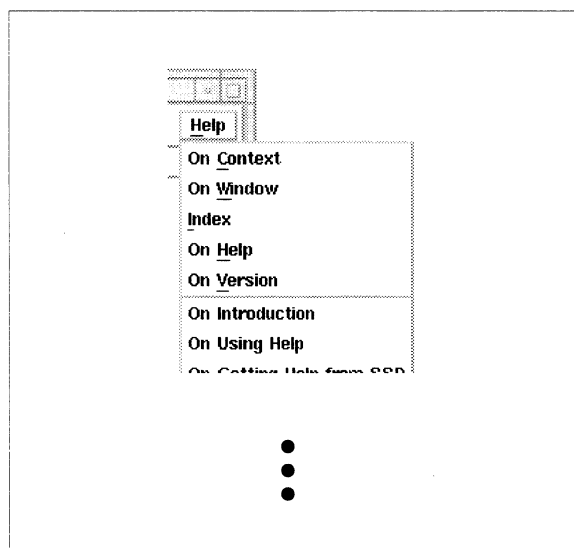


Figure 5-6. Help Menu

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. The *Index* dialog is described in the section "Help Index" on page 5-12.

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.

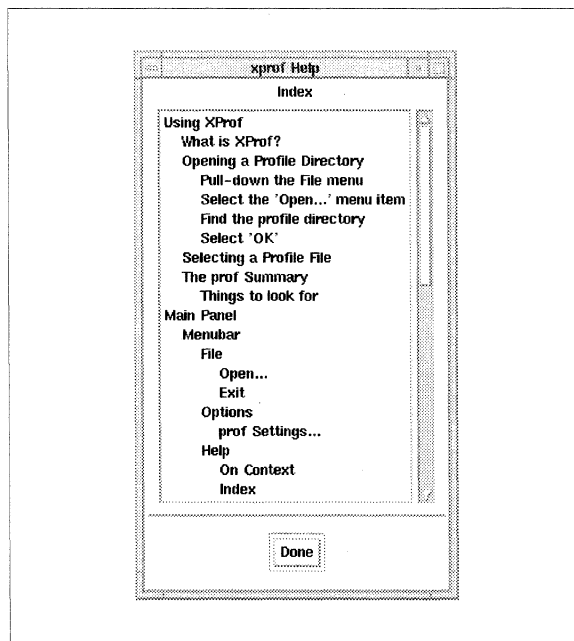


Figure 5-7. Help Index Dialog

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 *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.

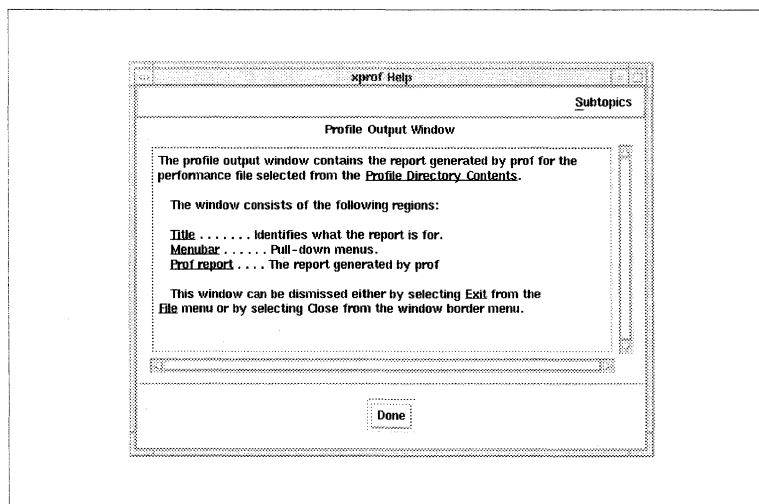


Figure 5-8. Help Topic Dialog

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.

The screenshot shows a window titled 'xprof' containing a table with the following data:

File	Options				Help
Node	Process	Type	Process ID	Executable Name	
0		0	655376	/hone/ericr/progs/parallel/fft/fft2d	
1		0	720912	/hone/ericr/progs/parallel/fft/fft2d	
2		0	917520	/hone/ericr/progs/parallel/fft/fft2d	
3		0	983053	/hone/ericr/progs/parallel/fft/fft2d	

Figure 5-9. Profile Output File List

For each entry, the following information is shown:

- 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 *Enter 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 shows a **prof** output window.

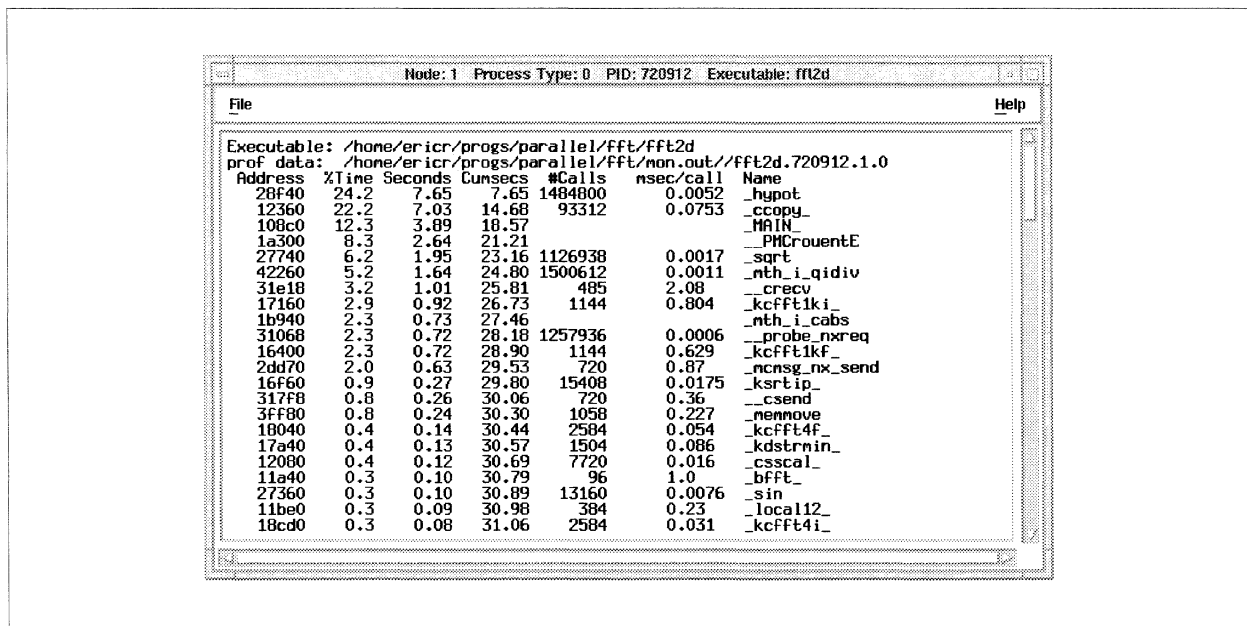


Figure 5-10. Output Window

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 ID, and executable name for the file.

Output Window File Menu

The output window *File* menu contains items for saving the **prof** output text, for reexecuting **prof**, and for removing the window from the display. Figure 5-11 shows the output window *File* menu.

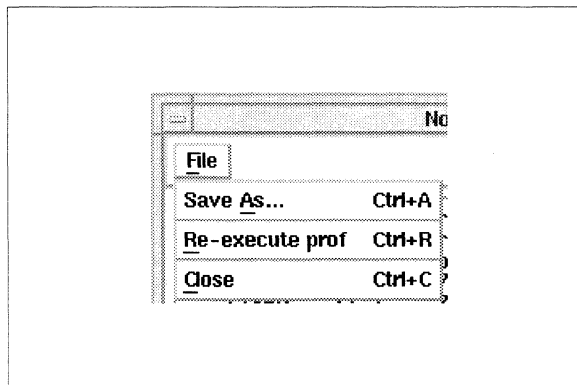


Figure 5-11. Output Window File Menu

Save As

Save As displays the *Save prof Output* dialog. This allows you to save the output of **prof** into a text file for later use. The default keyboard accelerator for *Save As* is **<Ctrl-A>**. The *Save prof Output* dialog is described in the section "Save prof Output Dialog" on page 5-17.

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 **<Ctrl-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 **<Ctrl-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.

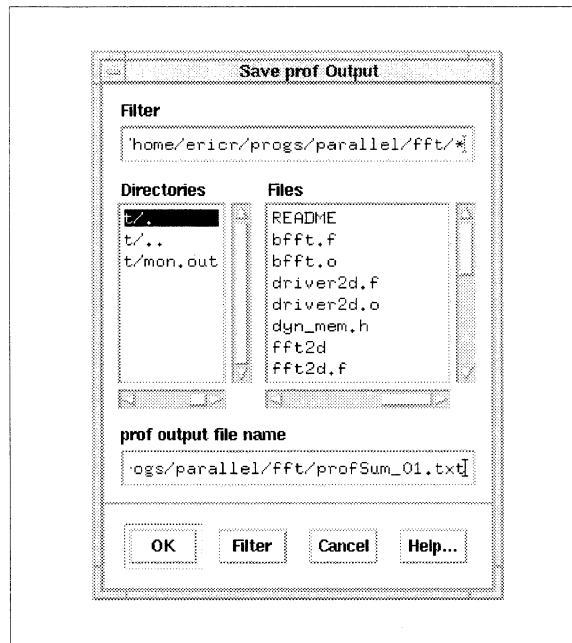


Figure 5-12. Save prof Output Dialog

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

<i>OK</i>	XProf checks to see if the entry in the <i>prof output file name</i> 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.
<i>Filter</i>	Displays a new file list (and possibly a new directory list) based on the entry in the <i>Filter</i> field.
<i>Cancel</i>	Dismisses the <i>Save prof Output</i> dialog.
<i>Help</i>	Displays help topic text about using the <i>Save prof Output</i> dialog.

Output Window 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.

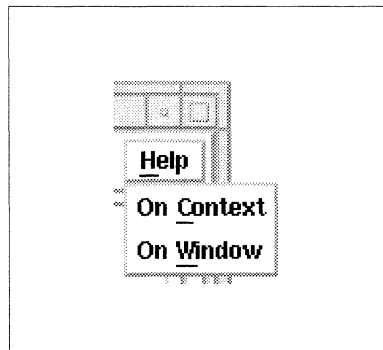


Figure 5-13. Output Window Help Menu

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, you can use the XProf application resources listed in Table 5-1 to configure XProf.

Table 5-1. XProf Application Resources (1 of 2)

Resource	Purpose
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 <i>Include non-globals</i> display option in the settings dialog.
XProf.includeAllSymbols	A boolean (True / False) value that chooses the <i>Include all symbols</i> display option in the settings dialog.
XProf.noReportHeader	A boolean (True / False) value that chooses the <i>No report header</i> display option in the settings dialog.
XProf.includeSummary	A boolean (True / False) value that chooses the <i>Include summary</i> display option in the settings dialog.
XProf.sortByTime	A boolean (True / False) value that chooses the <i>Total time</i> sort option in the settings dialog.
XProf.sortByCall	A boolean (True / False) value that chooses the <i>Number of calls</i> sort option in the settings dialog.
XProf.sortByAddress	A boolean (True / False) value that chooses the <i>Symbol address</i> sort option in the settings dialog.

Table 5-1. XProf Application Resources (2 of 2)

Resource	Purpose
XProf.sortByName	A boolean (True / False) value that chooses the <i>Symbol name</i> sort option in the settings dialog.
XProf.addressOctal	A boolean (True / False) value that chooses the <i>Octal</i> address option in the settings dialog.
XProf.addressHex	A boolean (True / False) value that chooses the <i>Hexadecimal</i> 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

Table 5-2 lists the default XProf resource settings.

Table 5-2. Default XProf Resource Settings

Resource	Default Setting
XProf.listRows	10
XProf.infoPathName	<i>mon.out</i>
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

This chapter describes XGprof, a graphical front end to **gprof**. For a description of **gprof**, refer to Chapter 4, *Program Profiling: prof and gprof*. Using **gprof** from the command line can be complicated in the Paragon™ system 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 system 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” on page 6-4.

Invoking XGprof

To invoke XGprof on the Paragon system do the following:

1. Enter the following command on your workstation:

```
% xhost + paragon_system
```

where *paragon_system* is the name of the Paragon system on which you are going to run XGprof.

2. Log onto the Paragon system.
3. Set the *DISPLAY* environment variable to your workstation as in the following example:

```
% setenv DISPLAY machine_name : 0
```

where *machine_name* is the name of your workstation.

4. Check to be sure you have */usr/bin/X11* in your search path.
5. Invoke 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. The default for *gprofdir* is *mon.out*.

X Toolkit parameters

The standard parameters supported by the X Toolkit (see command-line options in the *X Toolkit Intrinsic 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 with 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 Runtime 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” on page 6-19.

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.

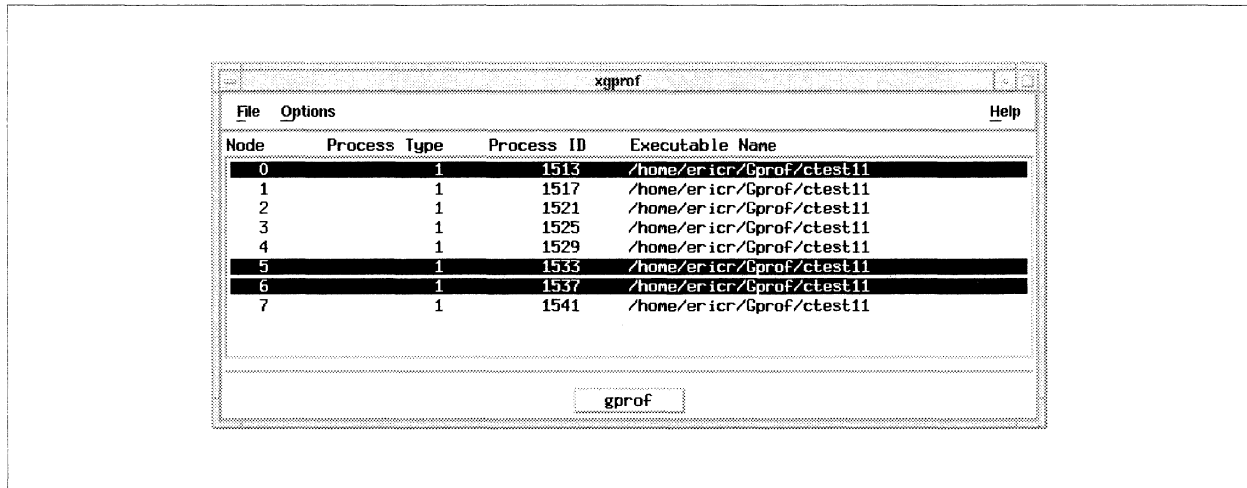


Figure 6-1. Main Window

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.

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.

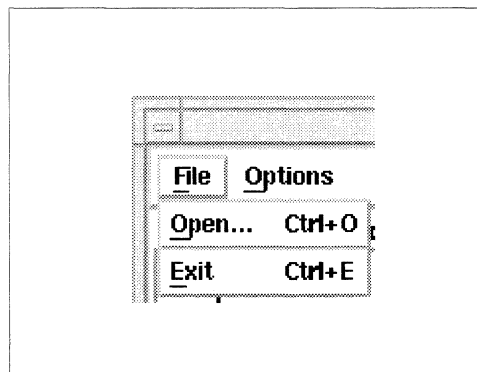


Figure 6-2. File Menu

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 **<Ctrl-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 **<Ctrl-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.

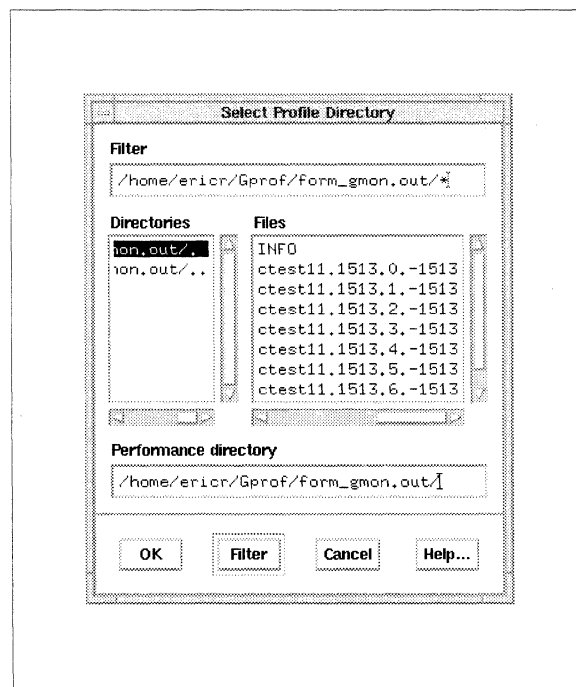


Figure 6-3. Select Profile Directory Dialog

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 an asterisk (*) 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

<i>OK</i>	When you select <i>OK</i> , XGprof checks the entry in the <i>Performance directory</i> 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.
<i>Filter</i>	XGprof uses the entry in the <i>Filter</i> field to obtain a new file list and a new directory list if the filter has been changed to a different directory.
<i>Cancel</i>	When selected, XGprof dismisses the dialog.
<i>Help</i>	Displays help topic text about using the <i>Select Profile Directory</i> 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.

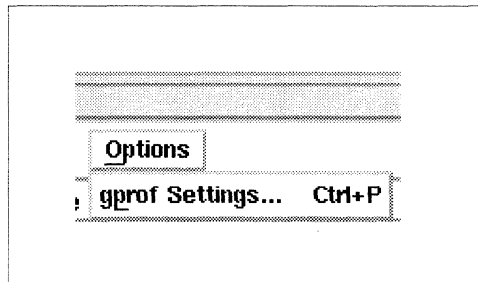


Figure 6-4. Options Menu

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 <Ctrl-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” on page 6-2) or with entries in the *XGprof* resource file. The *XGprof* resource file is described in the section “Configuring XGprof” on page 6-19.

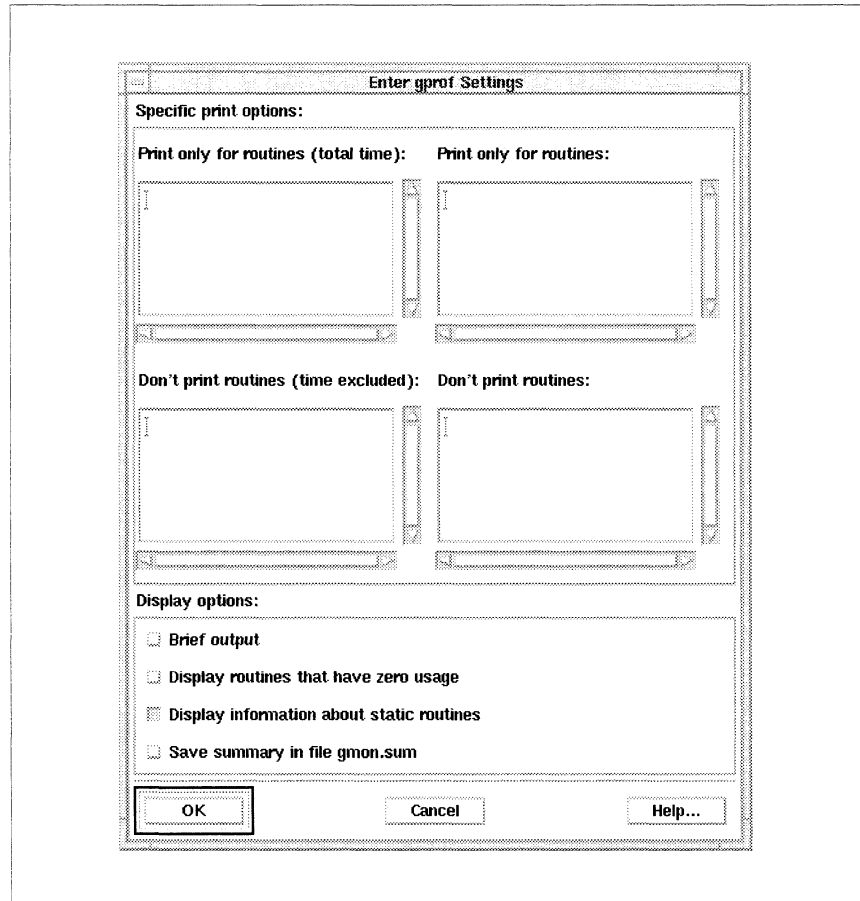


Figure 6-5. Enter gprof Settings Dialog

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

<i>Brief output</i>	Suppresses display of descriptions of each field in the profile.
<i>Display routines that have zero usage</i>	Displays routines which have zero call counts and zero accumulated time.
<i>Display information about static routines</i>	Displays information about statically-declared functions.
<i>Save summary in file gmon.sum</i>	Produces a <i>gmon.sum</i> 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

<i>OK</i>	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 <i>XGprof</i> resource file (refer to the section "Configuring XGprof" on page 6-19).
<i>Cancel</i>	Dismisses the <i>Enter gprof Settings</i> dialog. Any changes to the dialog are discarded.
<i>Help</i>	Displays help text for the <i>Enter gprof Settings</i> 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.

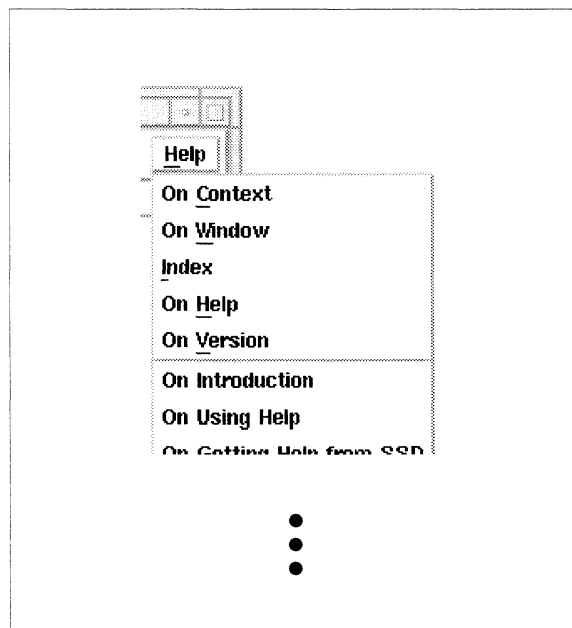


Figure 6-6. Help Menu

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. The *Index* dialog is described in the section “Help Index” on page 6-12.

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.

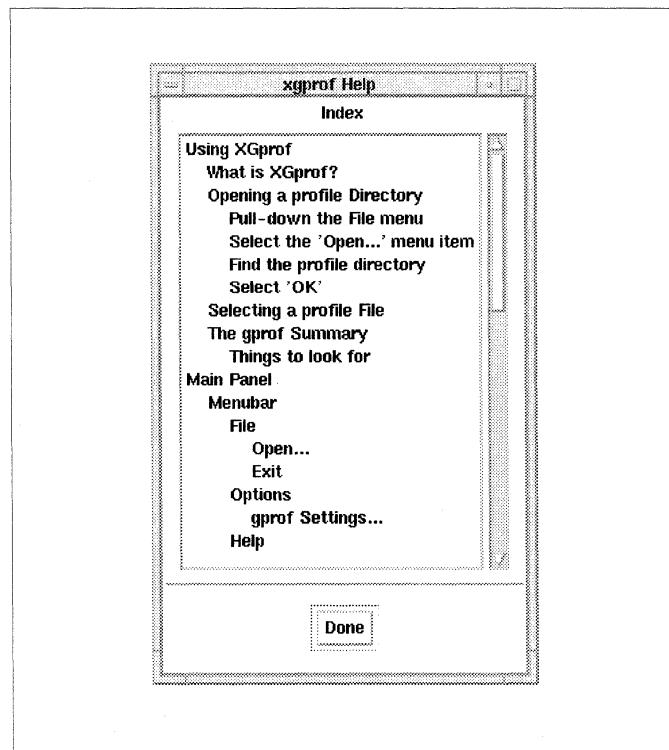


Figure 6-7. Help Index Dialog

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 *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.

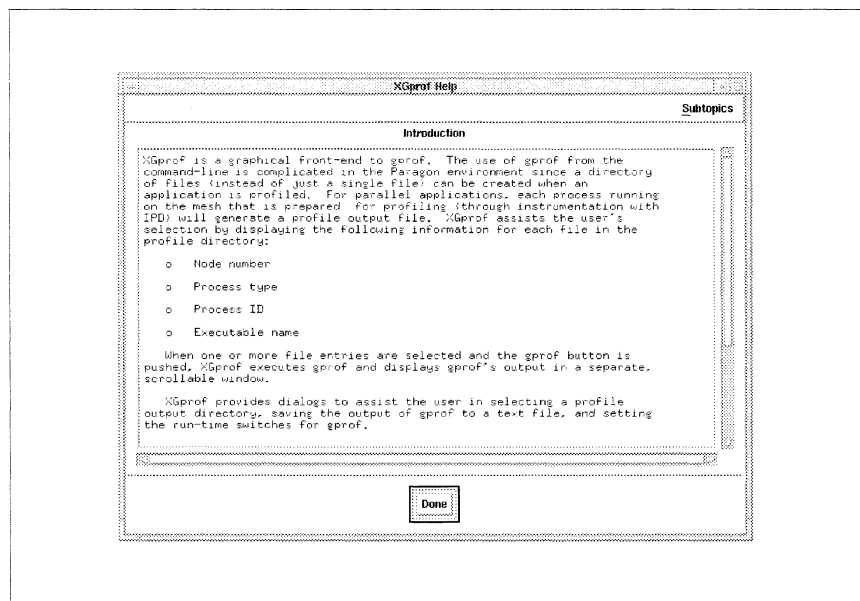


Figure 6-8. Help Topic Dialog

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/eric/Gprof/ctest11
1	1	1517	/home/eric/Gprof/ctest11
2	1	1521	/home/eric/Gprof/ctest11
3	1	1525	/home/eric/Gprof/ctest11
4	1	1529	/home/eric/Gprof/ctest11
5	1	1533	/home/eric/Gprof/ctest11
6	1	1537	/home/eric/Gprof/ctest11
7	1	1541	/home/eric/Gprof/ctest11

Figure 6-9. Profile Output File List

For each entry, the following information is shown:

- 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 *Enter 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 shows a **gprof** output window.

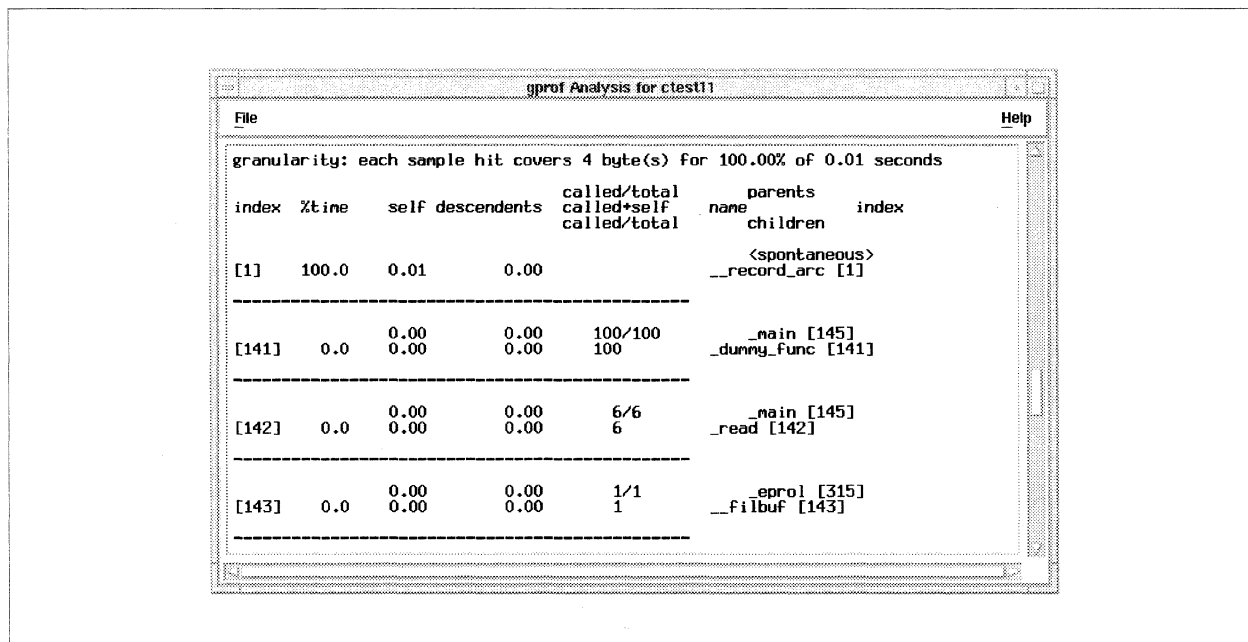


Figure 6-10. Output Window

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.

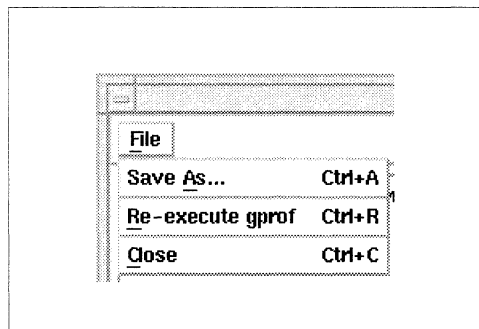


Figure 6-11. Output Window File Menu

Save As

Save As displays the *Save gprof Output* dialog. This allows you to save the output of **gprof** into a text file for later use. The default keyboard accelerator for *Save As* is **<Ctrl-A>**. The *Save gprof Output* dialog is described in the section "Save gprof Output Dialog" on page 6-17.

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 **<Ctrl-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 **<Ctrl-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.

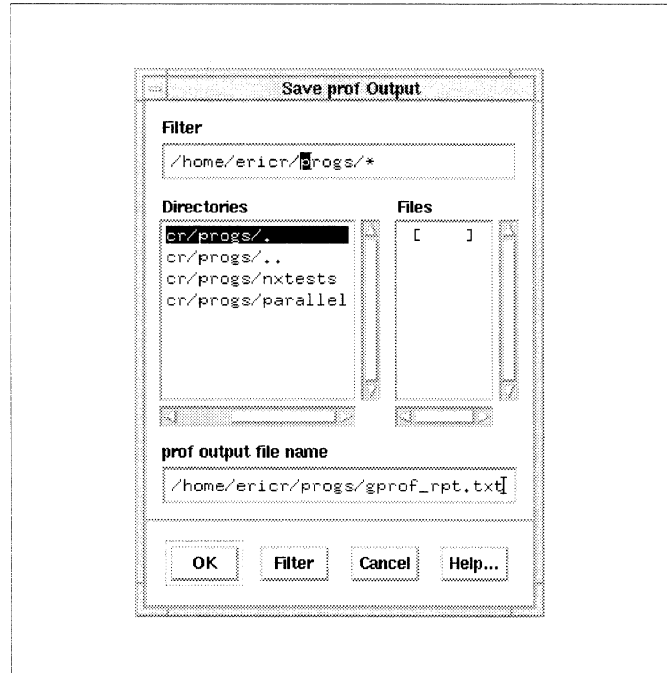


Figure 6-12. Save gprof Output Dialog

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

<i>OK</i>	XGprof checks to see if the entry in the <i>prof output file name</i> 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.
<i>Filter</i>	Displays a new file list (and possibly a new directory list) based on the entry in the <i>Filter</i> field.
<i>Cancel</i>	Dismisses the <i>Save gprof Output</i> dialog.
<i>Help</i>	Displays help topic text about using the <i>Save gprof Output</i> dialog.

Output Window 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.

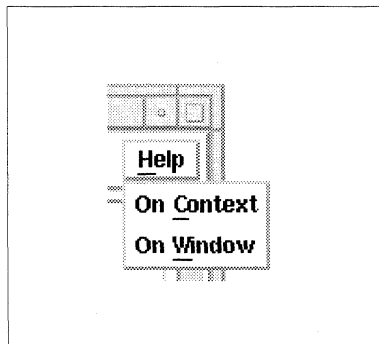


Figure 6-13. Output Window Help Menu

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, you can use the XGprof application resources listed in Table 6-1 to configure XGprof.

Table 6-1. XGprof Application Resources

Resource	Purpose
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 <i>Brief output</i> display option in the settings dialog.
XGprof.zeroUsage	A boolean (True / False) value that chooses the <i>Display routines that have zero usage</i> display option in the settings dialog.
XGprof.staticRoutines	A boolean (True / False) value that chooses the <i>Display information about static routines</i> display option in the settings dialog.
XGprof.saveSummary	A boolean (True / False) value that chooses the <i>Save summary if file gmon.sum</i> 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

Table 6-2 lists the default XGprof resource settings

Table 6-2. Default XGprof Resource Settings

Resource	Default Setting
XGprof.listRows	10
XGprof.infoPathName	<i>gmon.out</i>
XGprof.briefOutput	False
XGprof.zeroUsage	False
XGprof.staticRoutines	True
XGprof.saveSummary	False

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 system 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™ System Interactive Parallel Debugger Reference Manual*

Overview

ParaGraph is a graphical display system for visualizing the behavior and performance of Paragon system 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 on the Paragon system do the following:

1. Enter the following command on your workstation:

```
% xhost +paragon_system
```

where *paragon_system* is the name of the Paragon system on which you are going to run ParaGraph.

2. Log onto the Paragon system.
3. Set the *DISPLAY* environment variable to your workstation as in the following example:

```
% setenv DISPLAY machine_name : 0
```

where *machine_name* is the name of your workstation.

4. Check to be sure you have */usr/bin/X11* in your search path.
5. Invoke 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.

<code>-p</code>	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.
<code>-f filename</code>	Specifies the name of a trace file that contains previously-saved performance data in the Paragon SDDF trace format.
<code>-e environment_file</code>	Specifies the name of a file containing a layout environment produced through the Save Layout command.
<i>X Toolkit parameters</i>	The standard parameters supported by the X Toolkit (refer to the <i>X Toolkit Intrinsic Programming Manual</i>).

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 & Commands" on page 7-6.

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.

<i>idle</i>	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.
<i>overhead</i>	The processor is executing in the communication subsystem.
<i>I/O</i>	The processor is executing I/O statements.
<i>busy</i>	The processor is executing some portion of the program other than the communication or I/O subsystem.
<i>flush</i>	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 "Restrictions" on page 7-58.

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.

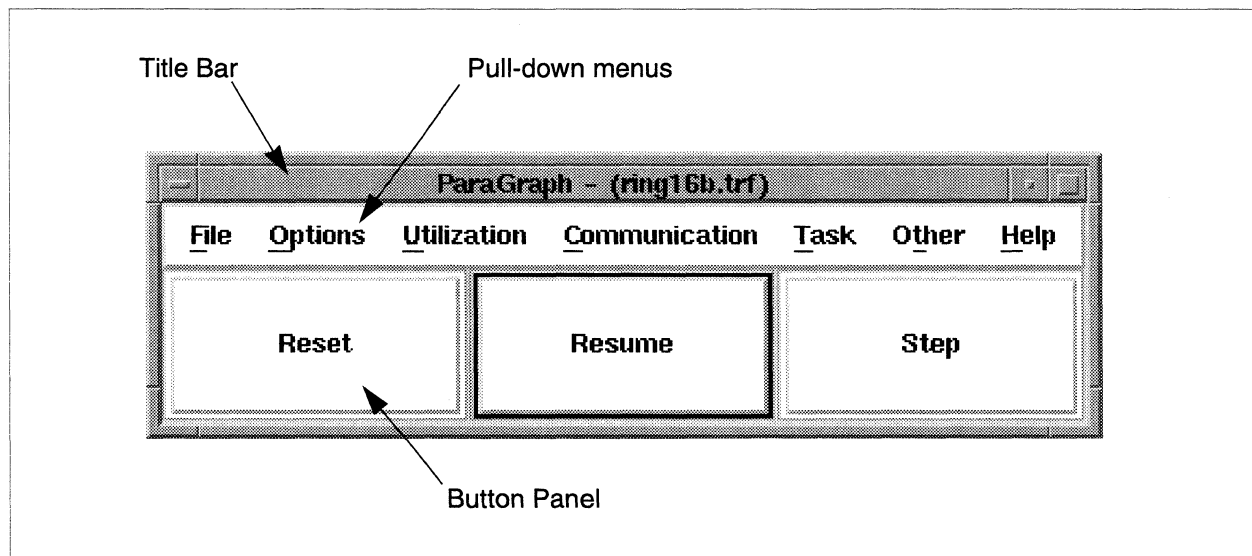


Figure 7-1. ParaGraph Main Window

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.

<i>Start</i>	Starts the visualization process. When you push the <i>Start</i> button, the button label changes to <i>Pause</i> . You can stop the visualization by pressing <i>Pause</i> , which changes the button label to <i>Resume</i> .
<i>Reset</i>	Returns to the beginning of the trace file.
<i>Step</i>	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 selections 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.

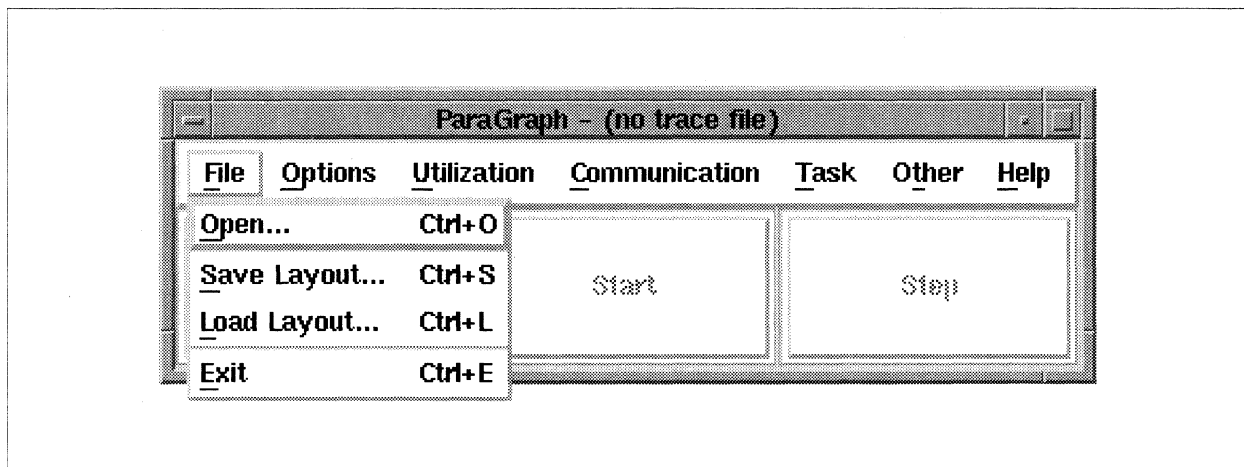


Figure 7-2. File Menu

Open

Open 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 <Ctrl-O>. Figure 7-3 shows the *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.

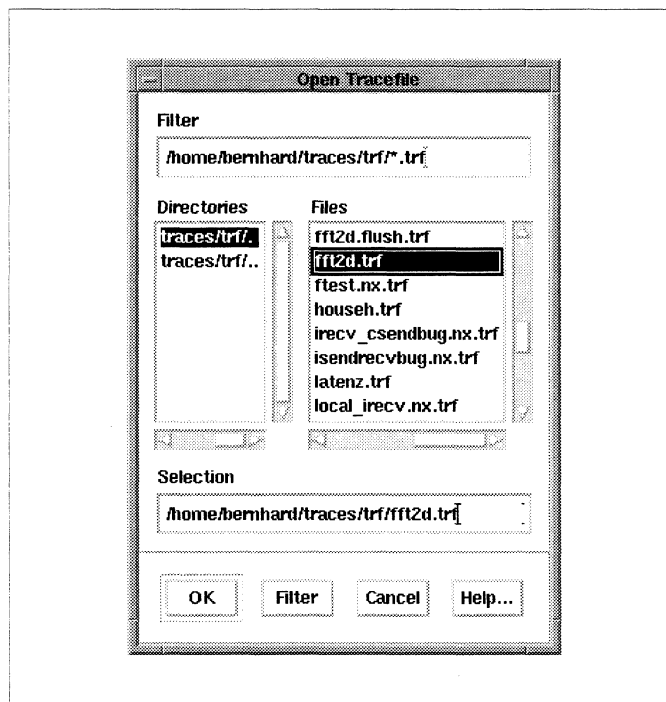


Figure 7-3. Open Tracefile Dialog

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

<i>OK</i>	Selects the entry in <i>Selection</i> as the trace file for replay and dismisses the file selection dialog.
<i>Filter</i>	Obtains a new file list, and possibly a new directory list.
<i>Cancel</i>	Dismisses the file selection dialog and no trace file is selected.
<i>Help</i>	Displays help topic text about the dialog.

Save Layout

Save Layout allows you to save the current layout of opened ParaGraph displays (including their position on the screen) to a file. *Save Layout* 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 **<Ctrl-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

Load Layout allows you to load a saved ParaGraph layout from a file. *Load Layout* 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 <Ctrl-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 menu items 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 menu items are provided:

- *Configure*
- *Select Nodes*
- *Colors*
- *Message Log*
- *Trace Filter*
- *Close All*

Configure

Configure displays the *Set ParaGraph Options* dialog. The default keyboard accelerator for *Configure* is <Ctrl-F>. Figure 7-5 shows the *Set ParaGraph Options* dialog.

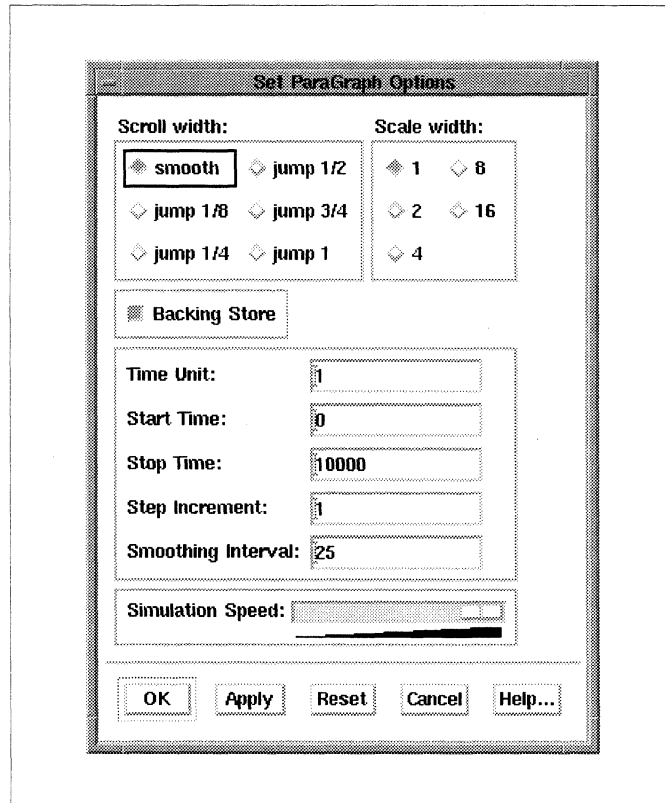


Figure 7-5. Set ParaGraph Options Dialog

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:

- | | |
|-----------------|---|
| <i>smooth</i> | Smooth scroll. |
| <i>jump1/8</i> | Displays are moved to the left by one eighth of the display width when the right edge of the display is reached. |
| <i>jump 1/4</i> | Displays are moved to the left by one quarter of the display width when the right edge of the display is reached. |

<i>jump1/2</i>	Displays are moved to the left by one half of the display width when the right edge of the display is reached.
<i>jump3/4</i>	Displays are moved to the left by three quarters of the display width when the right edge of the display is reached
<i>jump1</i>	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.

You can also set start and stop times for displays that scroll with time by pressing and dragging the middle mouse button over the display. Holding down the mouse button and dragging the mouse pointer forms a rectangle. When you release the mouse button, the rectangle defines the new start and stop times, and the time unit is adjusted so the new interval fits the default window size. These new values are shown in the Configure dialog, which automatically pops up. You can apply them or dismiss them with the Reset or Cancel buttons.

This allows you to examine a specific portion of a simulation simply by dragging over that portion to form a rectangle, applying the changes, resetting the simulation, and starting a new simulation run. The easiest method to restore the default start time, stop time, and time unit is to reload the tracefile using the Open command from the File menu.

When you set the start and stop time by forming a rectangle over a portion of a simulation, the start time is defined by the first event that occurs within the rectangle, and the stop time is defined by the first event that follows the end of the rectangle. It is possible to form a rectangle that contains no events. If this happens, the following message is displayed:

```
Defined stop time reached
```

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 Kiviatic 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

<i>Apply</i>	Applies changes to the dialog values.
<i>OK</i>	Applies changes and dismisses the dialog.
<i>Reset</i>	Cancels any changes and resets the status of the dialog to the state at the last <i>Apply</i> .
<i>Cancel</i>	Cancels changes and dismisses the dialog.
<i>Help</i>	Displays help text for the dialog.

Select Nodes

Select Nodes 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 **<Ctrl-M>**. Figure 7-6 shows the *Select Nodes* dialog.

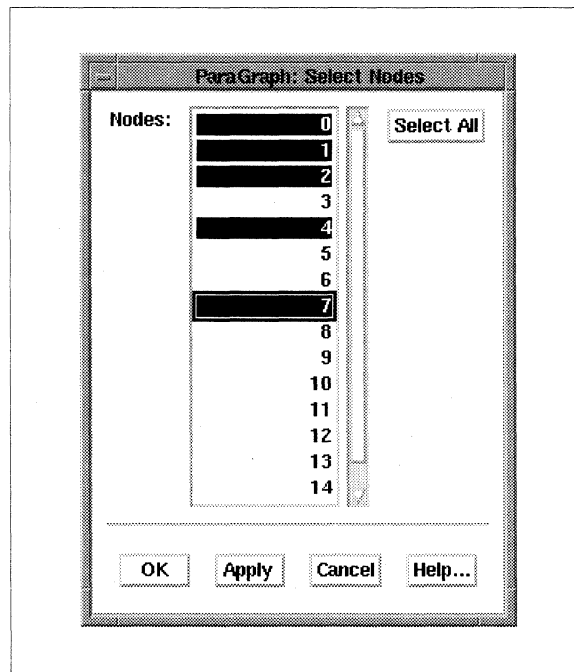


Figure 7-6. Select Nodes Dialog

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 **<Ctrl>** 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.

<i>OK</i>	Applies changes and dismisses the dialog.
<i>Cancel</i>	Cancels changes and dismisses the dialog.
<i>Help</i>	Displays help text for the dialog.

Colors

You can customize the colors used in the ParaGraph displays interactively or permanently. *Colors* allows you to choose display colors interactively. *Colors* 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 **<Ctrl-R>**. Figure 7-7 shows the *Select Colors* dialog.

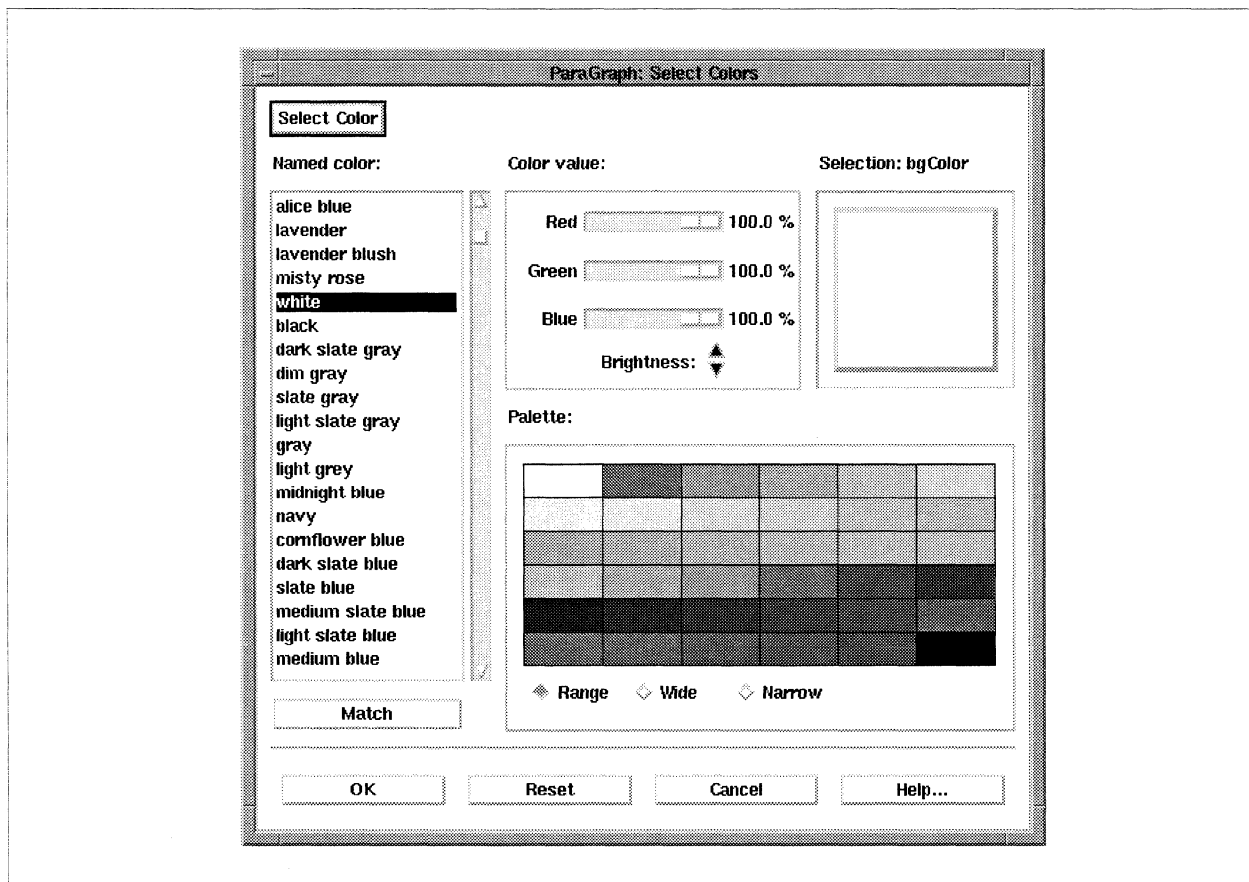


Figure 7-7. Select Colors Dialog

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.

<i>Range</i>	Covers the entire spectrum.
<i>Narrow</i>	Covers a narrow range around the current color.
<i>Wide</i>	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

<i>OK</i>	Applies changes and dismisses the dialog.
<i>Reset</i>	Cancel any changes not yet made permanent.
<i>Cancel</i>	Cancel changes and dismisses the dialog.
<i>Help</i>	Displays help text for the dialog.

You can save the color configuration to a file using *Save Layout* from the *File* menu. 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

Message Log 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 <Ctrl-M>. Figure 7-8 shows the *Message Log* window.

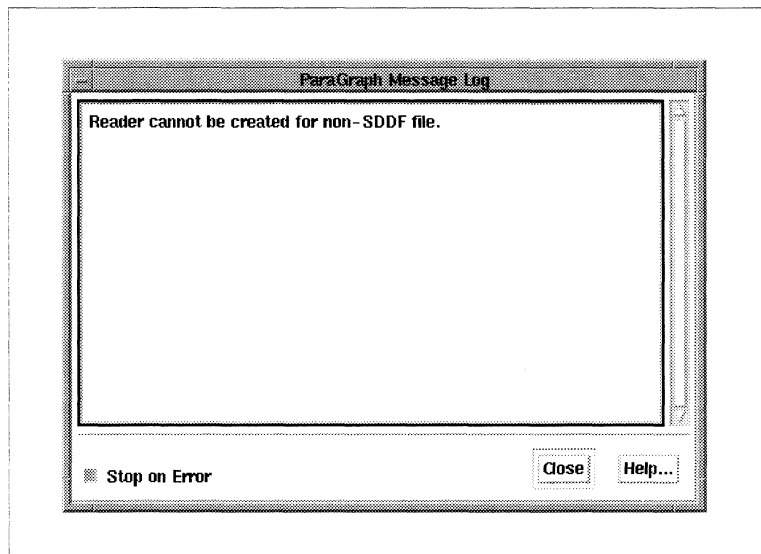


Figure 7-8. Message Log Window

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. The *Help* button displays help text about the message log.

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

Trace Filter 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 <Ctrl-T>. Figure 7-9 shows the *Filter Trace Records* dialog.

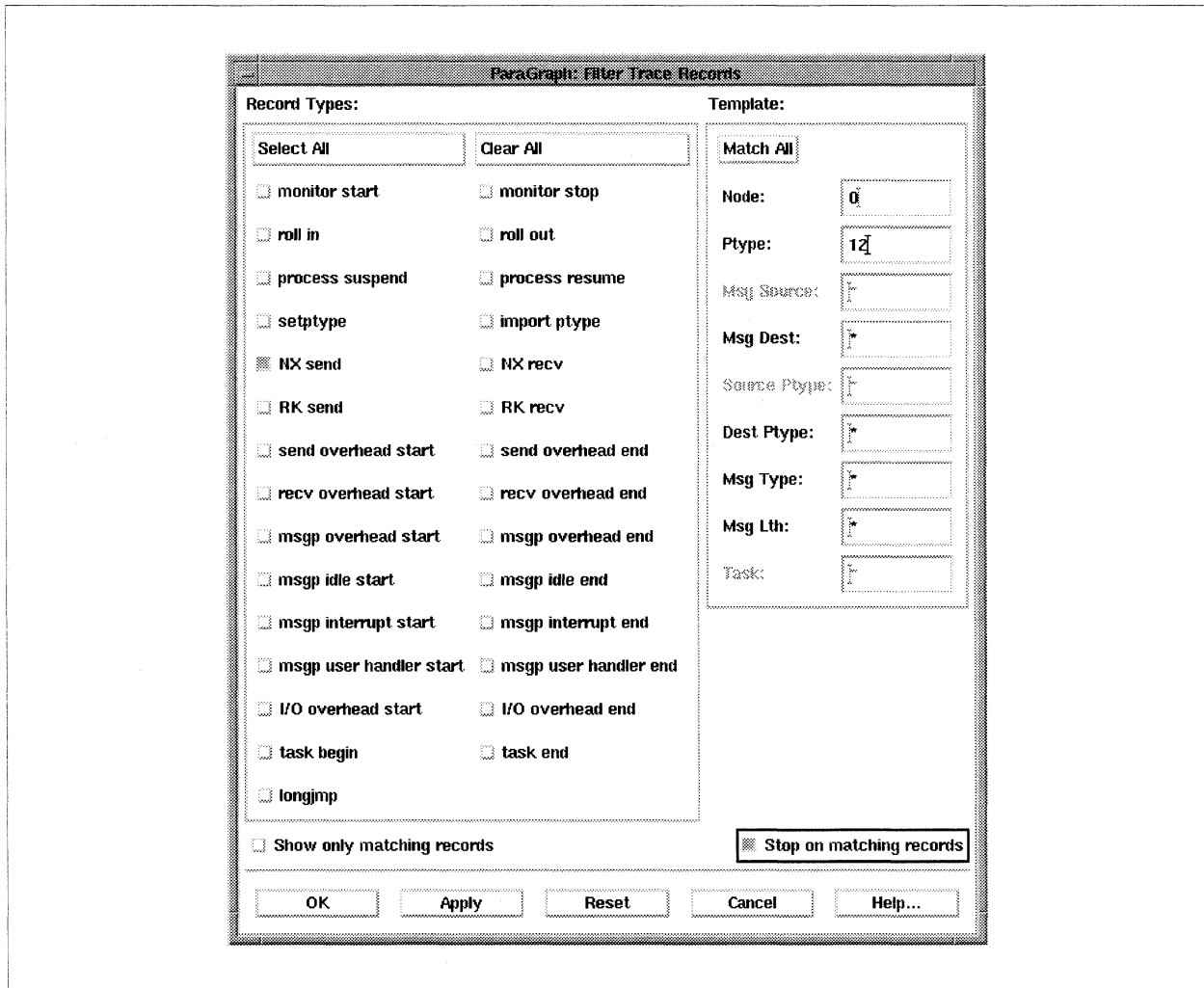


Figure 7-9. Filter Trace Records Dialog

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

<i>Apply</i>	Makes changes in the dialog effective.
<i>OK</i>	Applies the changes and exits out of the dialog.
<i>Reset</i>	Cancels any changes that have not yet been applied.

<i>Cancel</i>	Cancels changes and exits out of the dialog.
<i>Help</i>	Provides help text for the dialog.

Close All

Close All 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 <Ctrl-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” on page 7-29.

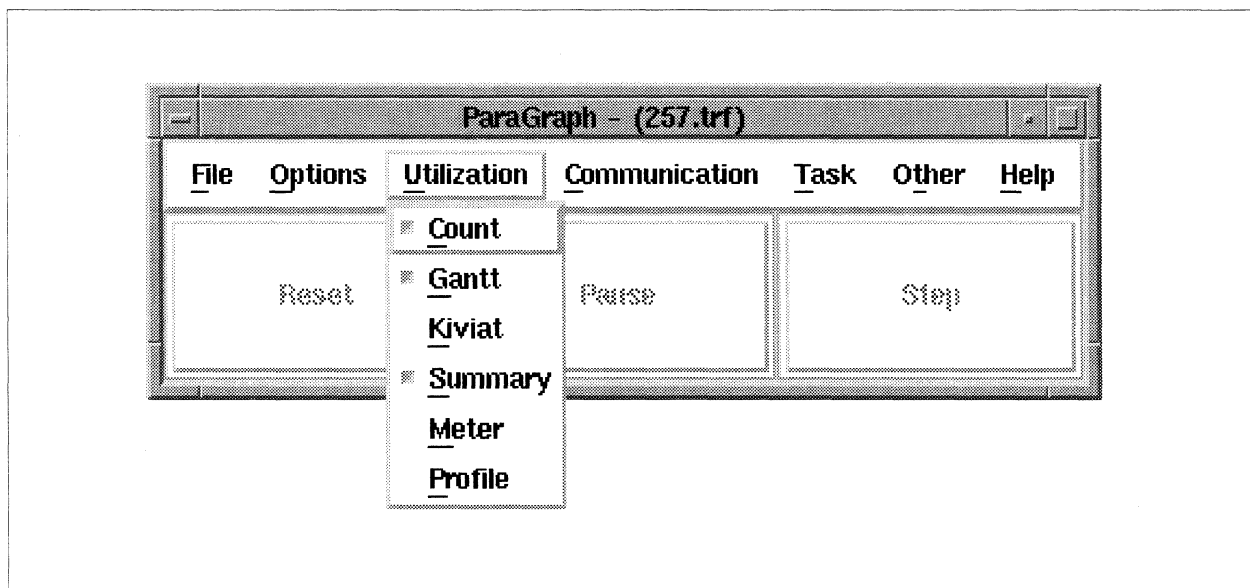


Figure 7-10. Utilization Menu

Utilization Menu

The *Utilization* Menu provides the following buttons:

- *Count*
- *Gantt*
- *Kiviat*
- *Summary*
- *Meter*
- *Profile*

Communication Menu

The *Communication* menu provides the following buttons:

- *Traffic*
- *Spacetime*
- *Queues*
- *Matrix*
- *Meter*
- *Animation*
- *Topology*
- *Network*
- *Node Info*
- *Color code*

Task Menu

The *Task* menu provides the following buttons:

- *Count*
- *Gantt*

- *Status*
- *Summary*

Other Menu

The *Other* menu provides the following buttons:

- *Clock*
- *Trace*
- *Statistics*
- *Processor Status*
- *Phase*
- *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.

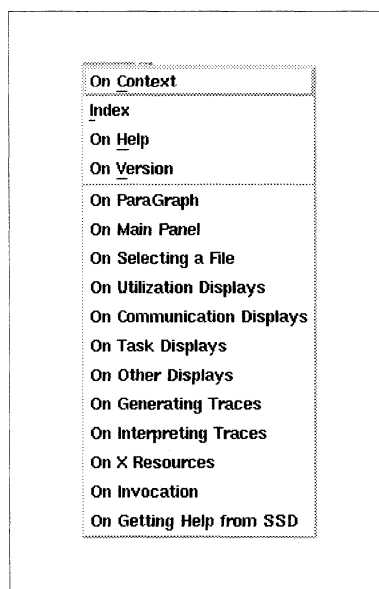


Figure 7-11. Help Menu

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.

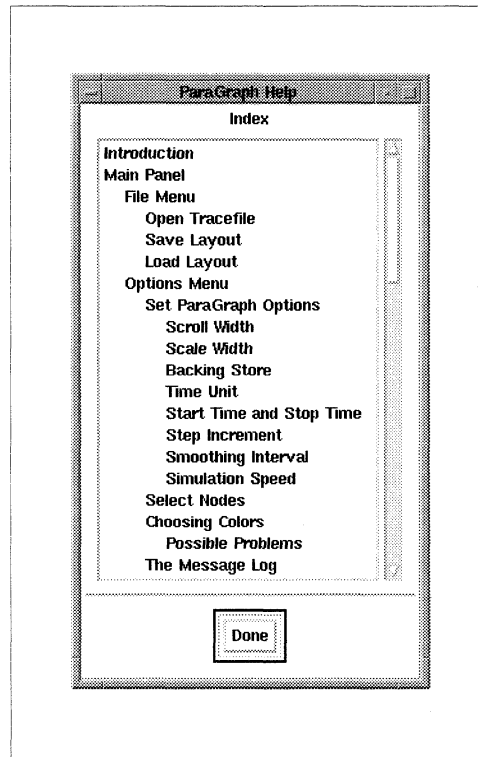


Figure 7-12. Help Index

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.

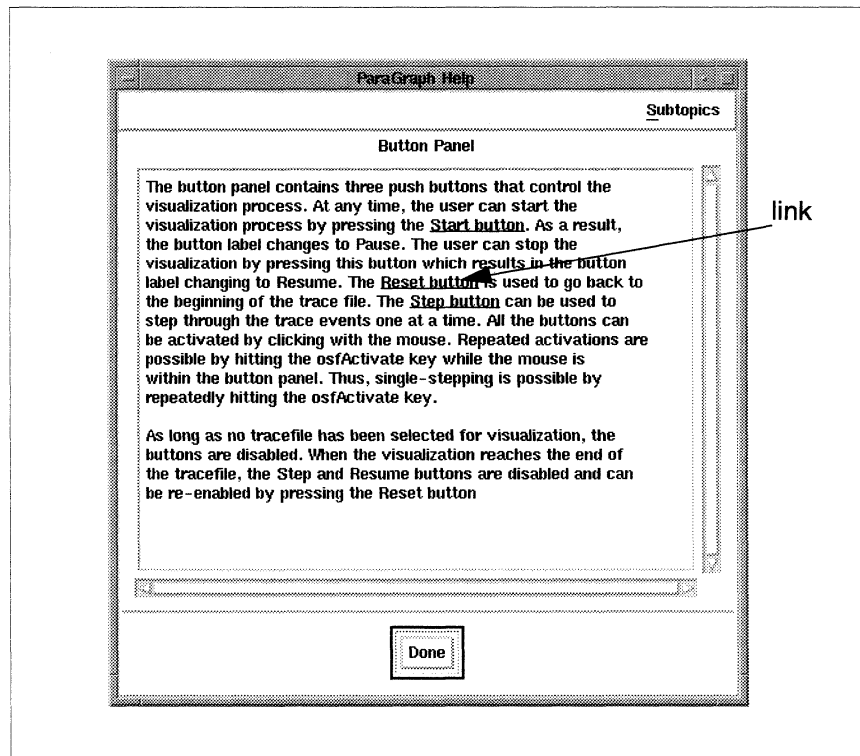


Figure 7-13. Help Topic Dialog

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. To increase the size of the performance monitoring trace buffer, use the IPD **instrument -bufsize** command. For a complete description of this command, refer to the *Paragon™ System Interactive Parallel Debugger Reference Manual*. 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.

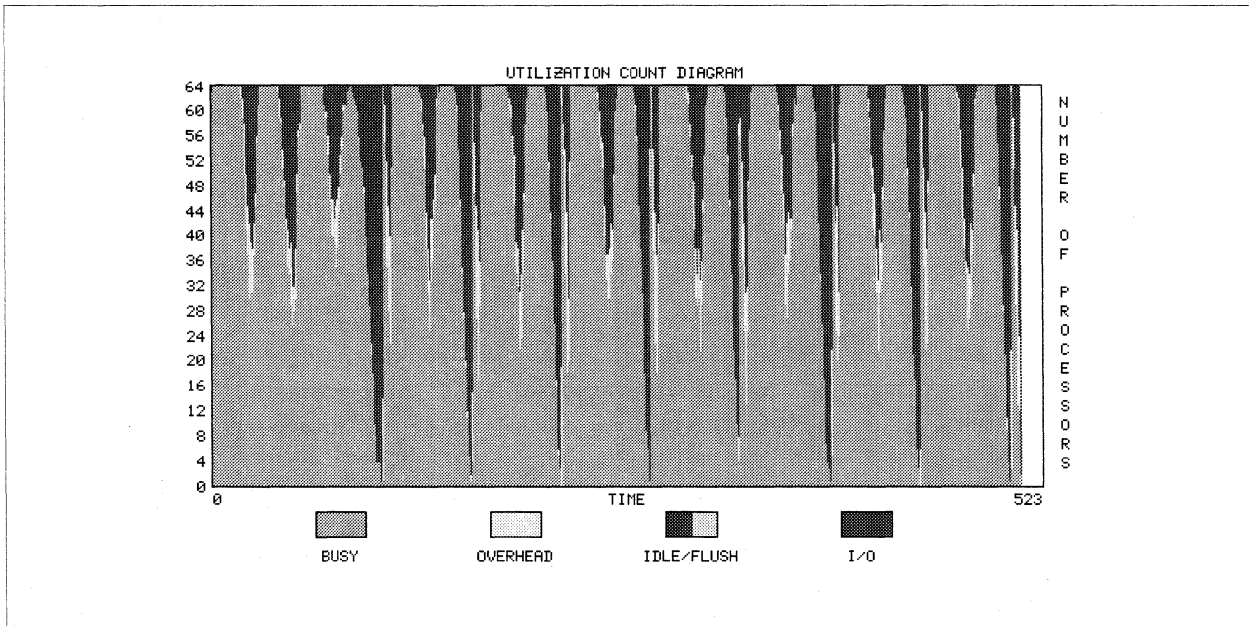


Figure 7-14. Count Display

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.

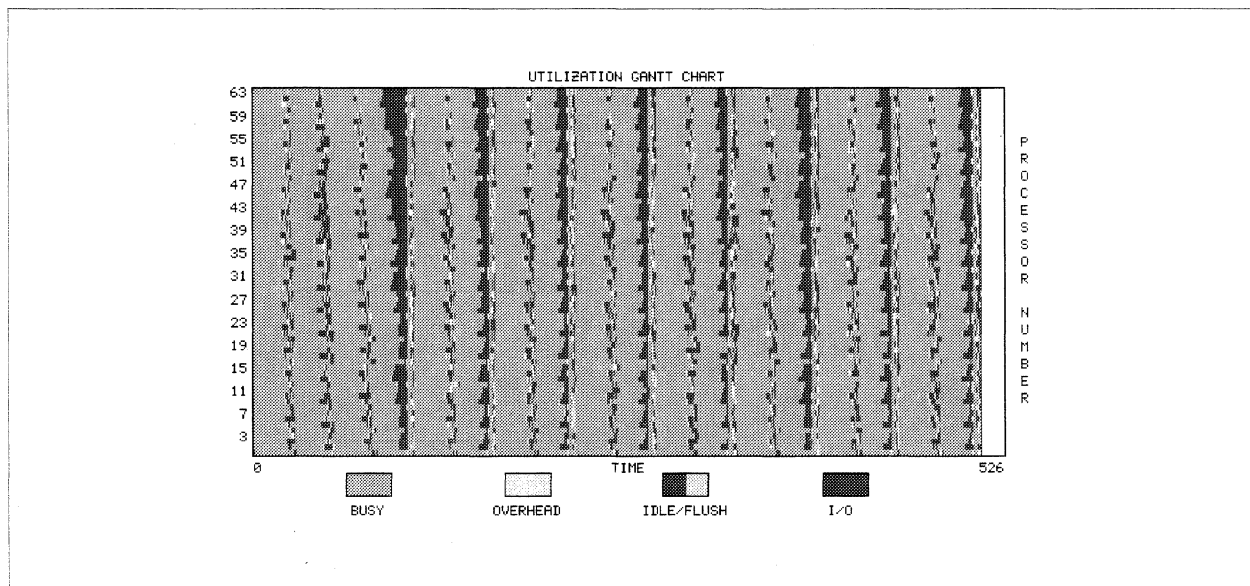


Figure 7-15. Gantt Display

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.

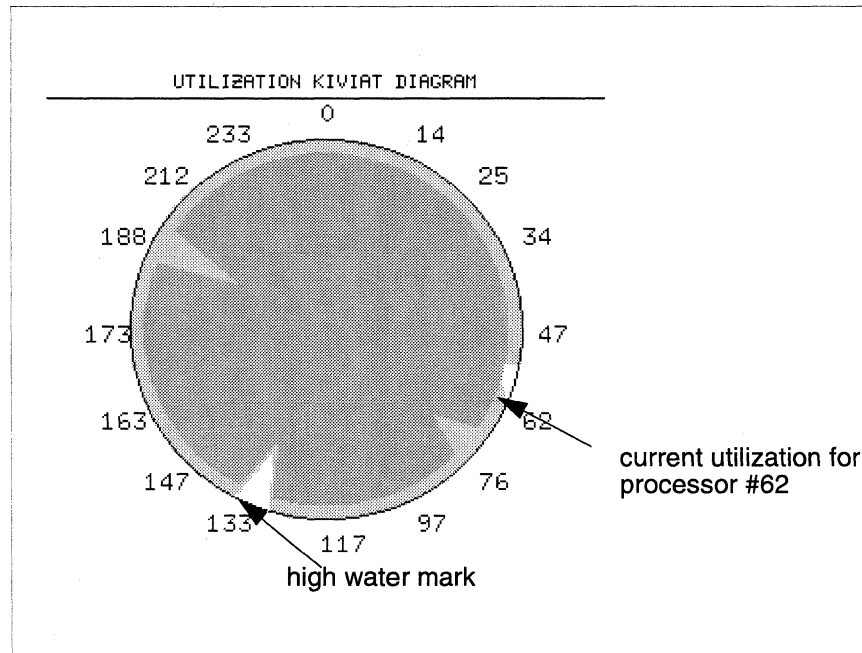


Figure 7-16. Kiviat Display

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.

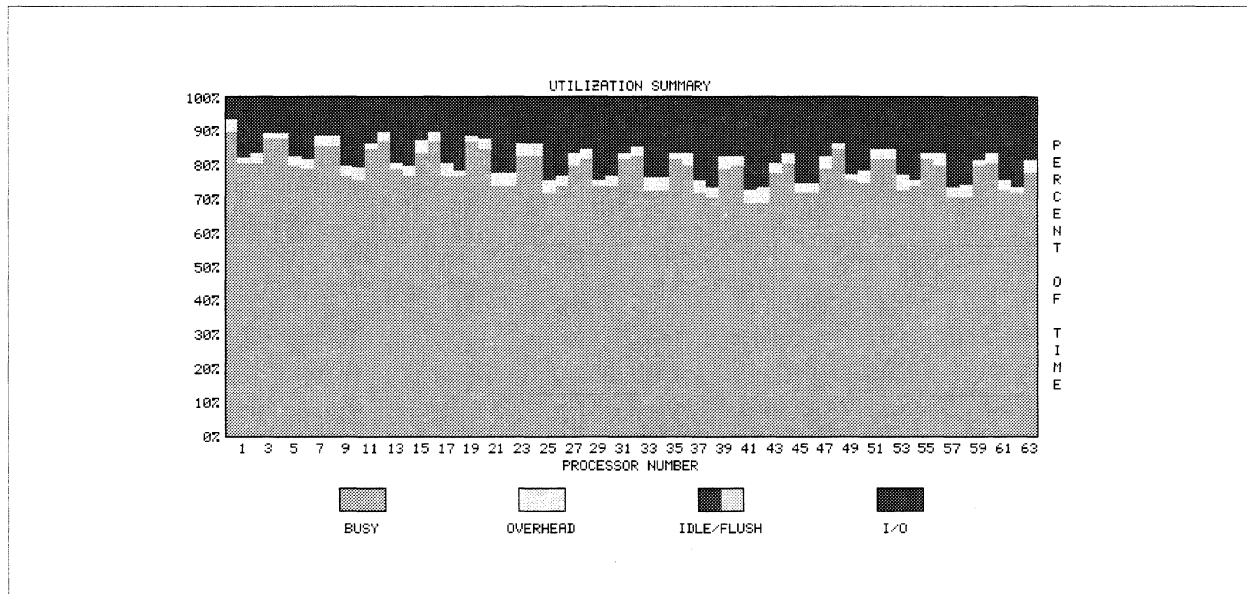


Figure 7-17. Summary Display

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.

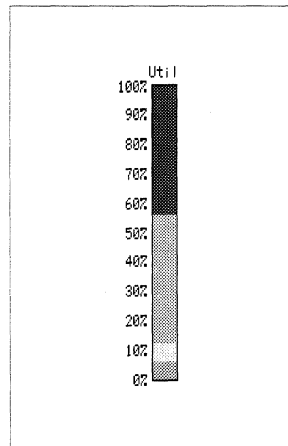


Figure 7-18. Meter Display

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.

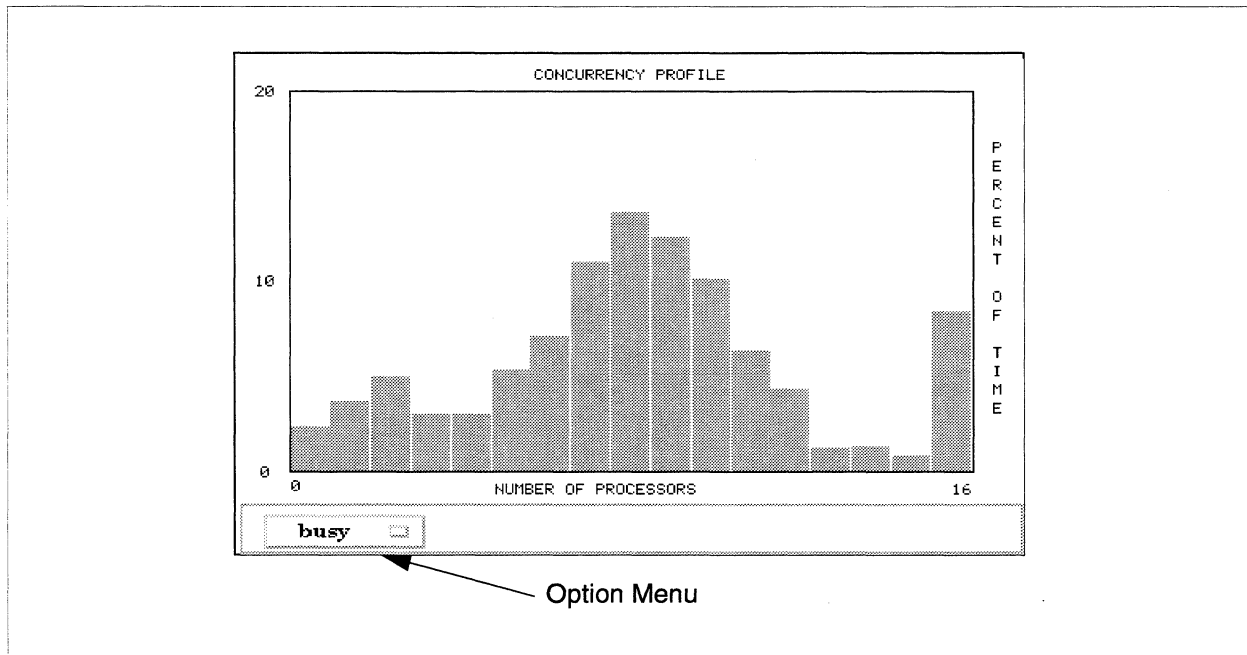


Figure 7-19. Profile Display

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

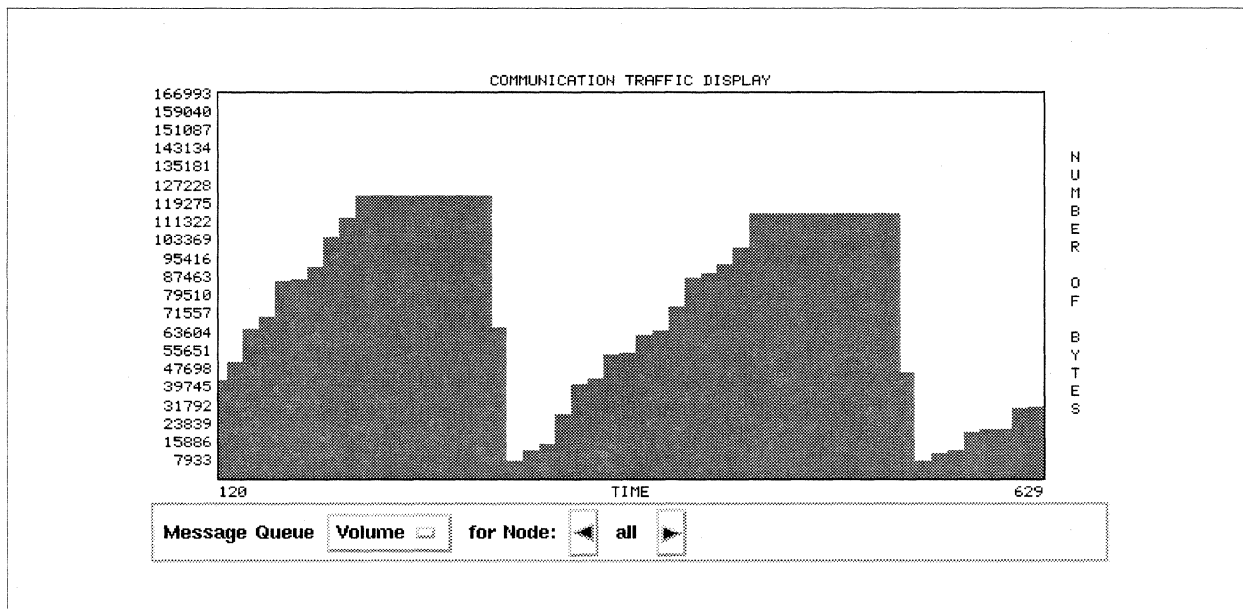


Figure 7-20. Traffic Display

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.

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

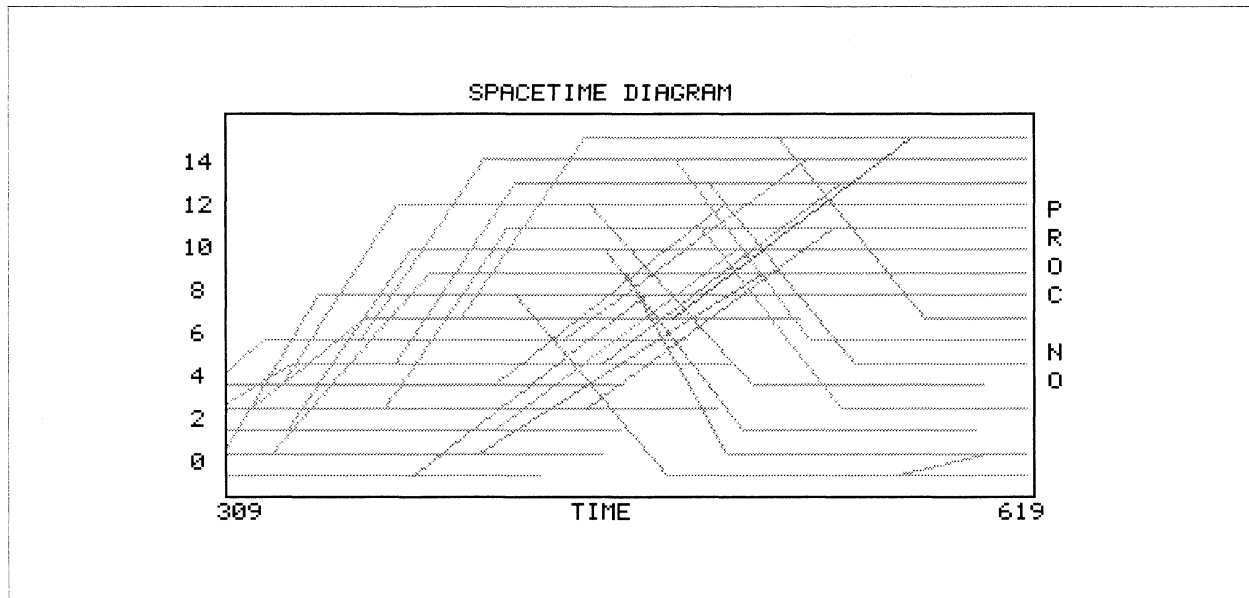


Figure 7-21. Spacetime Display

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.

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 7-45)). 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.

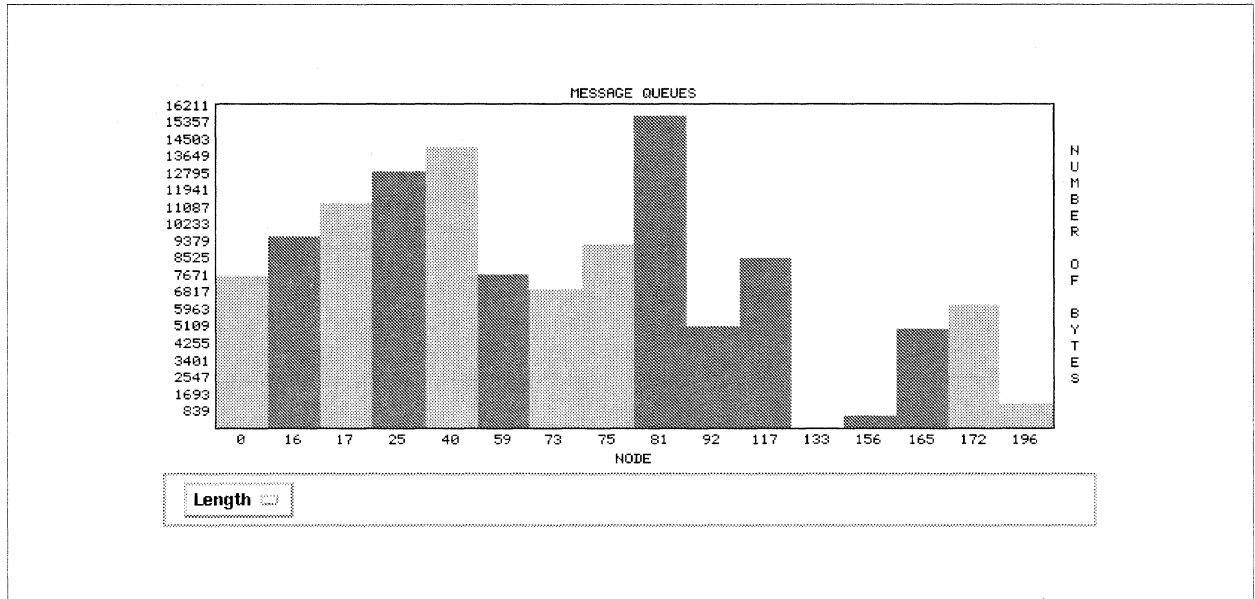


Figure 7-22. Queues Display

Matrix

Figure 7-23 shows the Matrix display.

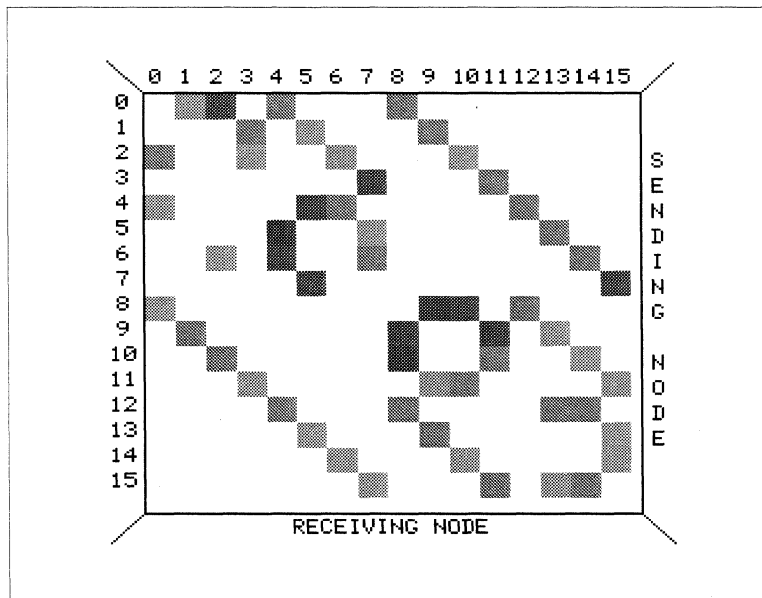


Figure 7-23. Matrix Display

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 7-45) 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.

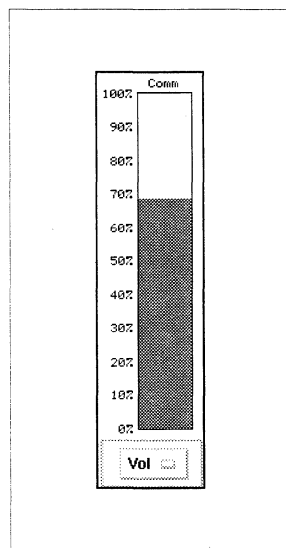


Figure 7-24. Communication Meter Display

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.

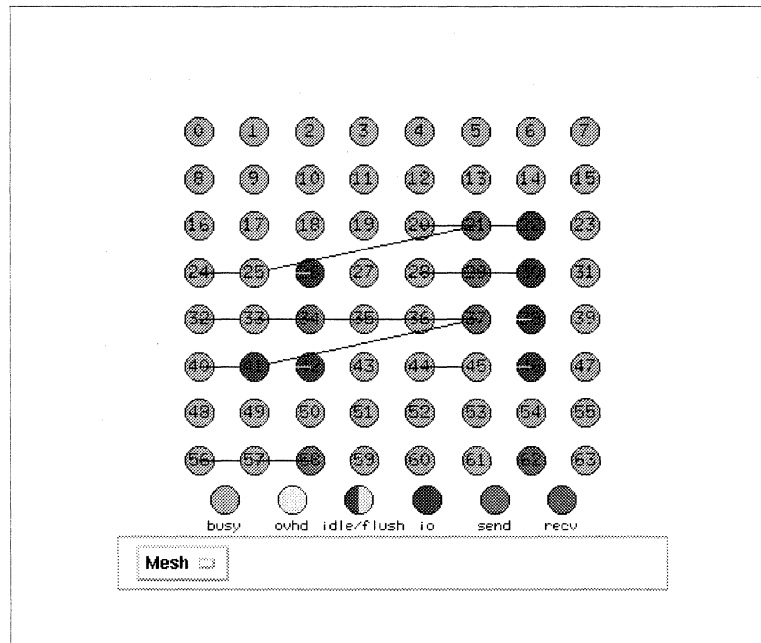


Figure 7-25. Animation Display

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.

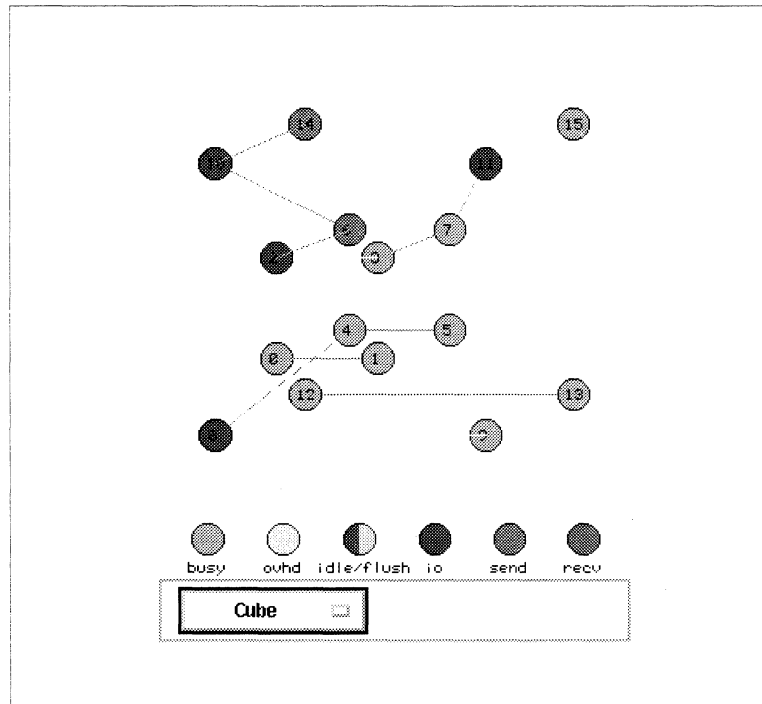


Figure 7-26. Topology Display

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.

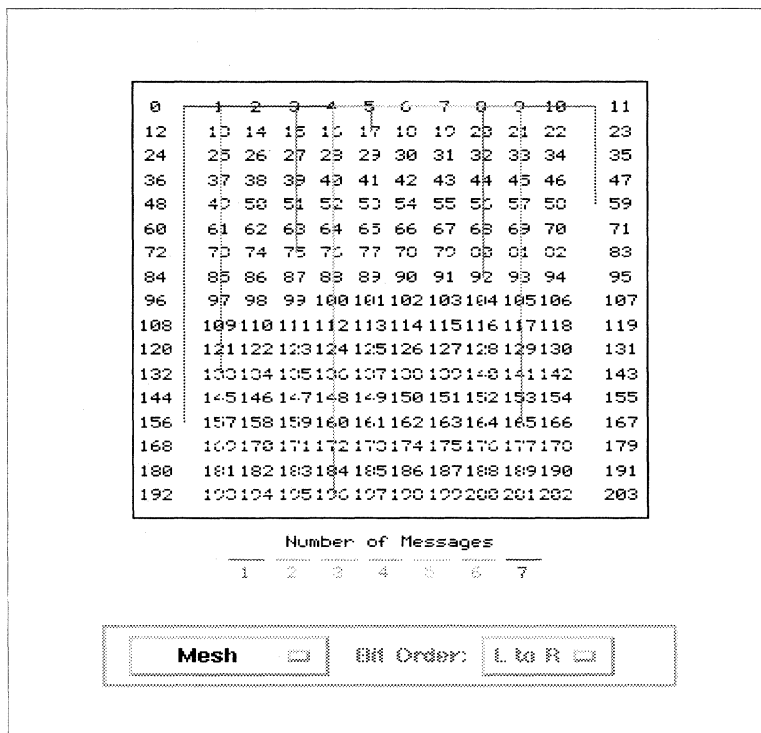
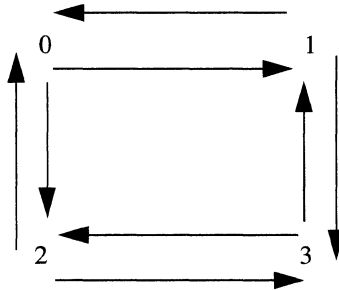


Figure 7-27. Network Display

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.

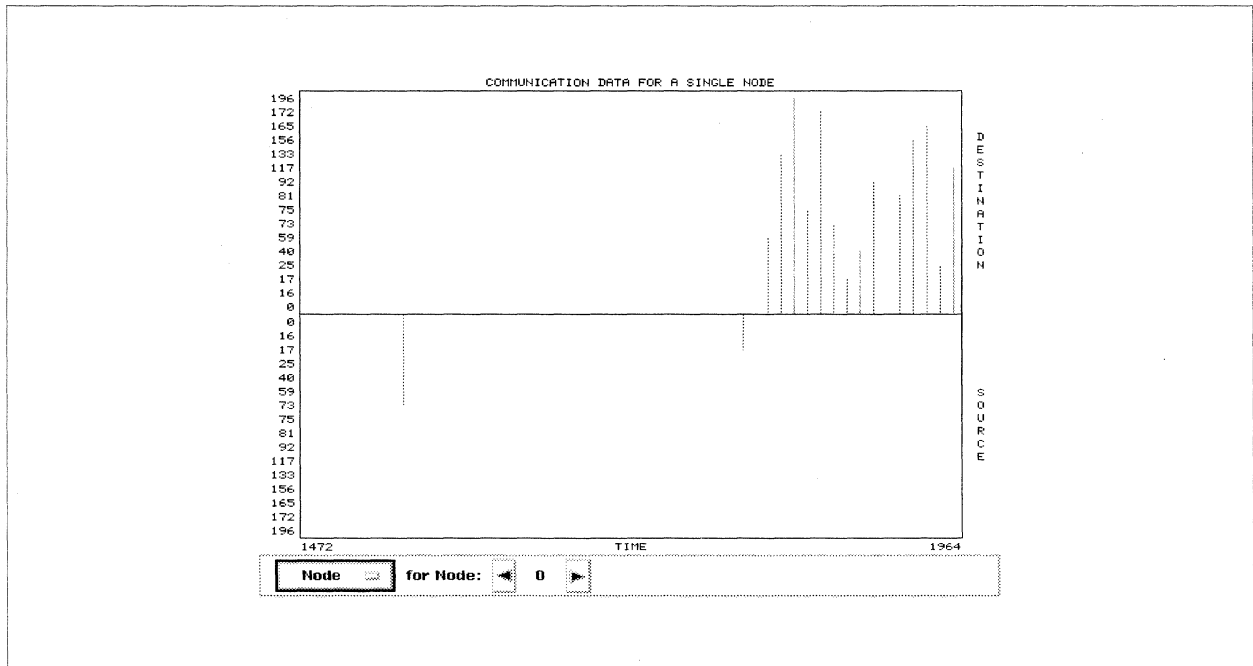


Figure 7-28. Node Info Display

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.

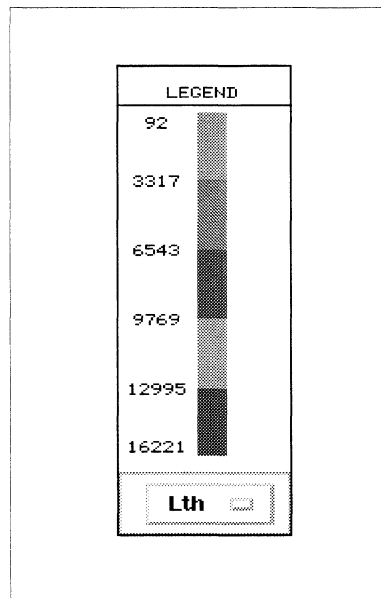


Figure 7-29. Color Code Display

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.

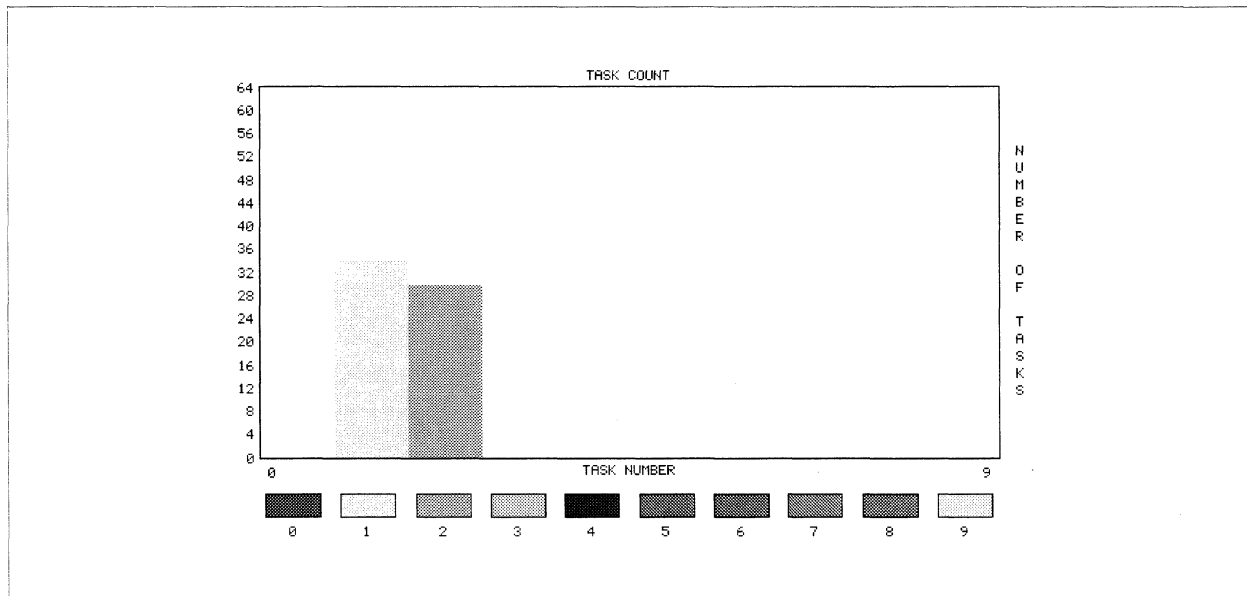


Figure 7-30. Task Count Display

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.

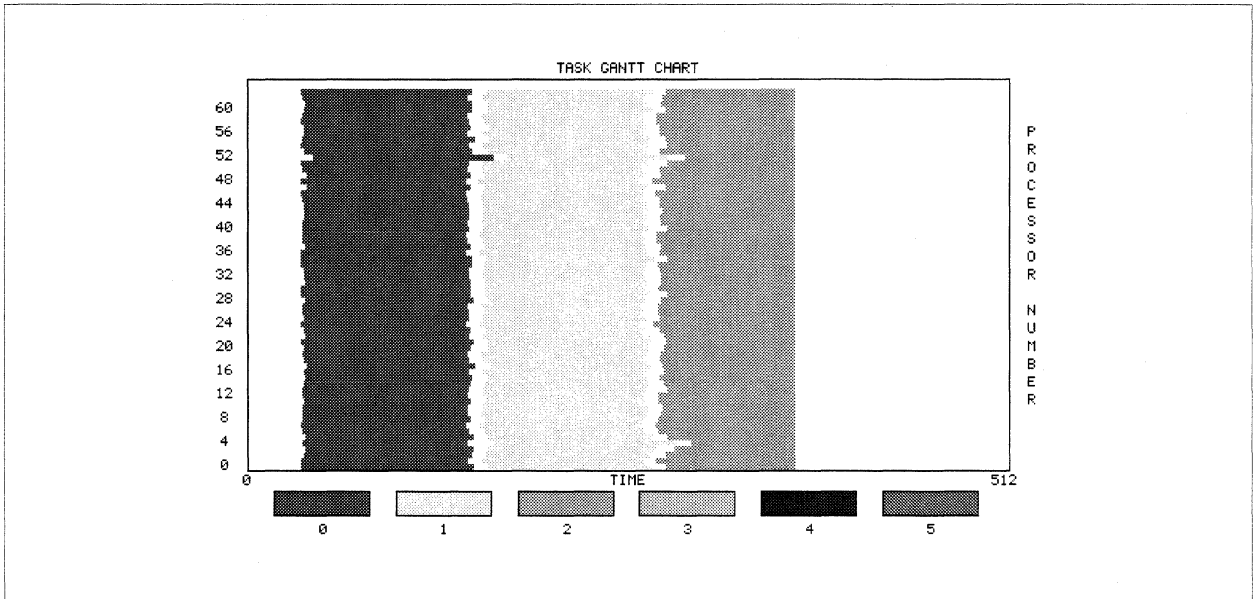


Figure 7-31. Task Gantt Display

Task Status

Figure 7-32 shows the Task Status display.

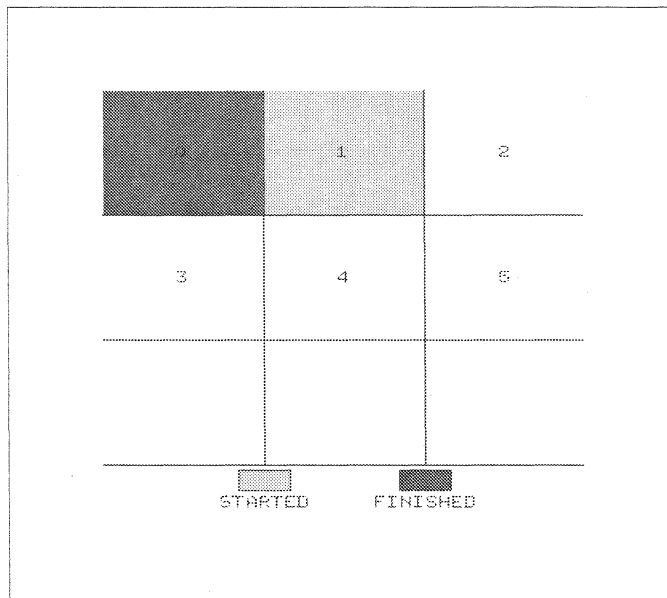


Figure 7-32. Task Status Display

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.

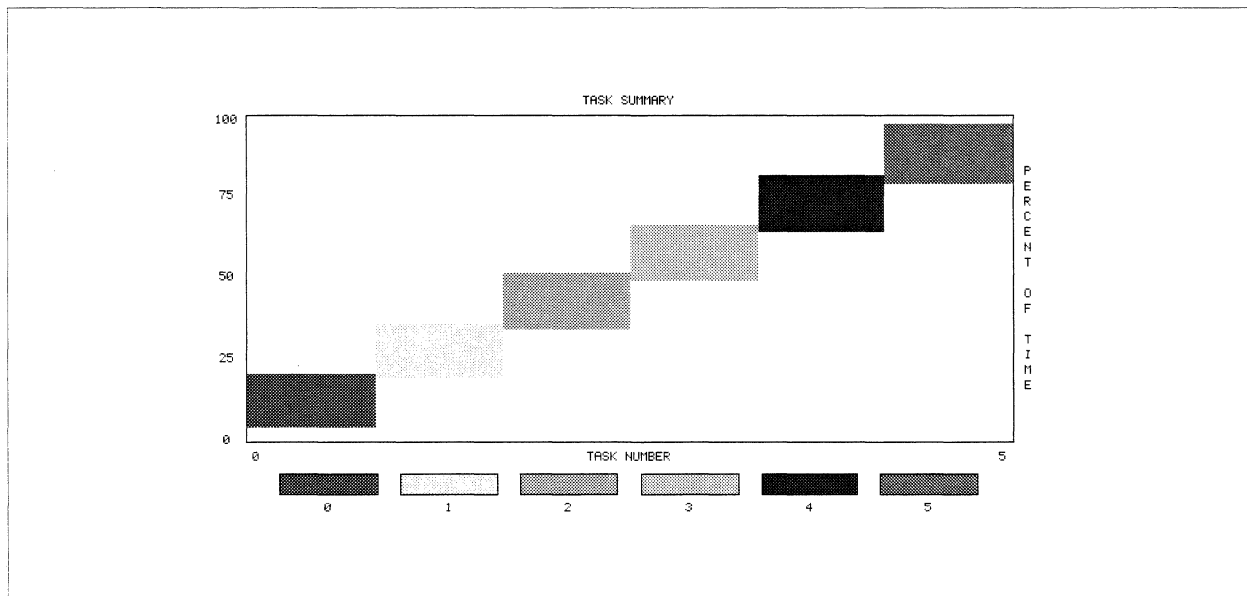


Figure 7-33. Task Summary Display

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.

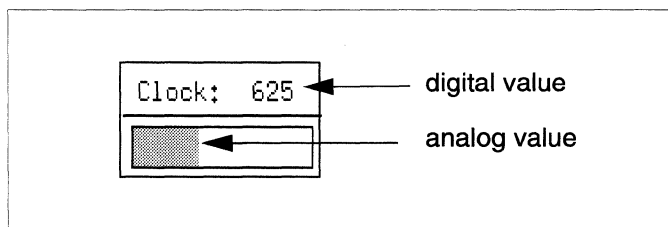


Figure 7-34. Clock Display

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.

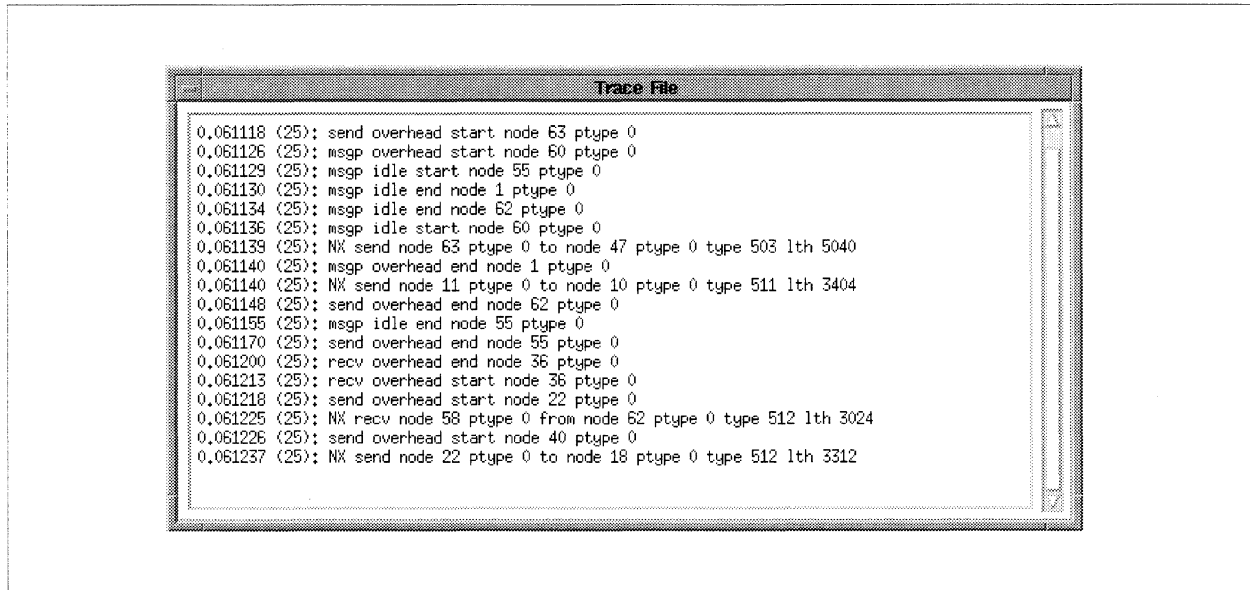


Figure 7-35. Trace Display

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.

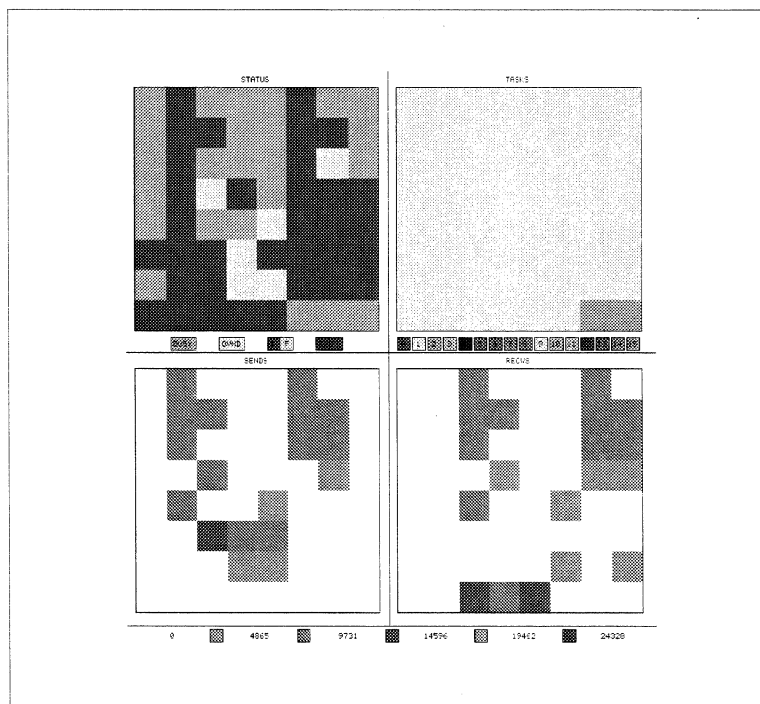


Figure 7-37. Processor Status Display

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.

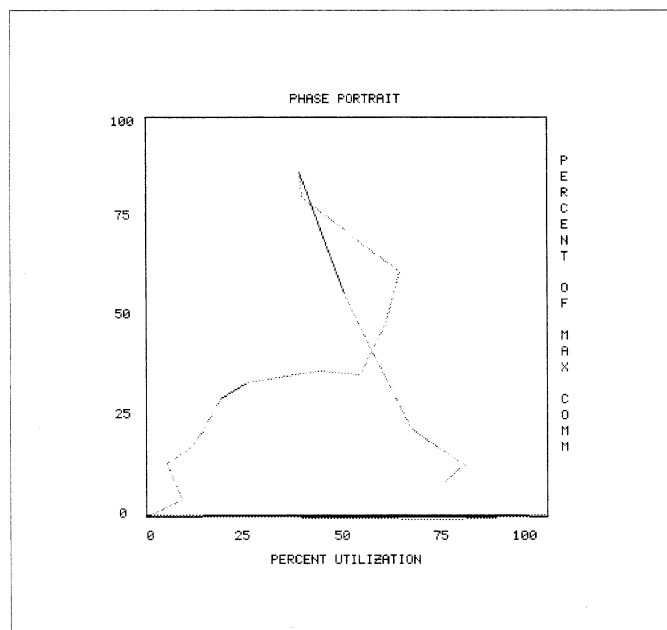


Figure 7-38. Phase Portrait Display

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.

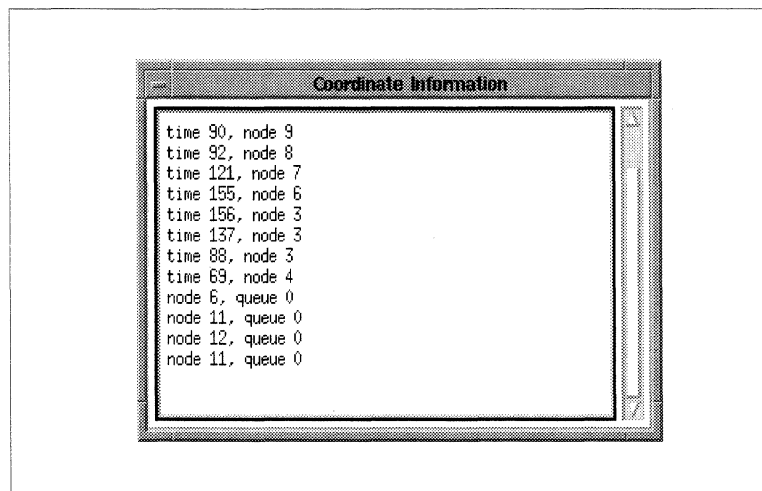


Figure 7-39. Info Display

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 Kiviati	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 with the **-bufsize** option of the IPD **instrument** command 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 (by using *Select Nodes*). 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`.

Paragraph*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 Communication Traffic 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 Queues display.
Paragraph*msghColor	The color used to draw message queue values high water 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*bkgColor	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 high water 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*msgColor	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*bkgColor	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/ -Cetc** 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 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 used. 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 section "Special Variables" on page 8-15 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 you 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. This option returns a status code of zero if the object files are up-to-date; otherwise, it 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. This option 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. This option 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** Simulates 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**, this option 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 context of the **pmake** command.

The other parameters to the **pmake** command are defined as follows:

|                         |                                                                                                                                                                                                                                                       |
|-------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>macro_definition</i> | 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 section "Macros" on page 8-14 for more information on using macros. |
| <i>target</i>           | Name of the target to build or update. Target names are typically executable files, but this is not always the case. If <i>target</i> is not specified, <b>pmake</b> 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, and configuration file support, as well as 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:

|           |                                                                                                          |
|-----------|----------------------------------------------------------------------------------------------------------|
| <b>-P</b> | Specifies the partition in which <b>pmake</b> runs jobs. The default is the service partition.           |
| <b>-j</b> | Specifies the maximum number of jobs that can run in parallel.                                           |
| <b>-l</b> | Restricts <b>pmake</b> 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 <i>targets</i> of the rules). Rules usually have a single target. A rule lists any files that the targets depend upon, called <i>dependencies</i> 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 <b>define</b> 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:

|                                      |                                                                                                                                                                                      |
|--------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>MACRO = value</i>                 | 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. |
| <i>MACRO := value</i>                | Simply expanding macro definition. The value is evaluated once when the macro is installed. Embedded macro references are evaluated at that time.                                    |
| <b>override</b> <i>MACRO = value</i> | Override directive. This 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 <i>suffix1</i> is replaced by <i>suffix2</i> in the expansion of <i>MACRO</i> . The <i>suffix1</i> must occur at the end of a word.                                                                                                                                                                                           |
| $\$(MACRO:pattern1=pattern2)$          | Pattern replacement references. The <i>pattern1</i> and <i>pattern2</i> values each contain the wild card character, <i>%</i> . Occurrences of <i>pattern1</i> in the expansion of <i>MACRO</i> are replaced by <i>pattern2</i> with the <i>%</i> character matching any stem.                                                                                            |
| $\$(MACRO/reg-expression/replacement)$ | Pattern replacement references. The <i>replacement</i> string is substituted for the <i>reg-expression</i> within the macro expansion. The valid forms of regular expressions are described in the manual page <b>regexp(3)</b> . Semicolons may be used in place of the slashes that separate the <i>MACRO</i> , <i>reg-expression</i> , and <i>replacement</i> strings. |
| $\$(MACRO:X)$                          | C-shell style modifiers. <i>X</i> may be <b>t</b> (tail), <b>h</b> (head), <b>r</b> (root) or <b>e</b> (extension).                                                                                                                                                                                                                                                       |
| $\$(MACRO?value1:value2)$              | Conditional expressions. Evaluates to <i>value1</i> if <i>MACRO</i> is defined and <i>value2</i> otherwise.                                                                                                                                                                                                                                                               |

## Special Macros

The following internal macros are automatically set as each target is processed:

|                     |                                                          |
|---------------------|----------------------------------------------------------|
| <code>\$@</code>    | The name of the current target.                          |
| <code>\$*</code>    | The base name of the current target.                     |
| <code>\$&lt;</code> | The name of the current dependency file.                 |
| <code>\$?</code>    | The list of dependencies that are newer than the target. |
| <code>%</code>      | The name of the library member being processed.          |
| <code>^</code>      | The list of all dependencies.                            |
| <code>\$\$@</code>  | 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:

|                  |                                                                                                                                                     |
|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>cpu</i>       | The CPU type of the target system in lower-case (for example, <code>i860</code> ). This variable is set automatically by the <b>pmake</b> command.  |
| <i>CPUTYPE</i>   | The CPU type of the target system in upper-case (for example, <code>I860</code> ). This variable is set automatically by the <b>pmake</b> command.  |
| <i>MAKE</i>      | The command line with which <b>pmake</b> was invoked, excluding options. This variable is set automatically by the <b>pmake</b> command.            |
| <i>MAKEFILES</i> | A list of description files to be read before any others. This variable may be set in the environment.                                              |
| <i>MAKEFLAGS</i> | A list of the options specified on the command line. This variable is set automatically by the <b>pmake</b> command.                                |
| <i>MAKELEVEL</i> | The current level of make recursion. This variable is set automatically by the <b>pmake</b> command.                                                |
| <i>OBJECTDIR</i> | The root of the object tree where <b>pmake</b> will build its targets.                                                                              |
| <i>SHELL</i>     | The shell to use for command execution. The default is <code>/bin/sh</code> . This variable may be set in the environment or in a description file. |

|                       |                                                                                                                                                             |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>SOURCEDIR</i>      | The roots of the source tree where <b>pmake</b> will search for sources.                                                                                    |
| <i>SUFFIXES</i>       | The list of default, known suffixes from built-in suffix rules. This variable is set automatically by the <b>pmake</b> command.                             |
| <i>target_machine</i> | The machine architecture of the target system in lower-case (for example, <i>paragon</i> ). This variable is set automatically by the <b>pmake</b> command. |
| <i>TARGET_MACHINE</i> | The machine architecture of the target system in upper-case (for example, <i>PARAGON</i> ). This variable is set automatically by the <b>pmake</b> command. |
| <i>VPATH</i>          | 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

**pmake** assigns special meanings to the following pseudotargets:

|                  |                                                                                                                                                                                                                                                               |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>.DEFAULT</b>  | 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.                                                                                                                              |
| <b>.EXIT</b>     | If defined in the description file, <b>pmake</b> processes this target and its dependencies after all other targets are built.                                                                                                                                |
| <b>.EXPORT</b>   | Variables listed as dependencies of this target are expanded and exported to the environment in which <b>pmake</b> runs its commands. <b>pmake</b> does not normally export variables defined within a definition file.                                       |
| <b>.IGNORE</b>   | Ignore errors. When this target appears in the description file, <b>pmake</b> ignores non-zero error codes returned from commands.                                                                                                                            |
| <b>.INIT</b>     | If defined in the description file, this target and its dependencies are built before any other targets are processed.                                                                                                                                        |
| <b>.PHONY</b>    | The dependencies of this target are considered to be “phony” targets. When it is time to consider such a target, <b>pmake</b> will run its commands unconditionally regardless of whether a file with that name exists or what its last modification time is. |
| <b>.PRECIOUS</b> | List of files not to delete. <b>pmake</b> does not remove any of the files listed as dependencies for this target when interrupted. <b>pmake</b> normally removes the current target when it receives an interrupt.                                           |
| <b>.SILENT</b>   | Run silently. When this target appears in the description file, <b>pmake</b> does not echo commands before executing them.                                                                                                                                    |
| <b>.SUFFIXES</b> | The dependencies of this target are the suffixes that <b>pmake</b> 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:

|                                            |                                                                                |
|--------------------------------------------|--------------------------------------------------------------------------------|
| <b>ifeq</b> ( <i>arg1</i> , <i>arg2</i> )  | The conditional evaluates to true if <i>arg1</i> is equal to <i>arg2</i> .     |
| <b>ifneq</b> ( <i>arg1</i> , <i>arg2</i> ) | The conditional evaluates to true if <i>arg1</i> is not equal to <i>arg2</i> . |
| <b>ifdef</b> <i>variable-name</i>          | The conditional evaluates to true if <i>variable-name</i> is defined.          |
| <b>ifndef</b> <i>variable-name</i>         | The conditional evaluates to true if <i>variable-name</i> is not defined.      |

For complete information on how to construct and use a description file, refer to the *GNU Make* manual.



# Index

---

## A

animation display 7-40

applications

    debugging 2-1

    examining and modifying 2-14

    loading 1-3

assign 2-38

## B

break 2-37

breakpoints 2-13

## C

clock display 7-50

color code display 7-45

commands

    debugger commands 2-3

    ipd 2-2

    IPD command syntax 2-5

    paragraph 7-2

    paraide 1-2

communication displays 7-4, 7-35

communication meter display 7-39

compiling for debugging 2-2

configuration file 8-9

configuring ParAide 1-36

context 2-28

continue 2-39

count display 7-31

## D

debug context 2-7

debug environment 2-10

debugger commands 2-3

debugging hints 2-24

debugging programs 2-1

description file 8-11

directives in makefiles 8-11

DISPLAY variable 1-2, 3-3, 5-2, 6-2, 7-2

**displays**

- animation 7-40
- clock 7-50
- color code 7-45
- communication 7-4
- communication meter 7-39
- count 7-31
- gantt 7-31
- info 7-54
- kiviat 7-32
- matrix 7-39
- meter 7-34
- network 7-42
- node info 7-44
- phase portrait 7-54
- processor status 7-53
- profile 7-34
- queue 7-37
- spacetime 7-36
- statistical summary 7-51
- summary 7-33
- task 7-5
- task count 7-47
- task gantt 7-47
- task status 7-49
- task summary 7-49
- topology 7-41
- trace 7-50
- traffic 7-35
- utilization 7-3

**E**

- environment variables
  - ParAide 1-40

- examining and modifying programs 2-14

**example**

- description 2-26
- preparation 2-27

**F**

- file management 1-3

**files**

- debug log file 2-11
- IPD command file 2-11
- layout file 7-10
- managing 1-3
- Paraide resource file 1-36
- selecting 1-34

- frame 2-33

**G**

- gantt display 7-31

- graphical tools 1-1
  - invoking 1-3
  - ParAide 1-1

**I**

- icon strip 1-32

- info display 7-54

- instrument command 7-1

- Interactive Parallel Debugger (IPD) 2-1

- interrupts from the keyboard 2-26

**invoking**

- graphical tools 1-3
- IPD 2-2
- ParaGraph 7-2
- ParAide 1-1

- invoking IPD 2-27

**IPD 2-1**

- breakpoints, tracepoints, and watchpoints 2-13
- command file 2-11
- commands 2-3
- compiling for debugging 2-2
- controlling the debug environment 2-10
- debug context 2-7
- debugging hints 2-24
- examining and modifying programs 2-14
- instrument command 2-14
- invoking 2-2
- loading programs for debugging 2-9
- recording debugging sessions 2-11
- running programs 2-12
- specifying a partition for debugging 2-2

**K**

- keyboard interrupts 2-26
- kiviat display 7-32

**L**

- launching graphical tools 1-3
- list 2-35
- loading applications 1-3
- loading files 2-28
- loading programs
  - for debugging 2-9
- log 2-30
- log file 2-11

**M**

- Makeconf 8-9
- makefile 8-11
  - dependencies 8-11
  - parallel execution 8-5
  - rules 8-11

managing files 1-3

matrix display 7-39

message log 7-19

meter display 7-34

more 2-31

msgqueue 2-34

**N**

network display 7-42

node info display 7-44

**O**

online help 1-28

opening a display 1-2, 3-3, 5-2, 6-2, 7-2

**P**

ParaGraph 7-1

- communication displays 7-4, 7-35

- configuring 7-61

- default configuration 7-64

- display menus 7-23

- displays 7-29

- error conditions 7-19

- file menu 7-7

- help menu 7-25

- hints for using 7-56

- invoking 7-2

- main window 7-6

- options menu 7-11

- restrictions 7-58

- saving layout 7-10

- setting colors 7-17

- task displays 7-5, 7-46

- utilization displays 7-3, 7-29

paragraph command 7-2

**ParAide**

- configuring 1-36
- default configuration 1-39
- environment variables 1-40
- file menu 1-5
- help menu 1-28
- icon strip 1-32
- invoking 1-2
- invoking graphical tools 1-18
- main window 1-3
- mesh menu 1-7
- selecting files and directories 1-34
- shell panel 1-34
- tool menu 1-18

paraide command 1-2

- options 1-2

parallel make 8-1

partitions

- specifying a partition for debugging 2-2

performance monitoring 2-14

performance visualization 7-1

phase portrait display 7-54

pmake 8-1

- configuration file support 8-9
- load control 8-7
- OBJECTDIR variable 8-9
- options not in GNU make 8-10
- separate object and source trees 8-9
- SOURCEDIR variable 8-9
- specifying a partition 8-5
- specifying multiple source trees 8-9
- targets not in GNU make 8-10

process 2-32

processor status display 7-53

processor utilization 7-4

profile display 7-34

**Q**

queues display 7-37

**R**

recvqueue 2-34

resource file 1-36

run 2-32

**S**

setting DISPLAY variable 1-2, 3-3, 5-2, 6-2, 7-2

shadow sources with pmake 8-9

shell panel 1-33

source 2-31

spacetime display 7-36

stack trace facility 2-33

statistical summary display 7-51

step 2-38

stop 2-33

summary display 7-33

**T**

task count display 7-47

task displays 7-5, 7-46

task gantt display 7-47

task status display 7-49

task summary display 7-49

topology display 7-41

trace display 7-50

trace events 7-56

tracepoints 2-13

traffic display 7-35

**U**

utilization displays 7-3, 7-29

**W**

wait 2-32

watchpoints 2-13

**X**

X resources 1-36, 7-61

X toolkit 1-2

xhost 1-1, 3-3, 5-2, 6-2, 7-2

