Chapter 3

Using Prism

The Prism programming environment is the standard tool with which users interact with the CM-5 in developing and executing their data parallel or message-passing programs. Prism provides an integrated graphical environment in which you can

- execute the program
- debug the program
- analyze the program's performance
- visualize data from the program
- obtain on-line documentation for the CM-5
- . . . and much more

This chapter gives an overview of how to use Prism. For complete information on Prism, see the *Prism User's Guide* and *Prism Reference Manual*.

In this chapter we use the sample data parallel program **`primes`** to illustrate many of Prism's capabilities. If you have the program (and a CM-5) available, you can follow along on-line as we help you get started. To compile the program for debugging via Prism, specify the **`-g`** option; note that compiling with **`-g`** causes performance to be artificially slow. To compile the program to collect performance analysis data, specify the **`-cmprofile`** option. See Section 2.3.

Section 3.12 describes Node Prism, the version of Prism used for message-passing programs. Even if you only use the CM-5 for message-passing programs, we recommend that you read the beginning of the chapter first, to get an overview of Prism's basic features, then read Section 3.12 to learn about the added features in Node Prism.

Copyright © 1993–1994 Thinking Machines Corporation

3.1 Starting Prism

Prism runs under the X Window System.

To start Prism, first log in to a partition manager as described in Chapter 2. For example,

```
% rlogin mars
```

Start X on the partition manager as you normally do; if you're not familiar with X, ask your system administrator for help. Make sure your **DISPLAY** environment variable is set for the terminal or workstation from which you are running X. For example, if your workstation is named Valhalla, you can issue this command (if you are running the C shell):

```
% setenv DISPLAY valhalla:0
```

To start up Prism, issue the command **prism** at your UNIX prompt; you can include the name of your executable program as an argument. Thus,

```
% prism primes
```

loads the **primes** program. (This assumes that you are starting Prism from the directory in which **primes** is located.) The main Prism window then appears, as described below.

To start up Node Prism, include the **-node** option; see Section 3.12 for more information on Node Prism.

If you don't have an X terminal or workstation, you can run a nongraphical version of Prism; specify the **-C** option on the command line. This version of Prism is discussed in Section 3.10.

3.2 A Tour of Prism

Figure 7 shows the main window of Prism with the **primes** program loaded. In Prism, you can operate with a mouse, use keyboard equivalents of mouse actions, or issue keyboard commands.

Copyright © 1993–1994 Thinking Machines Corporation

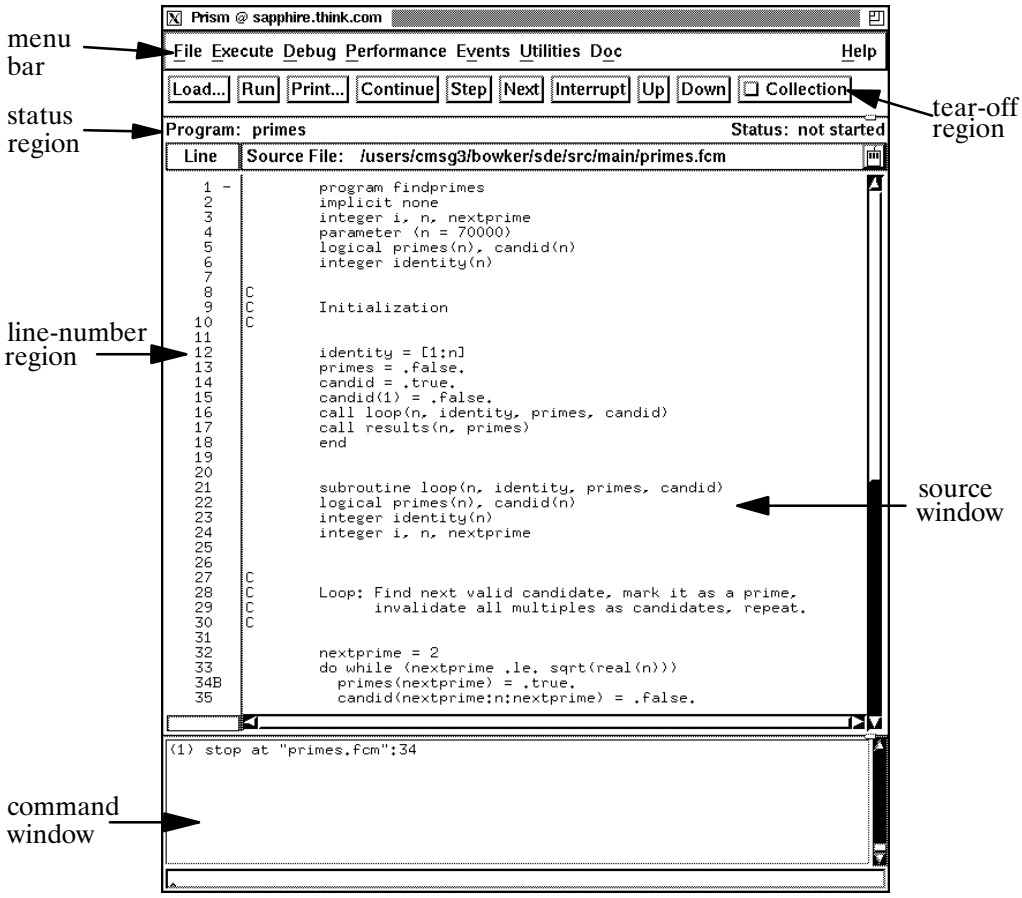

Figure 7. Prism's main window.

Left-clicking on items in the **menu bar** displays pulldown menus that provide access to most of Prism's functionality.

You can add frequently used menu items and commands to the **tear-off region**, below the menu bar, to make them more accessible; many items are there by default.

The **status region** displays the program's name and messages about the program's status.

The **source window** displays the source code for the program. You can scroll through this source code and display any of the source files used to compile the program. When a program stops execution, the source window updates to show the code currently being executed. You cannot edit the file in the

Copyright © 1993–1994 Thinking Machines Corporation

source window; instead, you call up a separate editor if you want to make changes to your code.

The **line-number region** is associated with the source window. You can click to the right of a line number in this region to set a breakpoint at that line. In Figure 7, a breakpoint is set at line 34. The *execution pointer* (**>**) in the line-number region shows the line at which the program is currently stopped. The *scope pointer* (−) shows the beginning of the scope that Prism uses in interpreting the names of variables. The symbol * appears when the execution pointer and the scope pointer are at the same line.

The **command window** at the bottom of the main Prism window has two areas. The *history region*, at the top, displays messages and output from Prism. You can type commands on the *command line* at the bottom of the window, rather than use the graphical interface.

3.3 Executing a Program

Once **`primes`** is loaded, you can execute it from within Prism. The simplest way to do this is to click on **Run** in the tear-off region of the main Prism window. (You can also choose the **Run** selection from the **Execute** menu, or issue the **`run`** command on the command line.)

After you have clicked on **Run**,

- The status changes to **running**.
- Many menu items are grayed out, indicating that they are temporarily unavailable.
- An Input/Output window appears; when the program has finished execution, the results will appear here. Figure 8 shows what it will look like for **`primes`**.

CMOST Version 7.3, April 1994
Copyright © 1993−1994 Thinking Machines Corporation

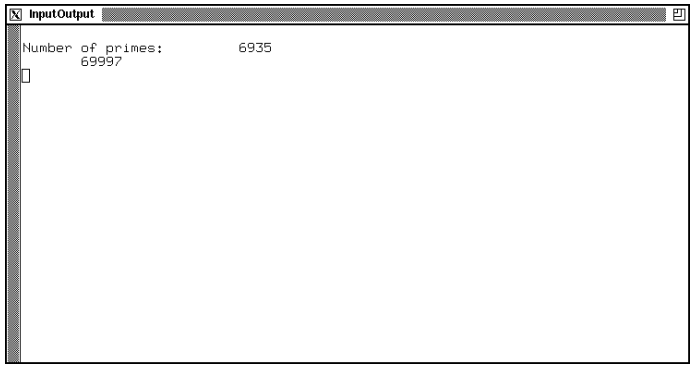

Figure 8. The Input/Output window.

3.3.1 Interrupting, Continuing, and Single-Stepping

You can also interrupt execution of a program in Prism, and single-step through it. Let's run **`primes`** again. As before, start execution by clicking on **Run**; you don't need to reload the program. Notice that Prism reuses the same Input/Output window.

While the program is executing, click on the **Interrupt** button in the tear-off region (notice that this button doesn't turn gray; this indicates that the function is still available). When you click on **Interrupt**,

- Prism displays a message in the history region, stating where the program was interrupted.
- The source window updates to show the code being executed. The execution pointer in the line-number region points to the line currently being executed.
- The status of the program is **stopped**.

When the program is stopped, you can single-step through it; for example,

- Click on **Step** in the tear-off region to execute the next line.
- Click on **Next** in the tear-off region to execute the next statement (a function call counts as a single statement).

Copyright © 1993–1994 Thinking Machines Corporation

In both cases, the execution pointer moves to point to the new execution point in the source window. If the step takes a long time to execute, unavailable buttons and menu selections are grayed out.

You can also issue the **stepi** and **nexti** commands from the command line to step by machine instruction. When you issue these commands, the history region displays the instruction at the address at which you stop.

Click on **Continue** in the tear-off region to continue execution without single-stepping. The program will run to completion.

3.4 Debugging a Program

In Prism, you can perform standard debugging operations like setting breakpoints and examining the call stack. Prism also provides a comprehensive method for controlling the execution of a program by means of an *event table*.

3.4.1 Setting a Simple Breakpoint

A *breakpoint* stops execution of a program. The easiest way to set a breakpoint in Prism is to left-click the mouse to the right of the line number at which you want the program to stop. Notice that when you move the mouse pointer into this area, it turns into a **B**. When you left-click next to a line, the **B** appears at that point.

Set a breakpoint at line 15 of **primes**. Note that when you do this, Prism displays a message in the history region:

```
(1) stop at "primes.fcm":15
```

Now, when you execute **primes**, the program will stop whenever it reaches line 15.

To delete the breakpoint, left-click on the **B**; it disappears.

Copyright © 1993–1994 Thinking Machines Corporation

3.4.2 Using Commands

You can set more complicated breakpoints by issuing the **stop** command on the command line. For example, issue this command:

```
stop in loop after 10
```

This tells Prism to stop execution the tenth time the program reaches the function **loop**. Note that it causes a **B** to be placed at line 32 in the line-number region, next to the first line in the routine **loop**.

Similarly, you can use the **trace** command to stop execution, then have it start again automatically.

3.4.3 Using the Event Table

The breakpoints and traces described above are *events* that control the execution of a program. Prism provides an event table that lets you create all such events in one place before you execute your program.

To display the event table, choose **Event Table** from the **Events** menu. Figure 9 shows the event table, listing the two breakpoints we discussed above.

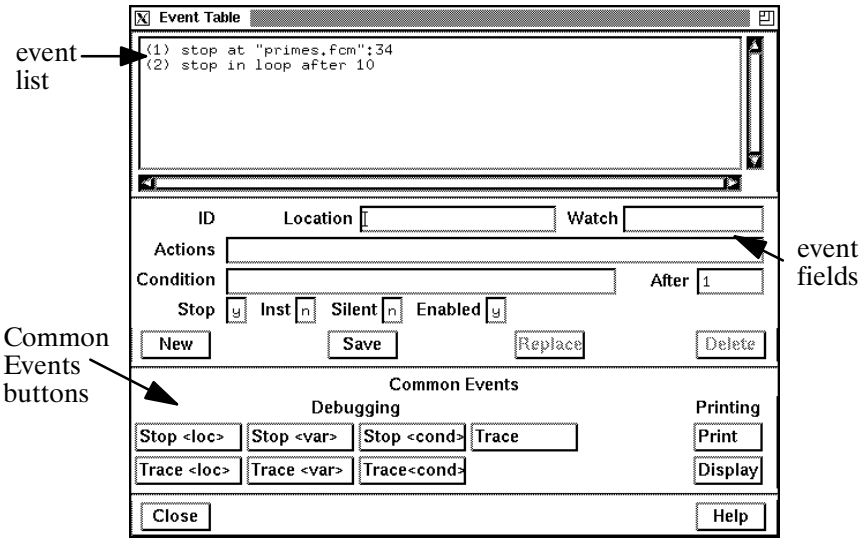

Figure 9. The event table.

Copyright © 1993–1994 Thinking Machines Corporation

The top area of the event table is the *event list* — a scrollable region in which events are listed. When you execute the program, Prism uses the events in this list to control execution. Each event is listed in a format in which you could type it as a command in the command window. It is prefaced by an ID number assigned by Prism. For example, in Figure 9, the events have been assigned the IDs 1 and 2. Creating an event by issuing a command, or in any other way, adds an event to this list.

The middle area of the event table is a series of fields that you fill in when editing or adding an event; only a subset of the fields is relevant to any one event. For example,

- Fill in the **Location** field to specify a line number when setting a breakpoint at a line.
- In the **Actions** field, specify any actions that are to accompany the event (for example, printing the value of a variable or displaying the call stack). You can include most Prism commands in this field.
- In the **Condition** field, specify a logical condition that must be met if the event is to take place.

The area headed **Common Events** contains buttons that provide shortcuts for creating certain standard events. For example, if you click on **Stop <cond>**, the **Condition** field is highlighted and the cursor is placed in it. You can then enter a condition and click on **Save** to add the event to the event list. The program will stop when the specified condition is met.

3.4.4 Displaying and Moving through the Call Stack

You can use Prism to display and move through the call stack — the list of procedures and functions currently active in a program. Choose the **Where** selection from the **Debug** menu to display a window containing the call stack; the window is updated automatically when execution stops or you issue commands that change the stack. You can click on a function in the window to make that function current. You can also click on **Up** or **Down** in the tear-off region to move up or down one level in the call stack.

If you run **`primes`** to the breakpoint you have set in **`loop`**, the call stack looks like the one shown in Figure 10. Note that the first two levels of the call stack are routines called by the CM-5 timesharing daemon and the run-

Copyright © 1993–1994 Thinking Machines Corporation

time system, respectively; you don't have to be concerned with them in debugging your program.

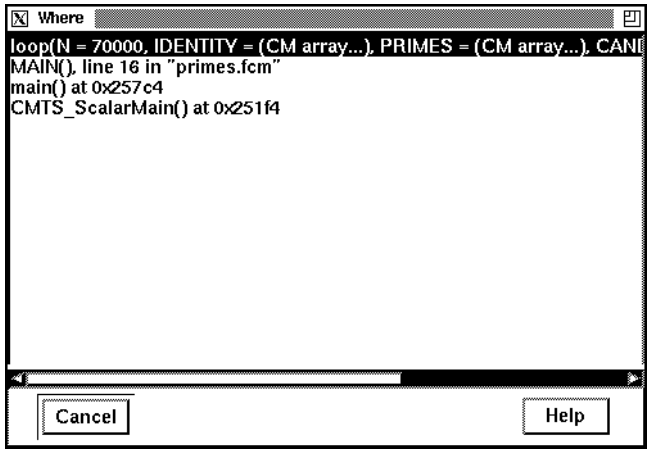

Figure 10. The Where window.

3.5 Visualizing Data

One of Prism's most important features is the ability to display graphically the values of the large arrays that are typical of CM-5 programming.

In the **`primes`** program, the largest variable is also called **`primes`**; it is an array of 70,000 logicals. Each element of the array is set to *true* if its index is a prime number, and *false* if it isn't.

Let's print the values of **`primes`**.

First, issue the command **`reload`** from the command line to reload **`primes`**. This deletes any existing events.

Then scroll through the source window and set a breakpoint at line 62.

In the source window, find a line where **`primes`** appears (for example, line 43). Move the mouse pointer to **`primes`**. Press the left mouse button, and drag the mouse until **`primes`** is highlighted. Now let go of the button.

Copyright © 1993–1994 Thinking Machines Corporation

Keeping the mouse pointer in the source window, press the right button. A menu appears. Right-click on **Print** in this menu.

Prism then displays a *visualizer* for the array, as shown in Figure 11.

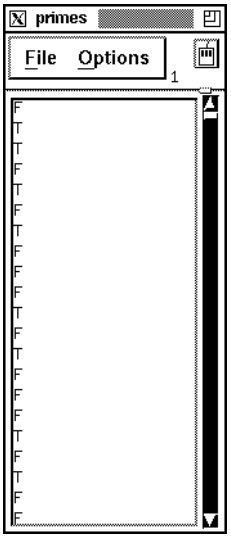

Figure 11. A visualizer for `primes`.

The figure shows the values of the first elements of the array. You can scroll through the window to see other elements.

Other representations are available besides straight text. Left-click on **Options** to display a menu. Then left-click on the **Representation** selection to display another menu. Left-click on **Threshold** in this menu. The T's and F's apparently disappear. In fact, the T's have changed into black pixels, the F's into white pixels. To make them more visible, follow these steps:

1. Left-click on **Options** again.
2. Left-click on **Parameters** from the menu.
3. Change the number in the **Field Width** box to 20.
4. Left-click on **Apply**. The black values will be much more visible.

Now left-click on **Ruler** in the **Options** menu. A ruler surrounds the values, making it easier to see separate elements. Figure 12 is an example.

Copyright © 1993–1994 Thinking Machines Corporation

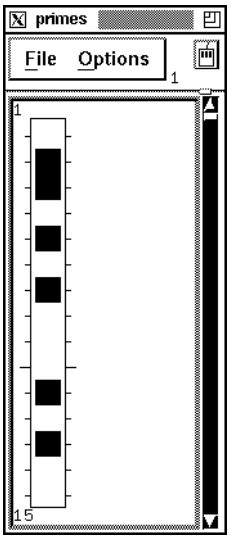

Figure 12. A threshold visualizer for `primes`.

To see the index of an individual value, move the mouse pointer onto the value. Press the **Shift** key; while pressing it, press the left mouse button. The index and value are displayed.

To close the visualizer, left-click on **File** in the visualizer window, then left-click on **Close**.

3.5.1 Other Representations

In addition to the text and threshold visualizers shown above, Prism provides a variety of other representations that you may find useful, depending on the kind of data you want to look at:

- Use the **Colormap** representation (if you have a color workstation) to display the values as a range of colors.
- Use the **Dither** representation to display the values as a shading from black to white.
- Use the **Graph** representation to display the values as a graph, with the index of each array element on the horizontal axis and its value on the vertical axis.

Copyright © 1993–1994 Thinking Machines Corporation

- Use the **Surface** representation (if your data has more than one dimension) to render the 3-dimensional contours of a 2-dimensional slice of data.

- Use the **Vector** representation to display complex numbers as vectors.

You can change the parameters of a visualizer to adjust the display. For example, you can pick a minimum and a maximum for many of these representations.

In addition, for most representations, you can set a context, using an expression that evaluates to true or false for every element of the array; the values of elements that evaluate to false are grayed out.

Finally, you can display summary statistics for an array or parallel variable, showing its minimum, maximum, mean, and other information.

3.5.2 Other Methods of Choosing the Data to Visualize

In the example at the beginning of this section, we chose the variable to visualize by dragging the mouse over it in the source window. Prism provides several other methods for choosing the variable. You can also visualize expressions.

In Prism, you can either *print* or *display* data. Printing data shows the value(s) of the data at a specified point during execution. Displaying data causes its value(s) to be updated every time the program stops execution. (You can also update a print visualizer, by choosing an option from its **File** menu.)

Here are some of the ways in which you can print or display data:

- You can create print and display events in the event table, or by choosing **Print** or **Display** from the **Events** menu. This lets you specify, for example, the location in the program at which the printing is to take place.

- You can print or display at the program's current stopping place by choosing **Print** or **Display** from the **Debug** menu. This pops up a dialog box in which you can specify the variable or expression, along with the kind of window in which you want the values to be shown.

CMOST Version 7.3, April 1994
Copyright © 1993–1994 Thinking Machines Corporation

- You can print from the source window by pressing the **Shift** key while pressing the left mouse button and dragging over a variable or expression. (If you don't press the **Shift** key, a menu pops up, from which you can choose **Print** or **Display**; that's what we did above.)

- From the command window, you can issue the **`print`** or **`display`** command. As part of the command, you can include a mask that sets the context for printing or displaying. For example,

  ```
  where (x .ne. 0) print x
  ```

  tells Prism to print the values of the array **`x`** in the history region of the command window; only cases in which **`x`** is not equal to 0 are printed.

3.6 Obtaining Performance Data

Prism lets you collect performance data on your program. Collecting and analyzing performance data can help you uncover and correct bottlenecks that slow down a program.

Prism collects separate data for the nodes and the partition manager — referred to as the *subsystems* — and for different computing resources within each subsystem. Note that this is different from the way most other performance analyzers and profilers work; they simply measure CPU time.

To collect performance data, compile your program with the **`-cmprofile`** option.

If you haven't already compiled the program with this option, you can recompile from within Prism, then reload the program. Follow these steps:

1. Choose the **Shell** selection from the **Utilities** menu. Prism creates a window containing a UNIX shell. You can issue UNIX commands from this shell, just as you would from any UNIX shell.

2. Issue this command to recompile **`primes`** with the **`-cmprofile`** option:

   ```
   % cmf -cmprofile -o primes primes.fcm
   ```


Copyright © 1993–1994 Thinking Machines Corporation

3. When the program is recompiled, you can load it from within Prism. Click on **Load** in the tear-off region. A dialog box appears. Find **`primes`** in the list of programs. Click on it to highlight it, and then click on **Select**. Prism then loads the program.

3.6.1 Collecting Performance Data

To collect performance data, you must turn collection on before running the program; collection then remains on until you explicitly turn it off. To turn collection on, click on **Collection** in the tear-off region. The toggle box next to it is filled in.

Then execute your program, as described in Section 3.3. (Note that you shouldn't set breakpoints or otherwise interrupt execution, since this distorts the performance data.) When the program finishes execution, the data is ready for display.

3.6.2 Displaying Performance Data

Prism displays three different levels of performance data:

- Performance statistics for the resources, along with totals for each subsystem.
- Per-procedure performance statistics for a specified resource or subsystem. You can choose either flat or call-graph display of these statistics.
- Per-source-line performance statistics for a specified resource and procedure.

All statistics are displayed as panes in the **Performance Data** window, along with the amount of time (or the percentage of total wallclock time) that each histogram bar represents.

To display the performance data, choose **Display Data** in the **Performance** menu. You see a **Performance Data** window like the one shown in Figure 13.

Copyright © 1993–1994 Thinking Machines Corporation

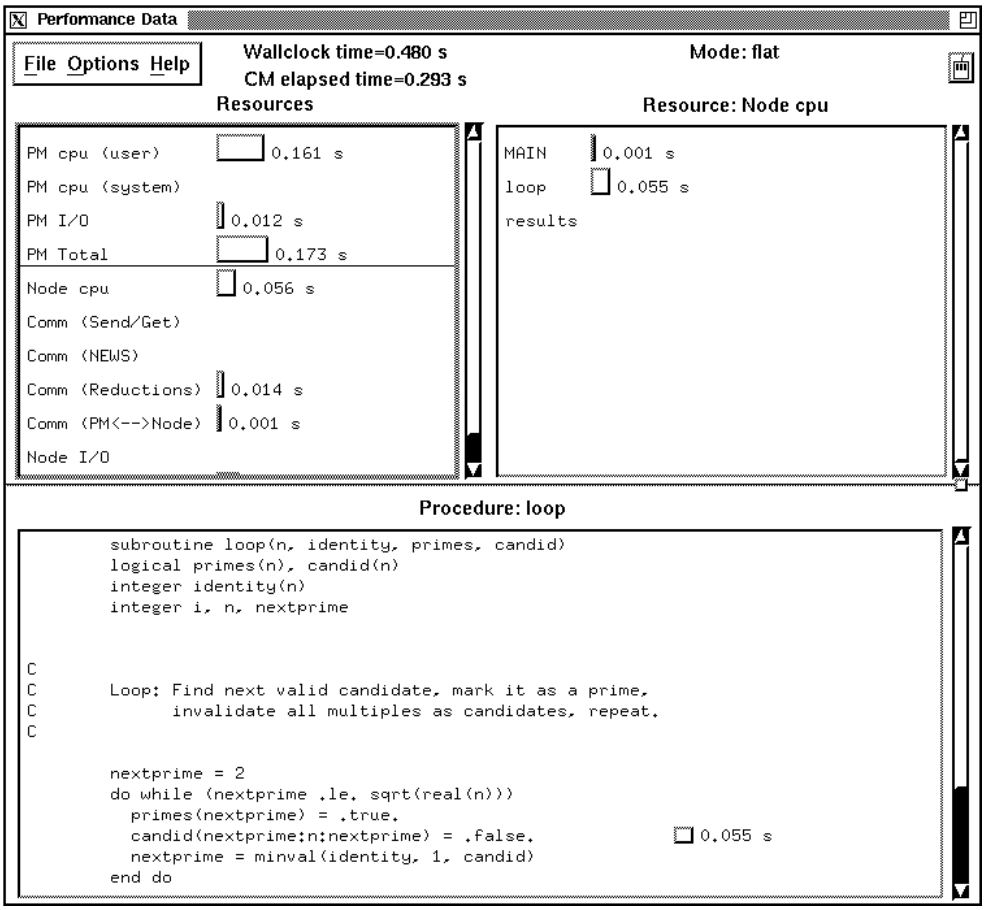

Figure 13. A Performance Data window for `primes`.

At the top left of the **Performance Data** window is a pane that shows the usage of the CM resources. Note that the partition manager (PM) and its resources are measured separately from the nodes and their resources. In the case of the **`primes`** program, most of the activity is on the partition manager.

To the right of this pane is a pane that shows individual procedures' use of a specific resource — in this case, the **Node cpu** resource. Left-click on a resource or subsystem to display its procedure-level data.

Figure 13 shows the procedure data in *flat* mode. In flat mode, the window lists all procedures in the program and each one's total use of the selected resource or subsystem.

Copyright © 1993–1994 Thinking Machines Corporation

You can also display the procedure-level data in *call-graph* mode. In call-graph mode, you see which procedures call which other procedures, and the use of the selected resource or subsystem for each individual call. In this case, procedure **`MAIN`** calls the other procedures listed; each accounts for the listed percentage of time that **`MAIN`** uses of the resource. To switch modes, choose **Mode** from the **Options** menu in the **Performance Data** window.

At the bottom of the **Performance Data** window is a pane that shows per-source-line usage of the selected resource or subsystem. Click on a procedure to show the source lines in that procedure.

To close the window, choose **Close** from the **File** menu.

3.6.3 Other Performance Analysis Features

Prism has additional features you can use in analyzing the performance of your program:

- Prism's *performance advisor* prints out a summary and analysis of the performance data it has collected. It zeroes in on the lines of code that are the heaviest users of the most-used resource. Working on these lines will result in the greatest performance gains in your program. To display the advisor, click on **Advice** in the **Performance** menu, or issue the **`perfadvice`** command.

- You can save your performance data to a file and subsequently reload it. This is useful if, for example, you collect the data in commands-only Prism, and you subsequently want to view it graphically. To save the data, choose **Save Data** from the **Options** menu in the **Performance Data** window or issue the **`perfsave`** command. To load the data, choose **Load Data** from the **Options** menu in the **Performance Data** window or issue the **`perfload`** command.

Copyright © 1993–1994 Thinking Machines Corporation

3.7 Getting Help

There are several ways in which you can get help in Prism:

- The **Help** menu in the menu bar provides help on several major topics. It includes the Help Index, which gives in-depth information about all aspects of Prism.
- The **Help** selection in menus and the **Help** button in windows and dialog boxes provide instructions for using these screen areas. Pressing the **F1** key in a window or dialog box also displays a help screen.
- Command-line help provides information about commands you can issue from the command window.

The Prism help system uses the WorldView viewing software also used in Thinking Machines Corporation's CMview on-line documentation system.

The most important thing to keep in mind when using the help system is that this symbol:

is a *link marker*. Click your left mouse button twice, rapidly, on this marker in a help screen to display more information about a topic. Typically, following the link displays the section of the on-line *Prism User's Guide* or *Prism Reference Manual* where the topic is discussed.

Prism also provides an interface to CMview. Using CMview, you can display any Connection Machine manual in the most recent collection of manuals, and do full-text searches across this entire collection.

To start up CMview, choose **Online Doc** from the **Doc** menu. In a few moments you will see the CMview Welcome screen. For more information on CMview, see Section 2.8.1.

Copyright © 1993–1994 Thinking Machines Corporation

3.8 Customizing Prism

Prism provides several methods for changing the way it looks and acts:

- Choose **Tearoff** from the **Utilities** menu to enter tear-off mode. In tear-off mode, clicking on a menu selection adds a button for that selection to the tear-off region below the menu bar. Clicking on a button in the tear-off region while in tear-off mode removes the button. You can also use the **`tearoff`** and **`untearoff`** commands to do the same thing. Use the **`pushbutton`** command to add a Prism command to the tear-off region.
- Choose **Customize** from the **Utility** menu to display a window you can use to change various default Prism settings — for example, its behavior when there is an error.
- You can also add settings to your X resource database that change Prism defaults.

3.9 Leaving Prism

To leave Prism, choose **Quit** from the **File** menu. Prism displays a dialog box asking you to confirm.

3.10 Commands-Only Prism

Commands-only Prism runs from your UNIX shell; you would use it if you were logged in to a partition manager from a dumb terminal, for example, and X wasn't available to you.

Use the **`-C`** option to the **`prism`** command to start up commands-only Prism. You receive this prompt:

```
(prism)
```

You can issue any Prism command at the prompt; output appears below the command you type. There are, of course, limitations: for example, you can't create visualizers in commands-only Prism, and you cannot obtain the

CMOST Version 7.3, April 1994
Copyright © 1993–1994 Thinking Machines Corporation

graphical display of the performance analysis data. You can, however, use the **print** or **display** command to print out the values of data, and you can use the **perf** command to display a text version of the performance data.

If you are running commands-only Prism from an Xterm, you can use the **-CX** option instead of **-C**. In this case, you *can* create visualizers and display performance data graphically.

To obtain on-line documentation in commands-only Prism, issue the **doc** command. Issuing **doc** displays a menu of available documents (including release notes and bug-update files). Choose the number associated with the document you want to view. In most cases, this displays another menu of the chapters within the document. Choose the number associated with the chapter, and the first screenful of text for that chapter is displayed. Answer **y** in response to the **more?** prompt or simply press the **Return** key to display the next screenful. Answer **n** to return to the menu.

Issue the **quit** command to leave commands-only Prism and return to your UNIX prompt.

3.11 Using Prism with CMAX

You can use Prism's split-screen capability to debug and analyze the performance of a CM Fortran program you have created using the CMAX Converter, comparing it with the original FORTRAN 77 program. See Section 4.6 for more information.

3.12 Node Prism

Node Prism is the version of Prism used for debugging, analyzing the performance of, and visualizing data in CM-5 message-passing programs (that is, programs that use the CMMD library). To invoke Node Prism, issue the **prism** command with the **-node** option.

Copyright © 1993–1994 Thinking Machines Corporation

3.12.1 Using PN Sets

Node Prism operates much like the version of Prism used for data parallel programs, as described earlier in this chapter. A key difference is the introduction of the concept of *PN sets*. A PN set is a predefined or user-defined group of nodes that you can view and operate on as a single entity. Many Prism commands operate on nodes in a PN set as a unit. This ability to aggregate processing nodes lets you analyze your program at different levels. For example, if at some point during execution different subsets of nodes are doing different things, you can group each set of nodes into a different PN set and examine the behavior of each set independently.

The PN Sets Window

From the **Debug** menu, choose the **PN Sets** selection to view the current status of nodes in your partition and to perform many of the actions associated with PN sets. Figure 14 is an example.

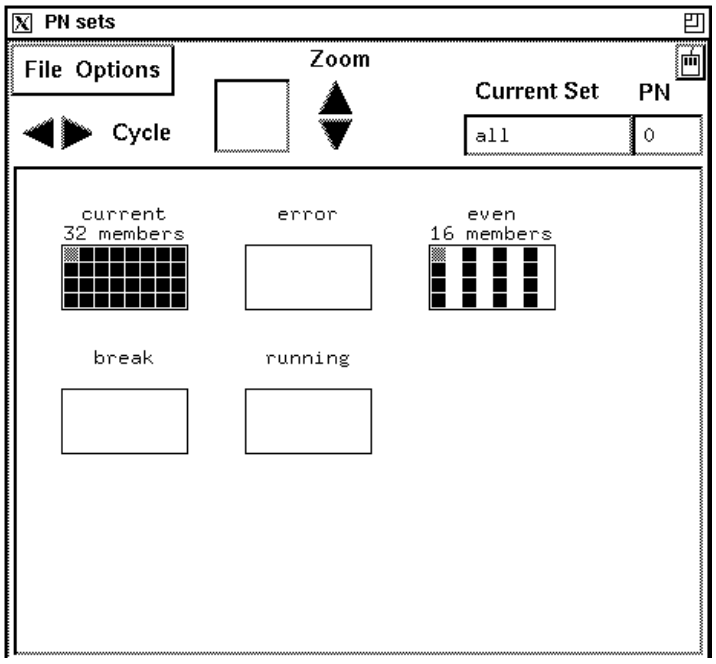

Figure 14. The PN Sets window.

Copyright © 1993–1994 Thinking Machines Corporation

Some of the sets shown in this window are predefined by Node Prism. For example, the **current** PN set is the set to which, by default, Prism commands apply. The **error** PN set is the set of all nodes that have encountered an error. This is an example of a *dynamic* set; Node Prism automatically updates the members of a dynamic PN set, based on the current status of the program.

The set **even** in Figure 14 is an example of a user-defined PN set. It includes the even-numbered nodes (0, 2, 4, 6 . . .) within the 32-node partition. You can define it within the **PN Sets** window, or by issuing this command:

```
define pnset even 0:31:2
```

You can also define PN sets based on the value of a variable — for example, you can create a PN set whose members consist of those nodes in which **x** is not equal to 0. You can subsequently update the membership of this PN set by issuing the **eval pnset** command.

Each PN set has a *current PN*. When the PN set is current, the source code for its current PN is displayed in the source window. In addition, the **Performance Data** window shows performance data for this PN. By default, the current PN is the lowest-numbered PN in the set. You can use the **pn** command to change the current PN.

Using PN Sets in Commands

For many Node Prism commands, you can include a PN-set qualifier that specifies the PN set to which the command is to be applied. For example,

```
stop at 12 pnset all
```

sets a breakpoint at line 12 in all nodes. If you don't include a PN-set qualifier, the command applies to the nodes in the current PN set. For example, if the current PN set is **even**, as defined above, the command

```
stop at 12
```

creates a breakpoint only in the even-numbered nodes.

Copyright © 1993–1994 Thinking Machines Corporation

3.12.2 Displaying Data, the Call Stack, and Performance Information

Node Prism also differs from data parallel Prism in the way it displays the values of data, the call stack, and performance information for the program.

- When displaying a visualizer for a variable, Node Prism adds an axis that represents the nodes in the PN set. Moving along this axis displays the values of the variable on each node.

- Node Prism aggregates all the call stacks for the individual nodes in a message-passing program and displays them graphically in a **Where** graph. Figure 15 shows an example; each box in the figure shows the arguments for the function and the line(s) at which other functions are called. You can display different levels of information in this graph by clicking on the **Zoom** arrows. For example, by zooming in on the graph in Figure 15 you can display a bitmap of cells that shows the nodes that have each function in their call stack.

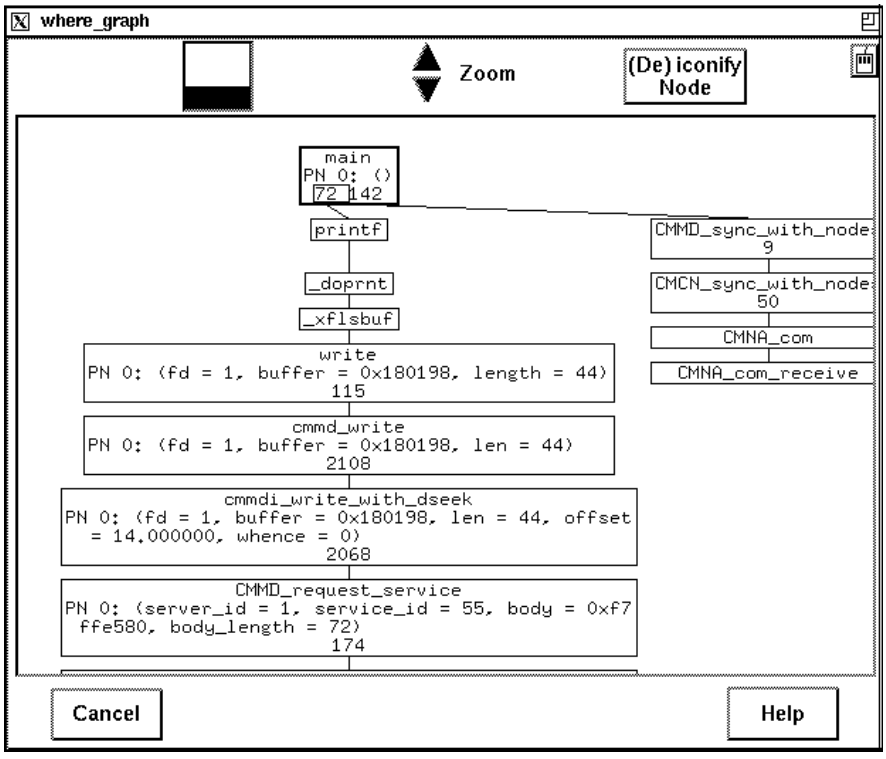

Figure 15. A Where graph.

Copyright © 1993–1994 Thinking Machines Corporation

- Node Prism displays performance data for C* and CM Fortran programs running on a node. Data is displayed by default for the current PN of the current PN set; you can change the displayed data by changing the current PN. The basic approach to displaying the data is the same as that discussed in Section 3.6.2, although the names of the resources are different in Node Prism, and there is no separate partition manager subsystem.

Copyright © 1993–1994 Thinking Machines Corporation