Chapter 5

Debugging a C* Program

Use Prism, the programming environment for the Connection Machine system, to debug your C* program. You can also use Prism to develop and execute your program. This chapter gives a brief overview of Prism. For complete information, see the *Prism User's Guide* and *Prism Reference Manual*. In particular, note that there may be some limitations in Prism's support of C*; for example, it may not recognize all C* syntax in expressions.

NOTE: If you compiled your C* program to run on the nodes and use the CMMD message-passing library, use **pndbx** to debug the program. For information on using **pndbx**, see the *CMMD User's Guide* and the **pndbx** release notes.

Note also that the use of other debuggers is not supported for CM-5 C*.

5.1 Compiling for Prism

To use Prism for debugging your C* program, compile the program with the **-g** or **-cmdebug** option. To use it for performance analysis (see Section 5.7), compile with the **-cmprofile** option.

5.2 Starting Prism

Prism has two modes:

- Graphical Prism operates on terminals or workstations running the X Window System.
- Commands-only Prism lets you operate on any terminal, but without the graphical interface.

5.2.1 Graphical Prism

Before starting Prism, 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 Prism, issue the command **prism** at your UNIX prompt. To load an executable program automatically into Prism, specify its name on the Prism command line. For example:

```
% prism primes.x
```

This displays the main Prism window, as shown in Figure 1.

5.2.2 Commands-Only Prism

To start commands-only Prism, issue the **prism** command with the **-C** option:

```
% prism -C
```

After an introductory message, you receive this prompt:

```
(prism)
```

You can issue any Prism command at the prompt. The rest of this chapter focuses on graphical Prism; however, all of the functionality of graphical Prism is available by issuing commands from commands-only Prism, except for features that require graphics.

5.3 Using Prism

Figure 1 shows the main Prism window, with a source file loaded.

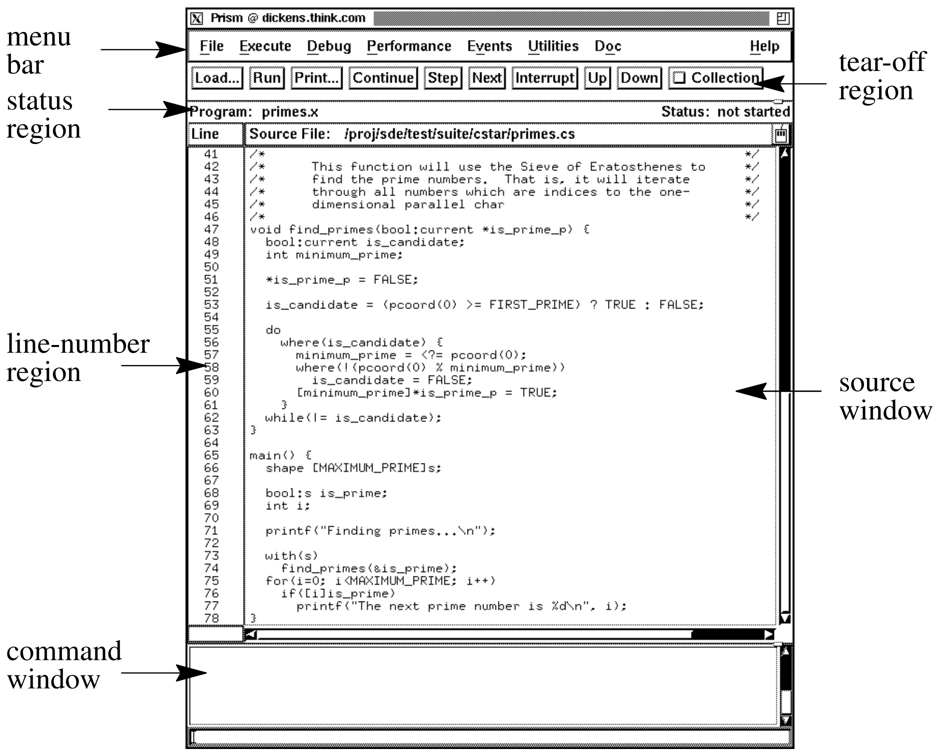

Figure 1. The main Prism window.

Clicking on items in the **menu bar** at the top of the Prism window displays pull-down 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. Clicking on a button in the tear-off region is equivalent to selecting the menu item or issuing the command.

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

The **source window** displays the source code for the executable program. You can scroll through the source code and display any of the source files used to compile the program.

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.

The **command window** at the bottom of the main Prism window displays messages and output from Prism. You can also type commands in the command window rather than use the graphical interface.

5.4 Loading and Executing Programs

As mentioned above, you can load an executable program when you start up Prism. You can also load it after Prism has started.

Once the program is loaded, you can run it or step through it, and interrupt execution at any time. You can also attach to a running program or associate a core file with a program.

5.5 Debugging

Prism lets you perform standard debugging operations such as:

- Setting breakpoints and traces
- Displaying and moving through the call stack
- Examining the contents of memory and registers

It also contains an *event table* that provides a unified method for controlling the execution of a program. For example, using the event table, you can specify that execution is to stop at line 33, and Prism is to print out the values of parallel variable **x**.

5.6 Visualizing Data

In debugging a C* program, it is often important to obtain a visual representation of the values of a parallel variable's elements. In Prism, you can create *visualiz-*

ers to display these values graphically. Prism visualizers provide a variety of representations, including:

- *Text*, where the values are shown as numbers or characters.
- *Colormap*, where each value is mapped to a color, based on a range of values and a color map you specify. (This representation is available only on color workstations.)
- *Dither*, where groups of elements are assigned different numbers of black and white pixels, depending on their values. This gives an impression of gray shading on a black-and-white display.

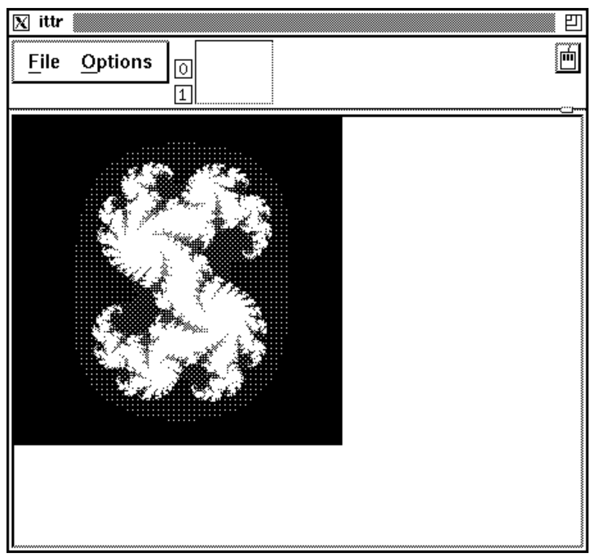

Figure 2. A dither visualizer.

- *Threshold*, where each value is mapped to either black or white, based on a cutoff value that you can specify.
- *Surface*, which renders the 3-dimensional contours of 2-dimensional data.

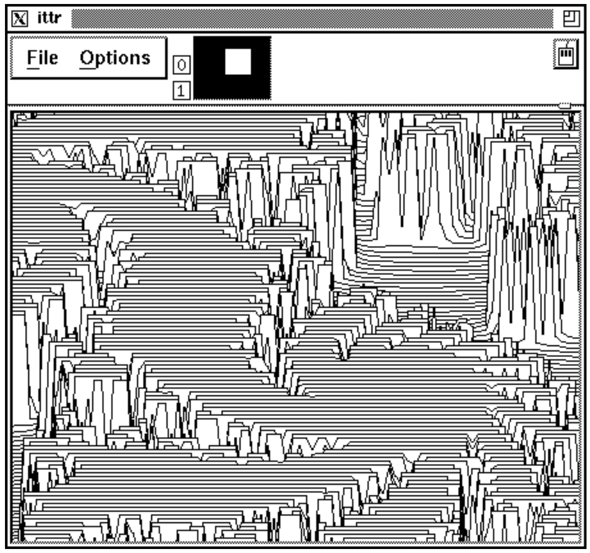

Figure 3. A surface visualizer.

- *Graph*, which displays values as a graph, with the index of each element plotted on the horizontal axis and its value on the vertical axis. A line connects the points plotted on the graph.

If the data doesn't all fit in the display window, you can pan through it. You can also increase the size of the fields being displayed; this gives the effect of zooming in on an area.

You can specify that the visualizer is to update whenever the program stops execution. And you can take a snapshot of the visualizer, so that you can compare it with later updates.

5.6.1 Visualizing Parallel Objects

When you visualize a pointer to a parallel object (for example, a parallel variable or parallel structure), you obtain three pieces of information:

- the CM memory address of the object being pointed to
- the address that represents the object's shape
- a memory stride that indicates how many bytes are between the starting addresses of successive elements of the object on each physical processor

Here are examples from a command-line Prism session:

```
(prism) whatis p
parallel int *p;
(prism) print p
pp`foo`p = [addr=0xa0000088; shape=0x3c018; stride=4]
(prism) whatis c
parallel struct foo *c;
(prism) print c
pp`foo`c = [addr=0xa0000000; shape=0x3c018; stride=8]
```

NOTE: CM-5 C* lets you obtain the memory address and stride information via functions in your program; this in turn lets you construct a pointer to a parallel variable. See the *C* Programming Guide* for more information.

5.7 Analyzing a Program's Performance

Prism provides performance data essential for effectively analyzing and tuning C* programs. The data includes:

- processing time on the partition manager
- processing time on the nodes
- time spent doing various forms of communication

The performance data is displayed as histogram bars and percentages for each resource. For each resource, you can also obtain data on usage for each procedure and each source line in the program. You can save the performance data in a file and redisplay it at a later time.

In addition, a performance advisor provides information about and analysis of the data Prism collects.