Chapter 6

Obtaining Performance Data

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

Section 6.1 is an overview of obtaining performance data in Prism. To learn:

- **How to write and compile your program to obtain performance data,** see Section 6.2.
- **How to obtain the most accurate performance data,** see Section 6.3.
- **How to collect performance data,** see Section 6.4.
- **How to display performance data,** see Section 6.5.
- **How to interpret performance data,** see Section 6.6.
- **How to save a file of performance data and reload it into Prism,** see Section 6.7.

See Section 10.7 for a discussion of performance analysis in Node Prism.

6.1 Overview

Prism helps you determine where your C* or CM Fortran program is spending its time, and why.

To determine where your program is spending its time, Prism provides data at the level of the entire program, individual procedures within the program

Copyright © 1991–1994 Thinking Machines Corporation

(with both call-graph and flat displays), and individual source lines within procedures. This allows you to zero in on the lines that have the greatest impact on a program's performance.

To determine why a procedure or a source line is a bottleneck in your program, Prism provides data on a program's use of several different computing resources, not just CPU time. For example, the code may be doing a lot of send/get communication or I/O. Providing data on the code's use of these resources makes it easier to determine how, or if, the code's performance can be improved.

Prism aggregates performance data separately for the front end (or partition manager) and for the CM (or the nodes); these are referred to as the *serial subsystem* and *parallel subsystem*, respectively. This is necessary because both subsystems contribute independently to a program's execution time. (Note that there are not separate subsystems for performance data in Node Prism.)

In addition to displaying the data, Prism provides a *performance advisor* that gives an interpretation of the data. See Section 6.6.2 for more information on it.

6.2 Writing and Compiling Your Program

Performance data is available for C* and CM Fortran programs. To collect performance data, you must compile your program with the **`-cmprofile`** option. Don't use the **`-O0`** option to turn off optimization for a CM-2/200 C* program.

If your program calls an individual routine not compiled with the **`-cmprofile`** option (such as a routine from a CM library like CMSSL), serial-subsystem data will be available for that routine. Specific information on parallel-subsystem resources will not be available; summary information is available only for routines written in CM Fortran Version 2.1. A routine not compiled with **`-cmprofile`** cannot call a routine compiled with **`-cmprofile`**.

Performance data is available if you compile your CM Fortran or CM-5 C* program with the **`-cmsim`** option and run it on a Sun-4. The timings will effectively be those for a one-node CM-5.

Version 2.0, April 1994
Copyright © 1991–1994 Thinking Machines Corporation

6.2.1 Including Timers within Your Program

Prism collects performance data using the CM timing utility; see the *CM Fortran User's Guide* or *C* User's Guide* for a description of this utility. By default, Prism uses timers 5–63, leaving timers 0–4 available for use within the program itself. If you want to use more than five timers, use the environment variable **CMPROF_N_USER_TIMERS** to specify the number. For example, if you want to use 10 timers, set the variable as follows (for the C shell):

```
% setenv CMPROF_N_USER_TIMERS 10
```

This reserves timers 0–9 for use within your program. Program execution becomes less efficient, and the performance data becomes more distorted, as you use more timers, leaving fewer available for Prism.

NOTE: If you fail to set this environment variable, and thereby try to use timers that Prism itself is using, the resulting performance data will be incorrect and possibly bizarre (for example, with utilization values well in excess of 100 percent). You receive a warning from Prism if you try to use timers that Prism is using.

6.3 Obtaining the Most Accurate Performance Data

This section gives some hints on how to obtain the most accurate performance data in Prism.

Note these general points:

- Collecting performance data slows execution of the program. The exact degree to which this occurs depends on the program. In general, programs that work on small amounts of data in each node will be affected more by this performance-collection overhead; the overhead will be negligible in programs that work on large arrays relative to CM-5 partition size. Prism corrects for this effect when it presents its data. If you are using your own timers to measure performance during a run in which Prism is also collecting data, however, you need to be aware that this effect will inflate the values in these timers.

Copyright © 1991–1994 Thinking Machines Corporation

- Interrupting your program while collecting performance data (for example, by stopping at a breakpoint and printing values) will distort the data.
- As described in more detail below, utilization percentages are affected by system load and timesharing. To decrease the variability of these percentages between runs of the program, use a lightly loaded system.

Note these points with regard to serial-subsystem data:

- Running on a heavily loaded front end or partition manager may inflate somewhat the time allocated to its CPU resource. Once again, for most accurate results, run your program on a front end or partition manager that is lightly loaded.
- Paging on the front end or partition manager may cause some discrepancies in the data for the serial subsystem; these discrepancies will be greater on smaller programs.

6.4 Collecting Performance Data

To collect performance data, you must turn collection on before running the program. Collection remains on until you explicitly turn it off.

- **From the menu bar:** Choose **Collection** from the **Performance** menu. (This selection is also available by default in the tear-off region.) **Collection** toggles the collection of performance data. Performance collection is off when the toggle box to the left of the menu selection is not filled in; this is the default. Choosing **Collection** turns it on, and the toggle box is filled in. To turn it off, choose **Collection** when the toggle box is filled in.
- **From the command window:** Issue the **`collection on`** command to turn collection on; issue **`collection off`** to turn it off. Issuing the **`collection`** command also affects the state of the toggle box in the **Collection** menu selection.

On a CM-2 or CM-200, Prism automatically turns safety off if you run a program with collection turned on; it turns safety back on if you subsequently run the program with collection turned off.

Version 2.0, April 1994
Copyright © 1991–1994 Thinking Machines Corporation

6.4.1 Collecting Performance Data Outside of Prism

You can also collect performance data by setting environment variables, without entering Prism. This is convenient if you can't enter Prism for some reason (for example, because the CM is only accepting batch jobs).

To turn on collection of performance data, set the environment variable **CMPROFILING** to **t**:

```
% setenv CMPROFILING t
```

To turn collection off, set the environment variable to **f**.

To specify the program on which data is to be collected, set the environment variable **CMPROFILING_EXECUTABLE_FILENAME** to the name of the executable program. For example:

```
% setenv CMPROFILING_EXECUTABLE_FILENAME a.out
```

To specify the file to which the performance data is to be sent, set the environment variable **CMPROFILING_DATA_FILENAME** to the name of the file. For example:

```
% setenv CMPROFILING_DATA_FILENAME perf.data
```

You can load this file into Prism for examination at a later time; Section 6.7 explains how.

6.5 Displaying Performance Data

To display performance data, the program must have finished execution. Choose **Display Data** from the **Performance** menu. A window appears, containing the data. Figure 33 shows an example.

Copyright © 1991–1994 Thinking Machines Corporation

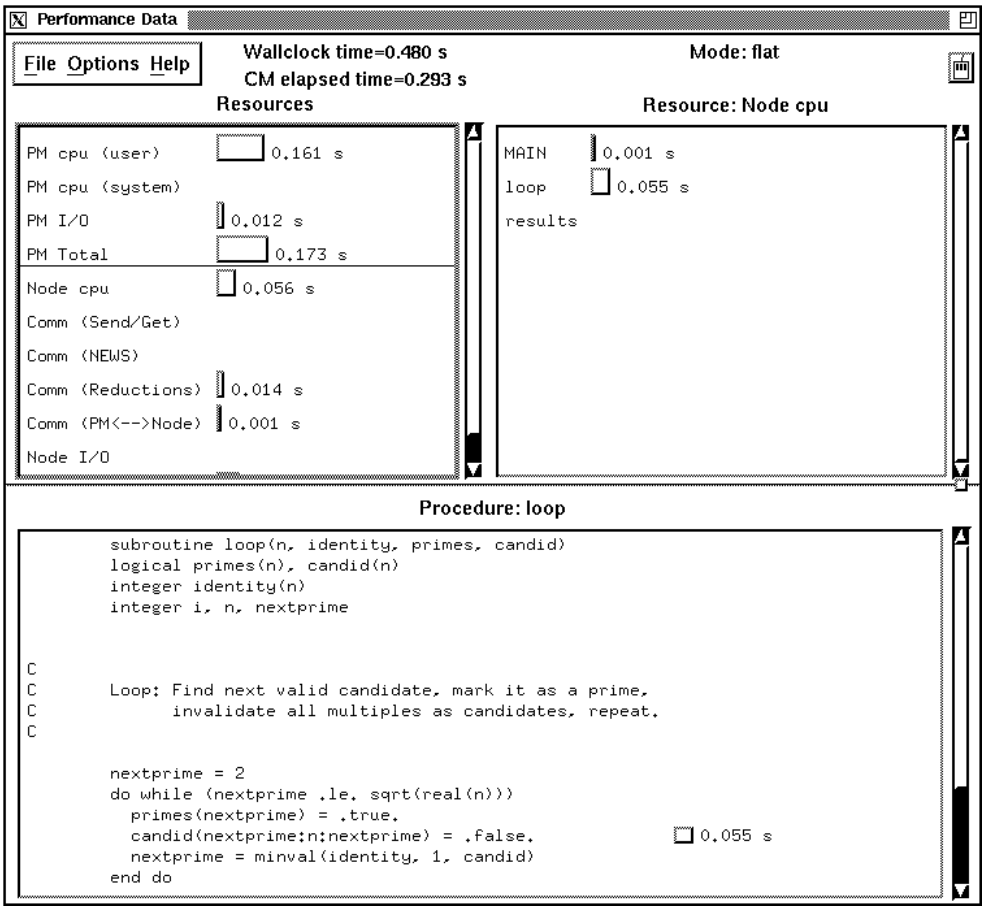

Figure 33. The Performance Data window.

At the top of the window are two values that represent the program's execution time:

- **Wallclock time** represents the actual elapsed wallclock time from when the program started to when it finished; thus, it is sensitive to effects of timesharing and front-end or partition-manager load.

- **CM elapsed** represents the time during which the CM or the nodes were allocated to the program; this measurement takes into account timesharing on the CM, and should be reasonably consistent no matter what the load is on the front end or partition manager. Thus, this measurement gives an indication of what your program's run time would be on a dedicated system.

Version 2.0, April 1994
Copyright © 1991–1994 Thinking Machines Corporation

The **Performance Data** window contains three levels of performance data:

- Performance statistics for the resources that Prism measures, along with totals for each of the two subsystems.
- 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 histograms in panes within the **Performance Data** window, along with the amount of time or the percentage of wallclock time that each histogram bar represents. If the program didn't use the resource, the histogram bar does not appear. (Occasionally, however, a resource will show a utilization of 0% because of rounding.)

By default, the window displays the number of seconds used by the resource next to each histogram bar. Choose **Units** from the **Options** menu to change this. You have these choices:

- Choose **Seconds** (the default) to display the actual time, in seconds, that the histogram bar represents.
- Choose **Microseconds** to display the actual time in microseconds.
- Choose **Utilization** to display the percentage of the total wallclock time that the histogram bar represents.

NOTE: As of Prism Version 2.0, the **Utilization** percentages are based on wallclock time; this means they are sensitive to effects of load and timesharing. These percentages may therefore vary considerably from run to run of the program; the lighter the load, the higher the percentages will be. If you choose the **Utilization** display option, a popup message warns you of this variability.

Although the actual percentages may vary, the relative percentages for most resources will generally remain the same. For example, if **Node cpu (user)** uses twice as much time as any other resource on one run of the program, it should use twice as much time as any other resource on the next run. I/O time, however, is an exception; it can vary non-proportionally to the other resources from run to run.

Copyright © 1991–1994 Thinking Machines Corporation