Paradyn Parallel Performance Tools

Tutorial

Release 2.1
May 1998

Paradyn Project
Computer Sciences Department
University of Wisconsin
Madison, WI 53706-1685
paradyn@cs.wisc.edu

Table of Contents

List of Figures

1 PRELIMINARIES

This document[1] covers the basics for using Paradyn: how to prepare an application for Paradyn (if necessary), run it, view its performance data, and run the Performance Consultant to automatically find performance bottlenecks in the application. Several simple example application programs come with the binary distribution of Paradyn. You can obtain Paradyn and the test programs (binaries and sources) by anonymous ftp to `grilled.cs.wisc.edu`. For more information on obtaining and installing Paradyn, including setting necessary environment variables, see the ***Paradyn Installation Guide***.

The simple test program used as an example in this document (**`potato`**) is a PVM program, requiring that PVM be installed on your system: if PVM is not installed, the basic form of the tutorial can be followed with any other subject application, such as the sequential test program (`bubba`) or an MPI program (such as `ssTwod`) where available.

Details of this tutorial apply to a program running on a SPARC Solaris platform, and small differences will be observed on other platforms. Note, however, that a Paradyn front-end is not yet available for WindowsNT, therefore while WindowsNT applications can be run and monitored by Paradyn, this currently requires a Paradyn front-end controlling the session from a Unix workstation.

This document will not cover all of the features in Paradyn. It is intended to guide you through a start-to-finish session with Paradyn, using some of the more common features. For a complete description of the features in Paradyn, see the ***Paradyn User's Guide***.

Further information, including technical papers, manuals and source code, is available from the Paradyn Project web pages **`http://www.cs.wisc.edu/~paradyn`**.

2 PREPARING AN APPLICATION

To run any of the simple programs that come with the binary distribution of Paradyn, this step can be skipped. On most platforms Paradyn works with unmodified application binaries, automatically loading the runtime instrumentation library (`libdyninstRT`) as necessary, however, some platforms still require an extra re-linking step to statically link subject applications with Paradyn's runtime instrumentation library (and also possibly to de-limit the application code of interest with special code block markers). For details, see the ***Paradyn User's Guide***, or any of the Makefiles provided with the source code for the test programs distributed with Paradyn.

1. *Note that some of the color figures in this document may be unclear when printed in gray-scale.*

3 RUNNING THE APPLICATION

Skip straight to Section 3.2 if you have a non-PVM application, otherwise you first need to start PVM itself before running Paradyn.

3.1 Start PVM

An example of starting PVM on the host `chocolate` is provided below:

```
% pvm
pvm> add cham
pvm> add beaufort
pvm> conf
3 hosts, 1 data format
      HOST     DTID     ARCH   SPEED
 chocolate    40000     SUNMP   1000
      cham    80000     SUNMP   1000
  beaufort    c0000     SUNMP   1000
```

The Paradyn daemon (`paradynd`) and the binary of the application (**`potato`**) must also be copied to the directory where you keep your PVM binaries (usually `$HOME/pvm3/bin/$PVM_ARCH` or `$PVM_ROOT/bin/$PVM_ARCH`):

```
% cp paradynd $HOME/pvm3/bin/$PVM_ARCH
$ cp potato $HOME/pvm3/bin/$PVM_ARCH
```

3.2 Start Paradyn and define the application process

The next step is to run Paradyn. This is done by entering the following command:

```
% paradyn
```

Paradyn will start running and display the Paradyn Main Control window (Figure 1) and the base Where Axis window (Figure 2). The status line in the Paradyn Main Control window (labeled “UIM status”) indicates that Paradyn’s user interface manager is ready. This means that Paradyn is now ready to loaded and run the subject application program.

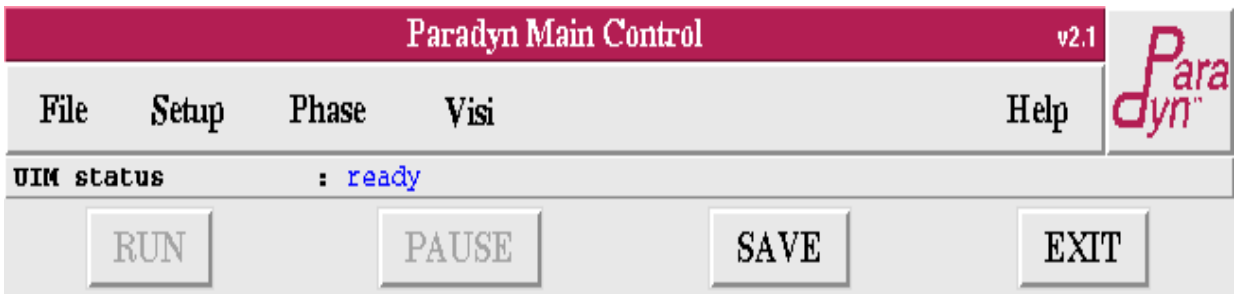

Figure 1: Paradyn Main Control window

To describe an application to Paradyn select **Define A Process** from the **Setup** menu. This will cause a dialog to appear that will allow you to specify the parameters that are necessary for Paradyn to start your application process. This dialog is shown in Figure 3. To describe the appli-

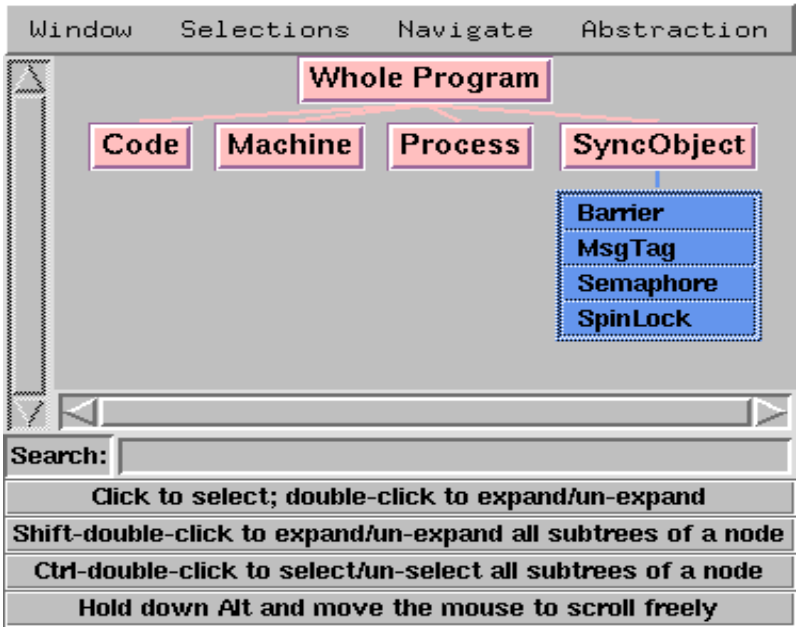

Figure 2: Paradyn base Where Axis

cation and its environment to Paradyn, the following should be specified in the **Define A Process** dialog:

1. **User:** The login name on the host on which Paradyn will start the application process. In this example we left the **User** field blank, which means that the login will have a value of the user's current login name.
2. **Host:** The host on which Paradyn will start the application process. A blank value will default to the current host (the one on which Paradyn is running).
3. **Directory:** If the host on which the application is to be started is different from the one on which the Paradyn process is running, then the current directory on the remote machine is the home directory of the user specified in the **User** entry. The **Directory** field allows you to specify a directory to change to before Paradyn starts the application process. In this example, Paradyn will change to `~/pvm3/bin/SUNMP` before starting **`potato`**.
4. **Command:** This entry takes the unix command that will start the application program. In this example we have entered "`potato 5 1000000`", which specifies the executable file (**`potato`**) with two command line arguments: the number of processes (`5`), and the number of messages each process will send (`1000000`).
5. **Daemon:** This option allows you to specify which version of the Paradyn daemon to run. Since this is a PVM application, the `pvmd` daemon is selected.

Once the fields of the **Define A Process** window have been filled in, click on the **Accept** button, and Paradyn will start your application process. This step can take anywhere from several seconds to several minutes, depending on the size of the application.

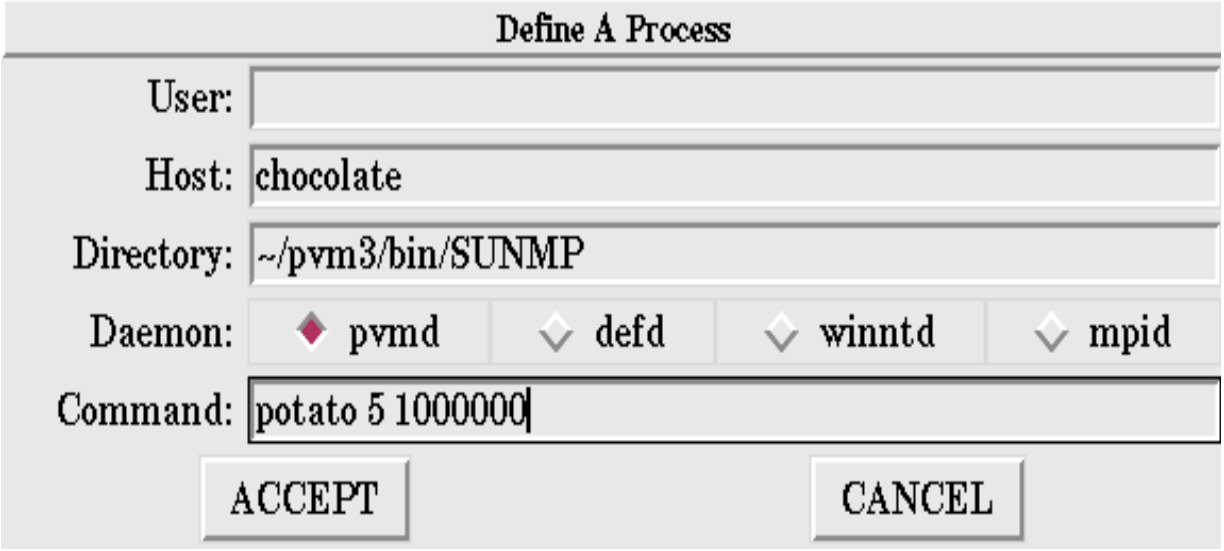

Figure 3: The Define A Process window

3.3 Start the application process

After an application has been defined, the Paradyn main window will contain more status lines, and the Where Axis will contain more entries. The new status lines provide information about Paradyn and your application process. These are shown in Figure 4 (which shows the Paradyn Main Control window after it has started running the application).

The following status lines are for the application process:

1. **Application name:** The name of the application program (**`potato`**), the name of the machine (chocolate), the name of the user (self), and the name of the daemon (pvmd)
2. **Processes:** A list of the process IDs of all the processes in the application. In this example, there is one pid (7657) corresponding to the process started on host chocolate. When **`potato`** runs it will spawn processes on the other hosts (cham and beaufort).
3. **Application status:** The current status of the application program (either RUNNING, PAUSED, or EXITED).
4. **chocolate, beaufort, cham:** Status lines for each host. Once the application starts running these will display the status of each host (running, paused, or exited).

The new status line for the Paradyn process (**Data Manager**) displays the state of Paradyn's Data Manager.

Now that Paradyn has had a chance to look over your program, it is able to add entries to the Where Axis. The new entries in the Where Axis correspond to resources that can only be obtained when the application process has been defined and started. These new entries include modules and procedures in the **Code** hierarchy, and process IDs in the **Process** hierarchy. Figure 5 shows the new Where Axis with these new resources added. The Process hierarchy contains five new processes (one for each PVM spawned process), and the Code hierarchy contains one new entry corresponding to a source code module.

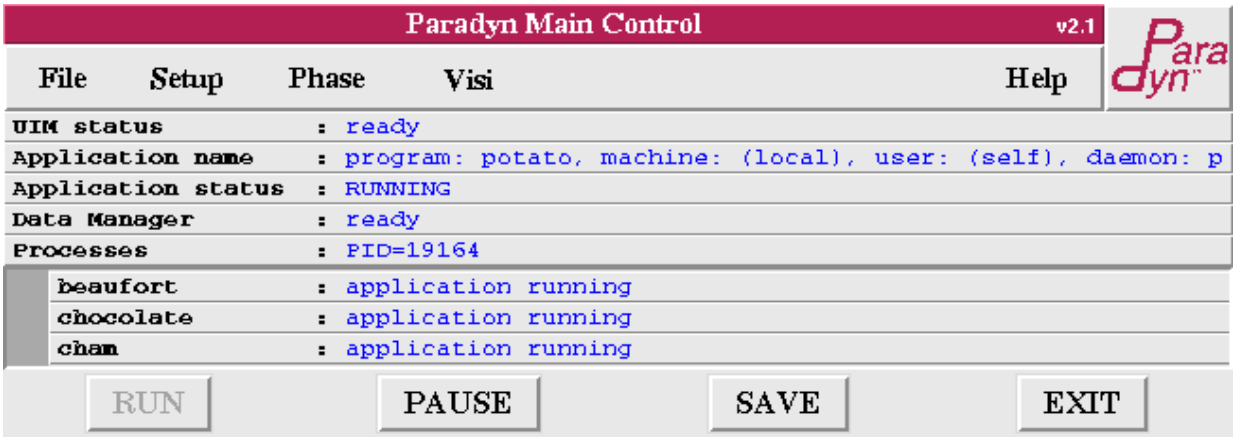

Figure 4: Paradyn Main Control window after application process is started

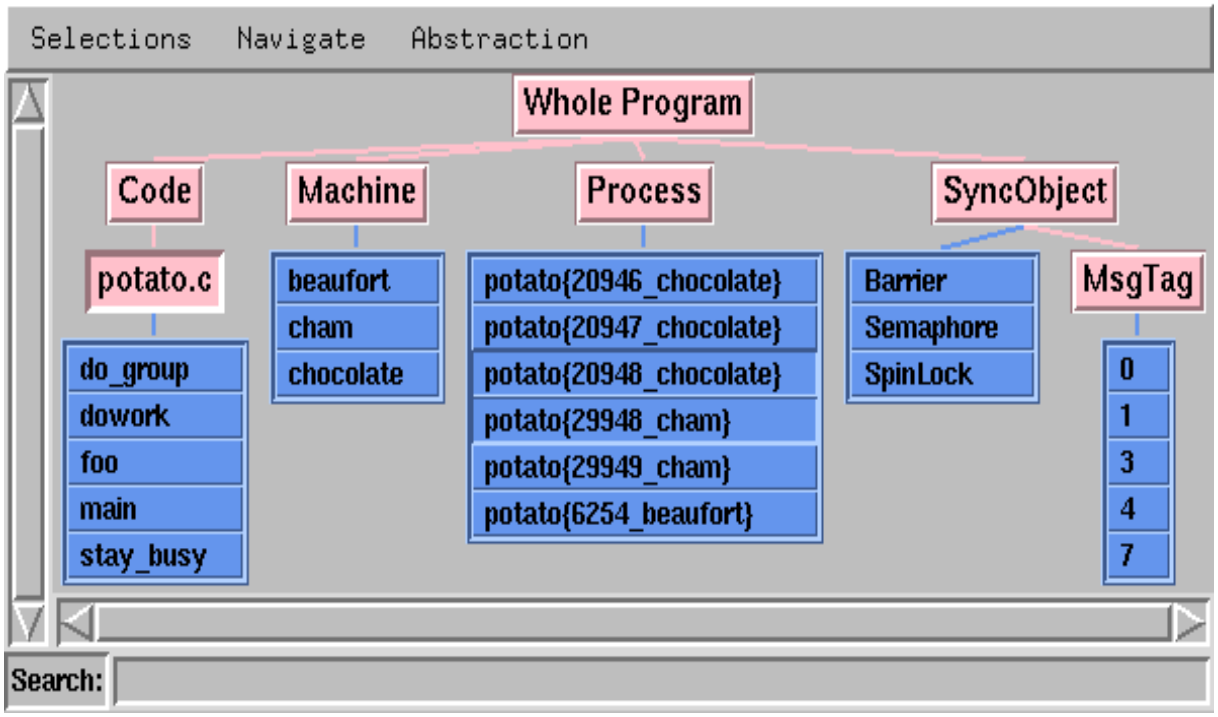

Figure 5: Where Axis after the application process is started

At this point, Paradyn is ready to start running the application. You can now select the **RUN** button from the Paradyn Main Control window to start executing **`potato`**, or alternatively first define some performance measurements and/or views before running it (as described in the following sections). Once execution has commenced, the **PAUSE** button can be used to temporarily halt it and **RUN** will resume execution. Note, however, that execution can only be resumed from the current point and not from the start (without exiting and restarting Paradyn).

4 VIEWING PERFORMANCE DATA

Before you run the application process, you may want to start a visualization process[2]. For this application, we will start a time-histogram visualization to view CPU utilization and synchronization blocking time for the application. In this section, we describe how to start a visualization process, and how to choose the set of metrics and parts of the program that a visualization will display.

4.1 Starting a visualization process

To start a visualization process, select the **Visi** option from the Paradyn main window menubar. This will open the **Start A Visualization** dialog that allows you to choose a type of visualization and a phase for the data. Figure 6 shows this dialog with a Histogram visualization selected for the Global Phase (Section 6 will discuss phases).

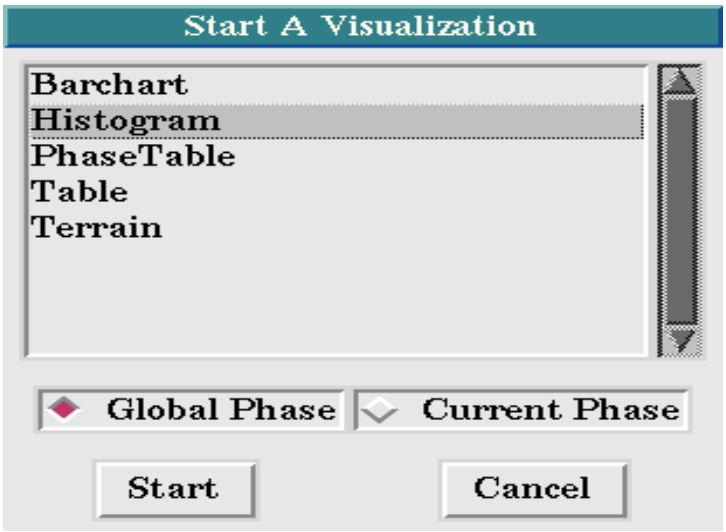

Figure 6: Selecting a Histogram visualization

Once the visualization selection has been made, click on the **Accept** button and Paradyn will display a metrics menu. This menu, shown in Figure 7, allows you to select the set of metrics to be displayed by the visualization. In this example, we have selected *sync_wait* (synchronization blocking time) and *cpu* (CPU time).

To choose the parts of the program for which the metric will be collected, you select resources by clicking on nodes in the Where Axis. A focus is a location in the application for which metric data can be collected. For example, if you select the nodes **potato{20948_chocolate}** and **potato{29948_cham}** from the Process hierarchy, you limit data collection to these two processes (20948 and 29948). If you select **potato.c** from the Code hierarchy, you limit data collection to module `potato.c`. Figure 5 shows the Where Axis with these nodes selected.

Paradyn combines selections from each of the resource hierarchies to create a *focus*, each selection further restricts the scope of data collection. If you had made the previous process and

2. Visualization processes do not have to be started now, but doing so before the program starts running will guarantee that you will get data for the complete execution of the application.

module selections, then you limit data collection to activity in module `potato.c` only in processes 20948 and 29948. This selection corresponds to two foci: the first focus is when the process 20948 is running in module `potato.c`; the second focus is when process 29948 is running in module `potato.c`.

If no Where Axis nodes are selected then Paradyn uses the default **Whole Program**.

Once you have made your selections, click on the Accept button on the metrics menu. Paradyn will then try to enable data collection for your selection. The selection is expanded to be the cross-product of metric-focus pairs from the list of metrics and foci selected. For example, if the metrics **CPU** and **sync_wait**, and the resource nodes **potato{20948_chocolate}**, **potato{29948_cham}**, and **potato.c** were selected, then Paradyn would try to enable four metric-focus pairs:

- *CPU* time for process 20948 when it is running in module `potato.c`.
- *CPU* time for process 29948 when it is running in module `potato.c`.
- *sync_wait* time for process 20948 when it is running in module `potato.c`.
- *sync_wait* time for process 29948 when it is running in module `potato.c`.

If at least one metric-focus pair was successfully enabled, Paradyn will start the visualization process and start sending performance data values to the visualization. If there are any metric-focus pairs that could not be enabled, Paradyn will display a message listing those pairs, and re-display the metrics menu for you to modify your selection. If this occurs, and you do not want to try enabling any other metric-focus pairs, you can choose the **CANCEL** button on the metrics menu.

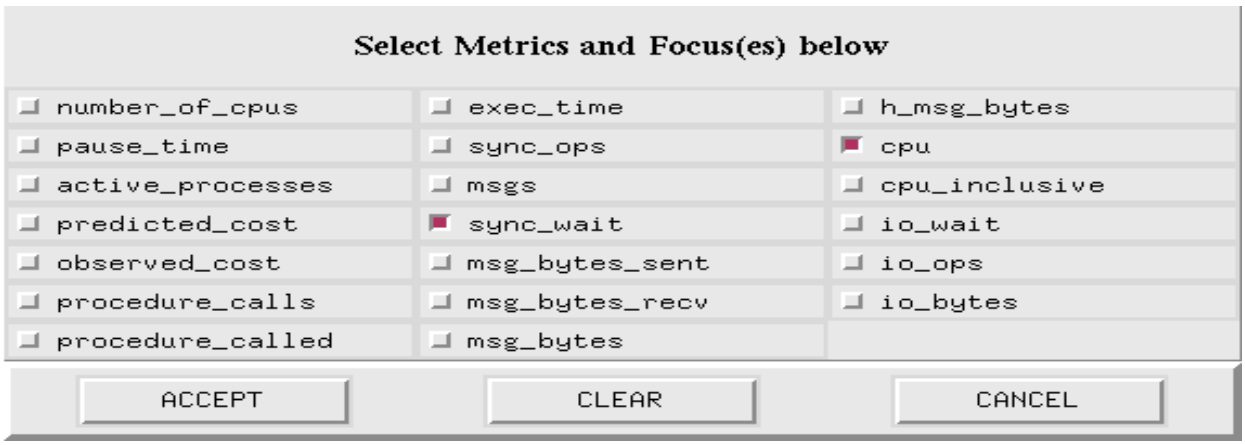

Figure 7: Metrics menu with “cpu” and “sync_wait” selected

The time-histogram shown in Figure 8 is the result of selecting the metrics “sync_wait” and “cpu” from the metrics menu and **potato{20948_chocolate}**, **potato{29948_cham}**, and **potato.c** from the Where Axis.

Once the time-histogram is created, click on the **RUN** button from the Paradyn main window to start the application process. Performance data will then be sent by Paradyn to the time-histogram. The time-histogram contains several menu options for changing the display of the performance data and for changing the set of performance data that is currently being displayed. These options are described in detail in the ***Paradyn User’s Guide***.

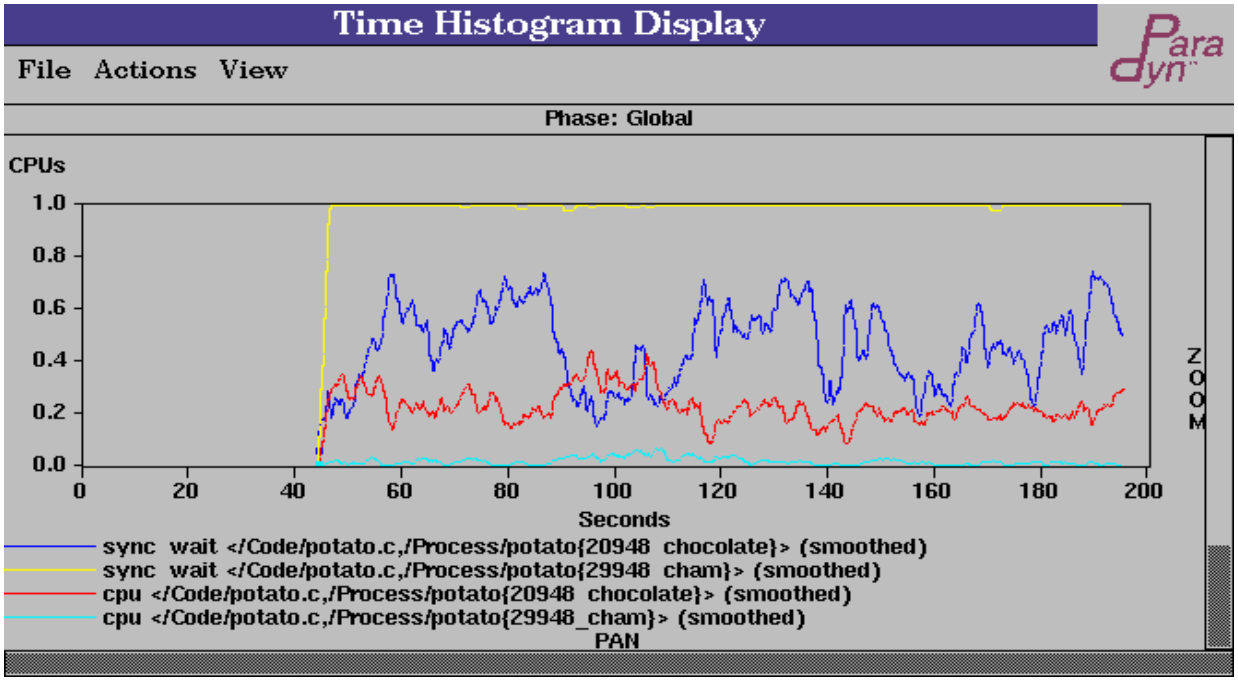

Figure 8: Histogram of global phase with “cpu” and “sync_wait” for two foci

5 RUNNING THE PERFORMANCE CONSULTANT

The Performance Consultant is the part of the Paradyn tool that performs a search for performance bottlenecks. It automatically enables and disables instrumentation for specific metric-focus pairs as the search progresses. The Performance Consultant starts looking for course-grained performance problems and then iteratively tries to refine the search to isolate the performance bottleneck to a specific location in the application's execution. This location is specified as a point in a three dimensional search space defined by a Why Axis, Where Axis, and When Axis.

5.1 The Performance Consultant window

The Performance Consultant is started by selecting the **Performance Consultant** option from the **SetUp** menu on the Paradyn main window. Figure 9 shows the Performance Consultant window. We briefly discuss the parts of the Performance Consultant window below:

1. **Searches** Menu: Allows you to view search history graphs from different phases.
2. Status line: The status line at the top of the window indicates the phase for which the search is defined (in this example, the search is defined for the **Global Phase**).
3. Search Text Output: This area is used by the Performance Consultant to print status messages about the state of the search
4. Search History Graph: This is a graphical representation of the state of the search. Nodes correspond to different points in the search space, and arcs correspond to different refinements that have been made. Figure 9 shows only the initial node, **TopLevelHypothesis**.
5. Buttons: These allow you to start or pause the search.
6. Search History Graph Key: The bottom portion of the window describes how to interpret the color of the nodes and edges in the search history graph, and how to navigate around the window.

5.2 Starting the search

The search can be started by clicking on the **Search** button in the Performance Consultant window. As the Performance Consultant search proceeds, status information will be printed to the window, and the search history graph will be updated to reflect the current state of the search. A Performance Consultant search is either defined over the entire run of the application (the global phase), or over a specific phase of the application's execution. In this example we selected the **Search** button in the Performance Consultant window to start a global phase search. Figure 10 shows the Performance Consultant window during the bottleneck search.

By looking at the Search History Graph, we can see how the Performance Consultant iteratively refines its search to isolate the bottleneck. The first hypothesis the Performance Consultant tests is whether there is a bottleneck in the whole program, if this is true, then it starts refining the search. Each level in the search history graph represents a refinement that was made in the search process. Refinements are only made on hypotheses that test true, and are used to further isolate the bottleneck to a particular part of the application's execution. In general the results of the search can be obtained by following the blue nodes from the root of the search history graph to a leaf node. Also, by clicking the middle mouse button on any node in the search history graph, you can see a text string representation of the hypothesis associated with any node in the graph. This string

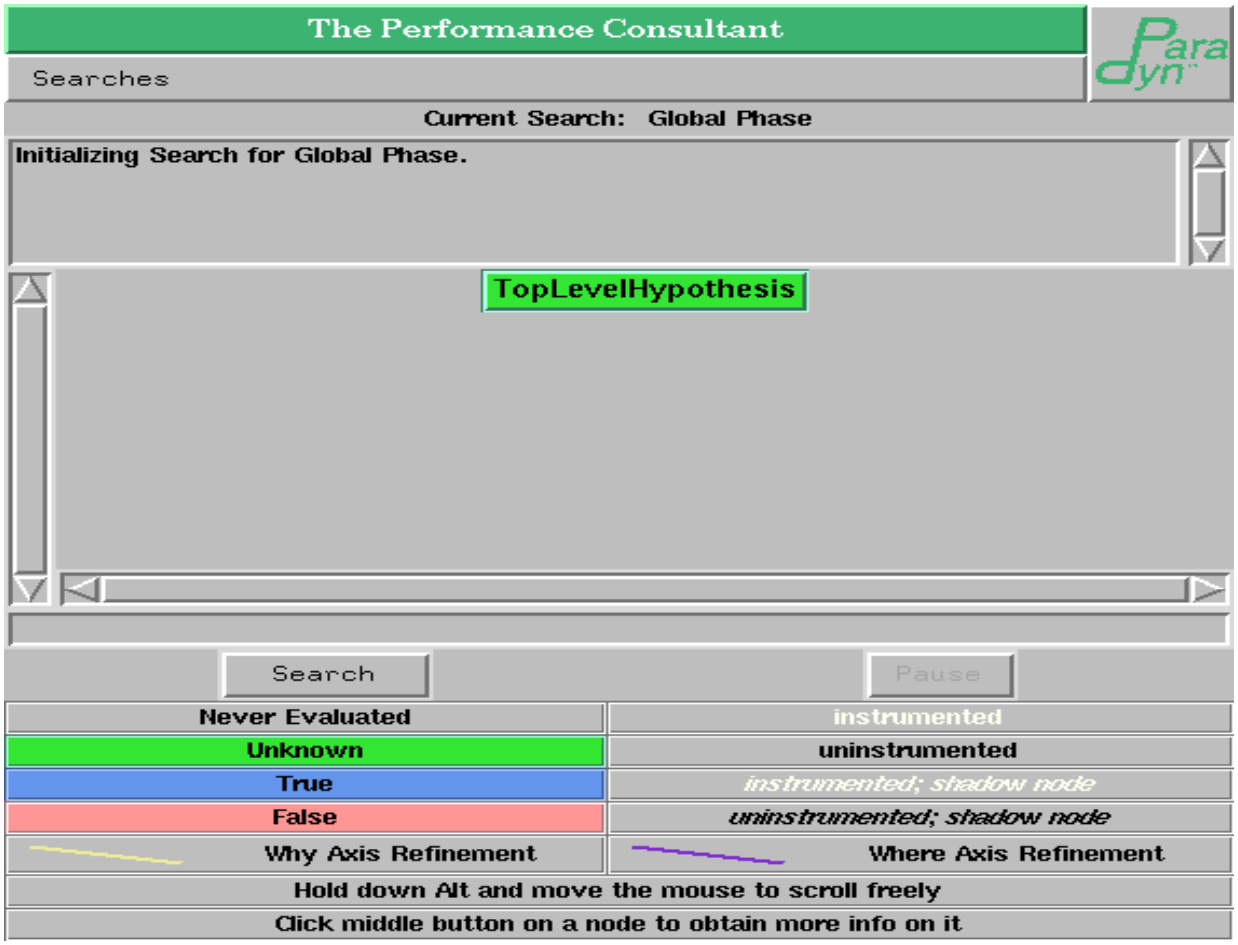

Figure 9: The Performance Consultant window

is displayed in the information line below the search history graph. For example, the information line below the search history graph in Figure 11 shows the hypothesis associated with the bottom-most node **4** in the graph.

Figure 10 shows the search history graph during the search for a bottleneck in **`potato`**. You can see that there have been refinements on both the Why and Where axis (these are indicated by yellow and purple edges in the search history graph). Also, there are nodes representing hypotheses that have tested true (blue nodes), nodes representing hypotheses that have tested false (pink nodes), and nodes representing hypotheses that have not yet been decided (green nodes).

Figure 11 shows the search history graph after the search has progressed further, and with only the nodes representing true hypotheses shown. The first hypothesis evaluated to true (the blue colored **TopLevelHypothesis** node at the top of the graph). The first refinement was on the Why axis and resulted in finding that there was a synchronization bottleneck in the application (the **ExcessiveSyncWaitingTime** node is true). Next, the synchronization bottleneck was isolated to a specific module in the application (`potato.c`) and to a specific type of synchronization object (**MsgTag**). The fact that these two nodes are siblings indicates that these refinements were done at the same time. These two nodes were then further refined in parallel. The result is that the bottleneck is isolated to a specific procedure (`dowork`) in module `potato.c`, and to a specific mes-

sage tag (message tag **4**). This means that the Performance Consultant found that there is excessive synchronization waiting time associated with message tag **4** in procedure `dowork`. At this point, the Performance Consultant was unable to further refine the bottleneck. However, it will continue to evaluate true nodes in the graph.

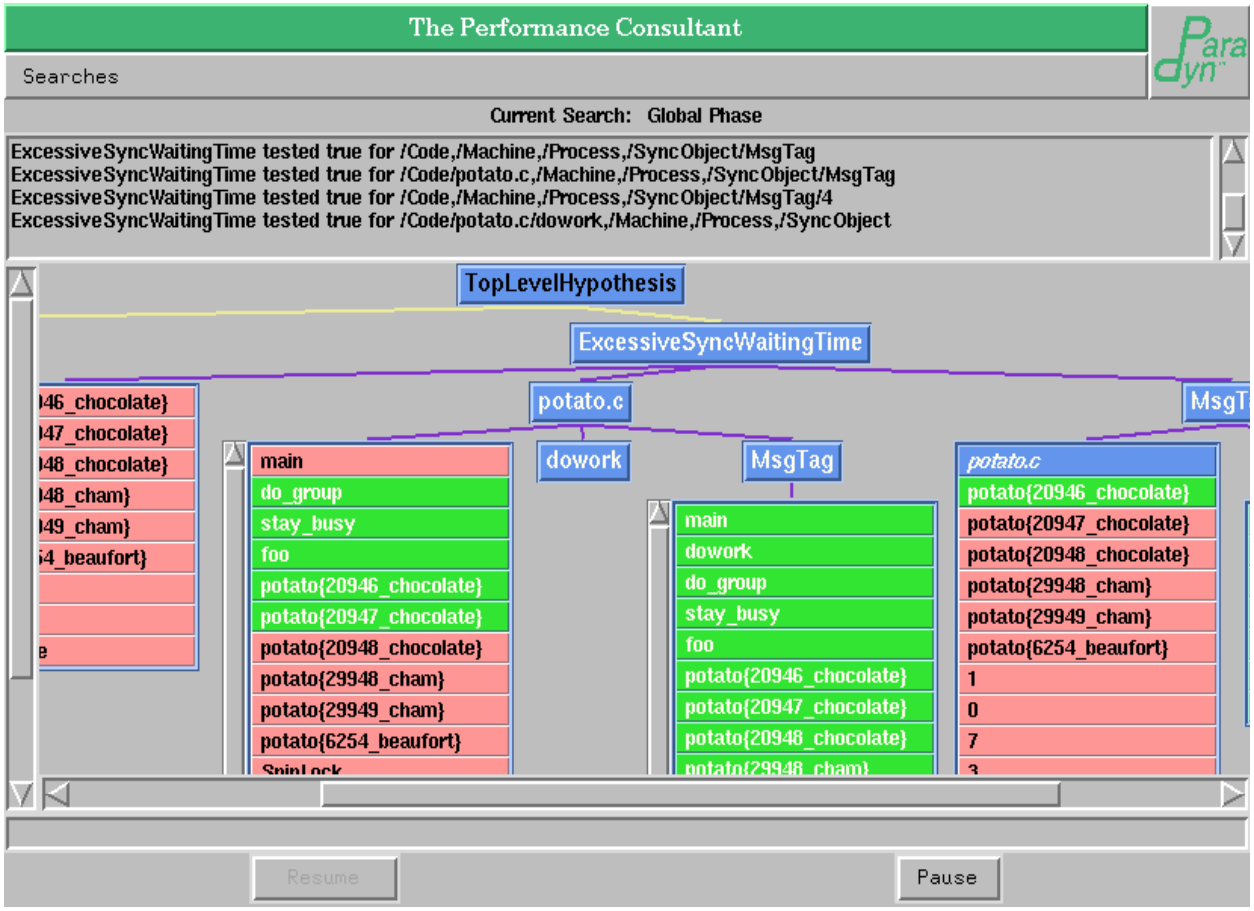

Figure 10: The Performace Consultant bottleneck search

5.3 Verifying the Performance Consultant's results

Typically, after running the Performance Consultant, you would like to see the performance data corresponding to the bottleneck in the application. To do this, you can start a visualization process to display performance data. In this example, after running the Performance Consultant, we started a barchart visualization by choosing BarChart from the list **Start A Visualization** menu (like Figure 6). The barchart is shown in Figure 12. It shows that most, if not all, of the *sync_wait* time for the whole program can be attributed to procedure `dowork` in module `potato.c` (the pink bars for each focus). It also shows that the *sync_wait* time is pretty evenly distributed across all processes, so it would be unlikely that the Performance Consultant would isolate the synchronization bottleneck to a proper subset of the processes.

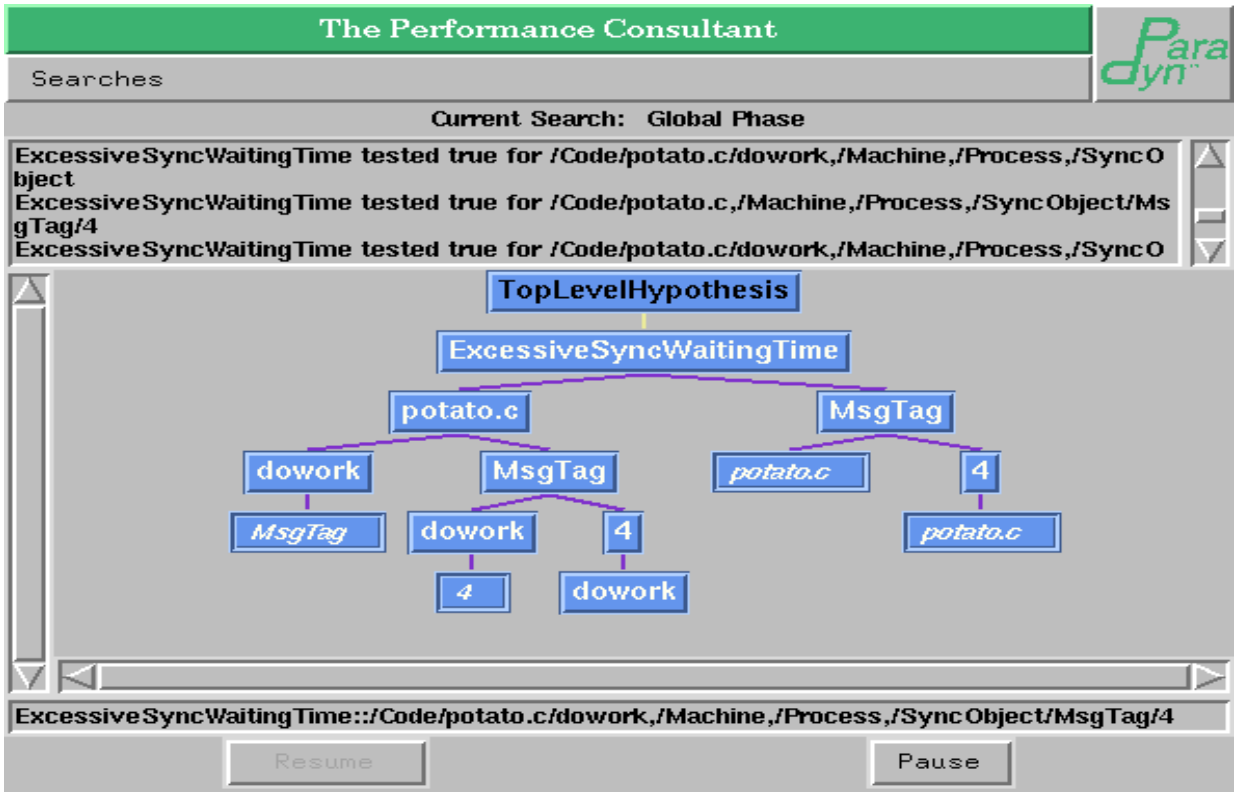

Figure 11: The Search History Graph showing only TRUE (blue) nodes

6 PHASES

In this section we briefly discuss Paradyn's notion of phases.

Phases are contiguous time-intervals within an application's execution. There are two types of phases: a *global phase* and zero or more *local phases*. The global phase includes the entire period of execution, from the start of the application program until the current time. This phase is the default for the Performance Consultant or any visualization. A local phase restricts performance information to a particular time interval. A local phase can be started at any time; the local phase ends when a new local phase is started. This means that, at any given time, you can select performance data from the global phase and from the current local phase.

One use of phases in Paradyn is to change the granularity of performance data collection after the application process has been running for some time. Because Paradyn uses fixed-size data structures to store performance data, the granularity of performance data becomes more coarse the longer the application runs. For some applications, the interesting behavior may not occur until several hours into its execution when the granularity of performance data is large. To obtain performance data at a finer granularity, you can start a new local phase. The data collection at the start of the new phase will be at the finest granularity supported by Paradyn. We use the **`potato`** program to provide an example of starting an phase part way into the application's execution.

To start a new phase, first create a phase table visualization by choosing **Phase Table** from the **Start A Visualization** menu. A phase table is shown in Figure 13. Next, click on the **Start A**

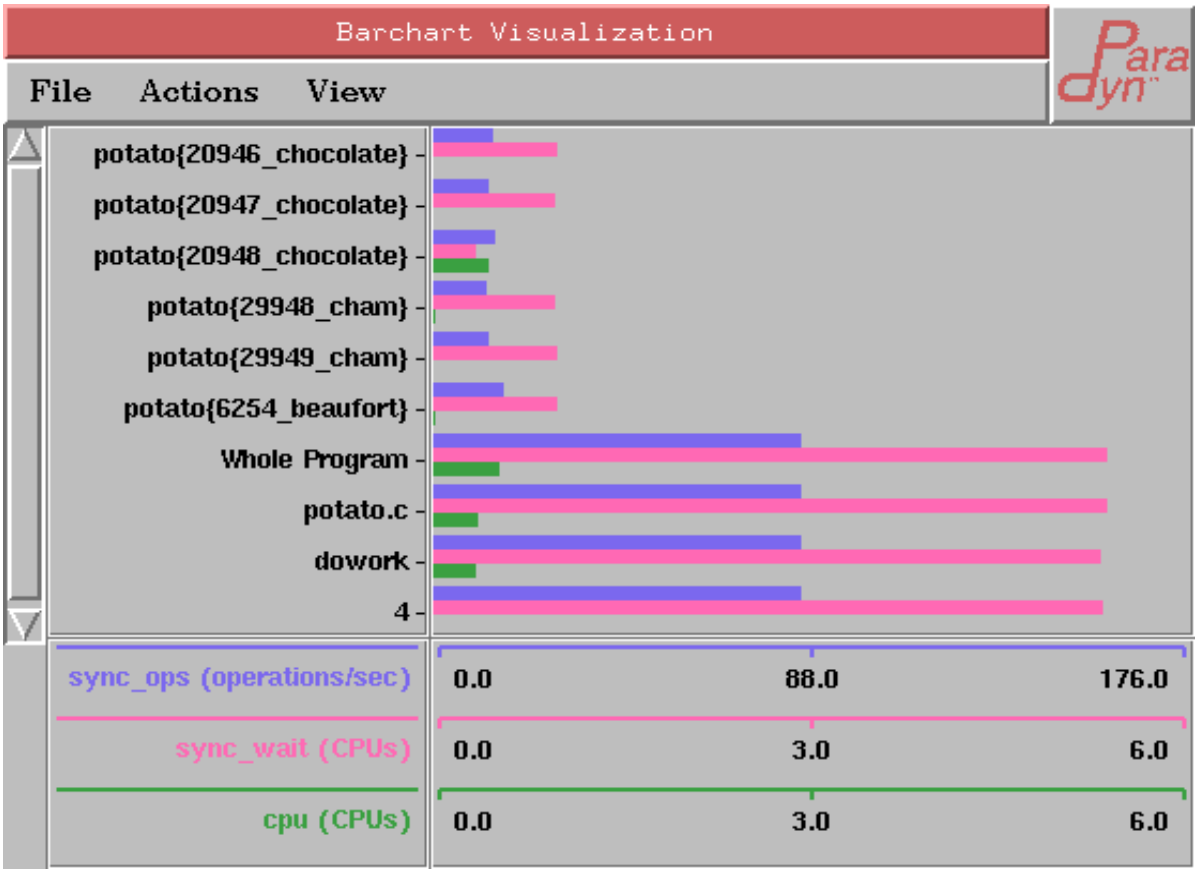

Figure 12: BarChart visi presenting the performance bottleneck data

Phase menu option from the phase table's menu bar. This will cause the phase table to display an end time for the previous phase (**phase_0** in the example), and a phase name and phase start time for the newly created current phase (**phase_1** and **11m 54s** in the example).

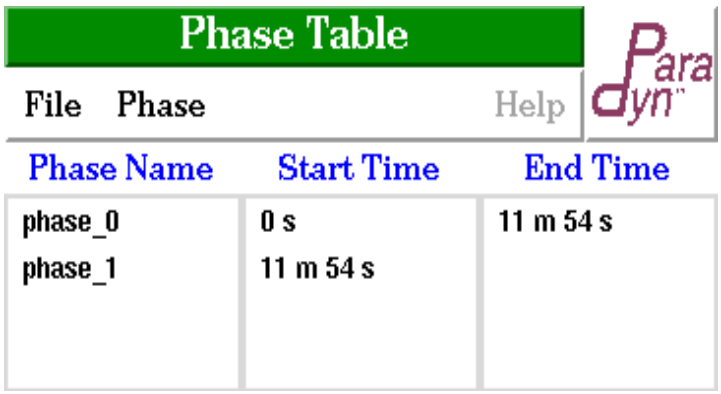

Figure 13: PhaseTable visi presenting phase durations

Once a new phase is started, you can create visualizations to display data from it by clicking on the **Current Phase** button in the lower right corner of the **Start A Visualization** window. Figure 14 and Figure 15 are time-histograms for the global and current phases respectively..

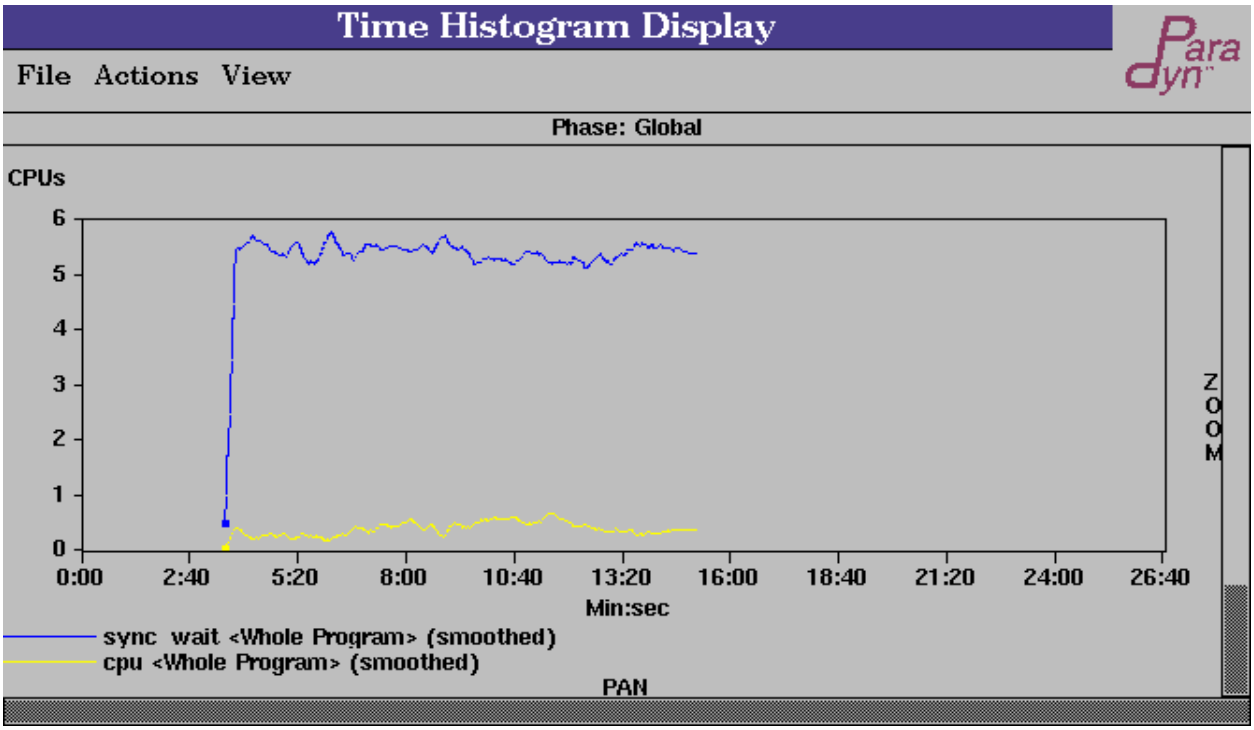

Figure 14: Histogram for global phase

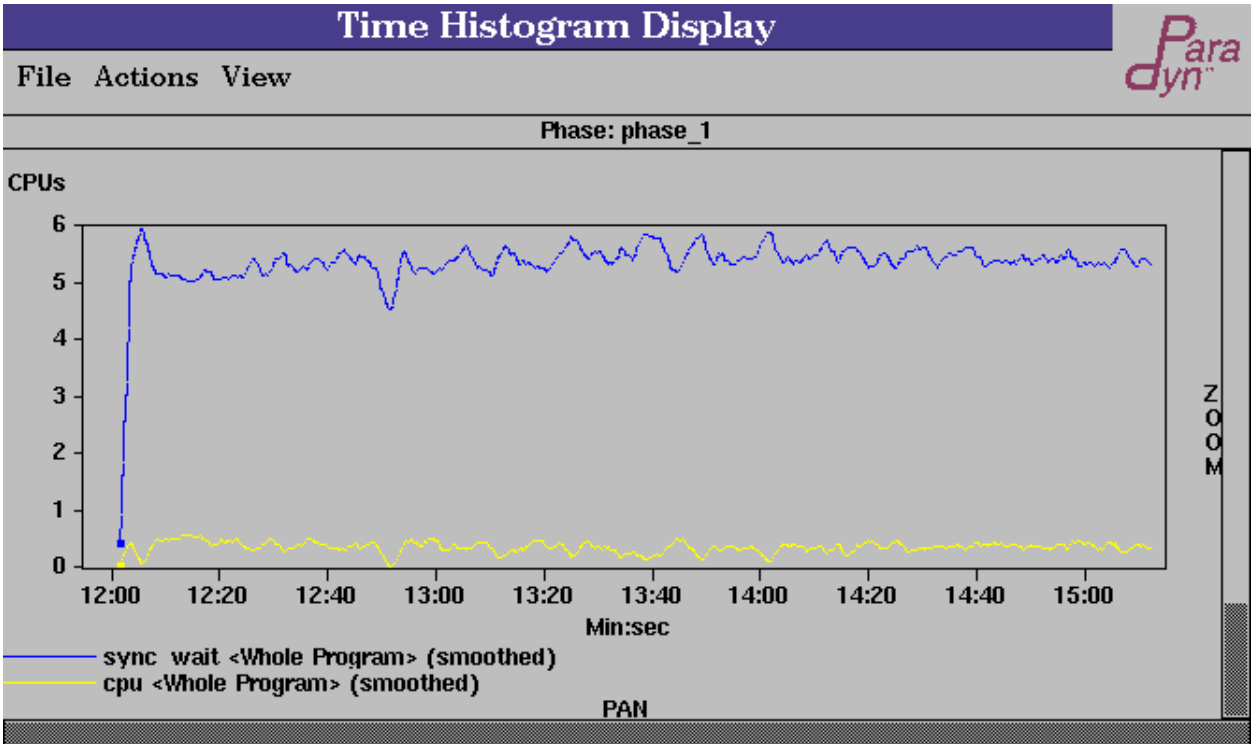

Figure 15: Histogram of current phase

Note that the current phase histogram starts at phase_1's start time (11:54) and displays data at a finer granularity than the same performance data displayed by the global phase histogram

■