Paradyn Parallel Performance Tools

Tutorial

Release 3.3
January 2002

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 start Paradyn, run an application, 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***.

This tutorial is provided in two parts. The first part covers the basic use of Paradyn, its visualizers and Performance Consultant using a simple sequential C application (**bubba**). This is followed by an additional tutorials for MPI (**decomp**, also provided in appropriate binary distributions), which may not be available on all systems or relevant to all Paradyn users. While there is some redundancy between the MPI tutorial and the basic tutorial, it considers MPI--specific functionality of Paradyn and additional examples of the use of Paradyn visualizers and Performance Consultant with message-passing programs.

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

2 COMMON TUTORIAL - BUBBA_SEQ

This first tutorial section covers the basic use of Paradyn with a simple sequential application (**bubba**) provided as part of the Paradyn binary distribution for every platform Paradyn supports. This common tutorial section is an introduction to Paradyn and its capabilities, which will be elaborated in the following section with additional functionality for MPI applications.

2.1 Running an application

Paradyn can start an application on the local or remote machine. The standard output and error messages of the application are displayed in a separate terminal window. The information displayed can be saved to a file.

2.1.1 Start Paradyn and define the application process

Paradyn can be started by entering the following command at a command prompt:[2]

```
% 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 load 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 application 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. On Windows NT, the command prompt is accessible via the "Command Prompt" item in the Start menu. Alternatively, the command may be issued from the "Run..." item in the Start menu. In both cases, the PATH environment variable must include the Folder in which the Paradyn executable (**paradyn.exe**) resides in order for Paradyn to run.

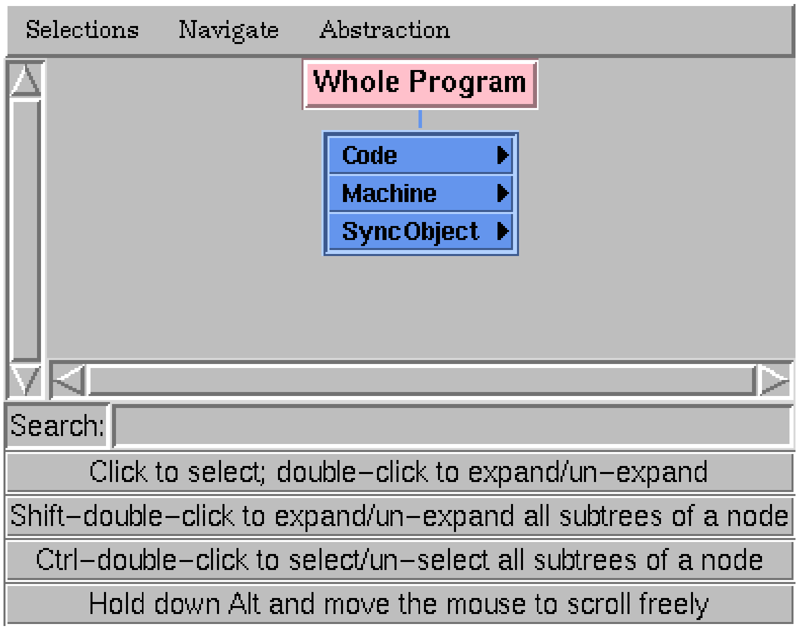

Figure 2: Paradyn base Where Axis.

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 different working directory for the application process. In this example, Paradyn will change to `/p/paradyn/demo/Paradyn/bubba_seq/@PLATFORM` before starting **bubba**.[3] Note that the path specified in this field is interpreted on the host specified in the **Host** field, which is not necessarily the host on which Paradyn is running.
4. **Command:** This entry takes the command that will start the application program. In this example we have entered "`bubba ../dat/example5`", which specifies the executable file (**bubba**) with one command line argument, `../dat/example5`, the input file.[4]
5. **Daemon:** This option allows you to specify which version of the Paradyn daemon to run. Since this is a sequential application, the `defd` daemon is selected. If the application is to be run under Windows NT, the winntd daemon should be selected.

   Once the fields of the **Define A Process** window have been filled in, click on the **Accept** but-

3. To simplify this tutorial, the macro `@PLATFORM` is used as shorthand for the environment variable `PLATFORM` specifying the processor-vendor-OS tuple for this host/executable. Paradyn's input parser currently doesn't make the appropriate environment variable substitutions itself, therefore, you must manually substitute the appropriate information. (Alternatively, filesystems such as AFS may permit definition of a symbolic link called `@PLATFORM` to achieve this illusion.)

4. On Windows NT, the bubba executable is called "**bubba.exe**." Also, note that if you choose to use backslashes instead of forward slashes, they must be escaped in the **Command** field on Windows NT. For example, to run the bubba executable located one folder up from the folder specified in the **Directory** field, the command would be "..\\bubba.exe ..\\..\\dat\\example5".

ton, and Paradyn will start your application process. This step can take anywhere from several seconds to several minutes as Paradyn examines and starts your application, depending on its size, the load on the machine, network connection speed, etc.

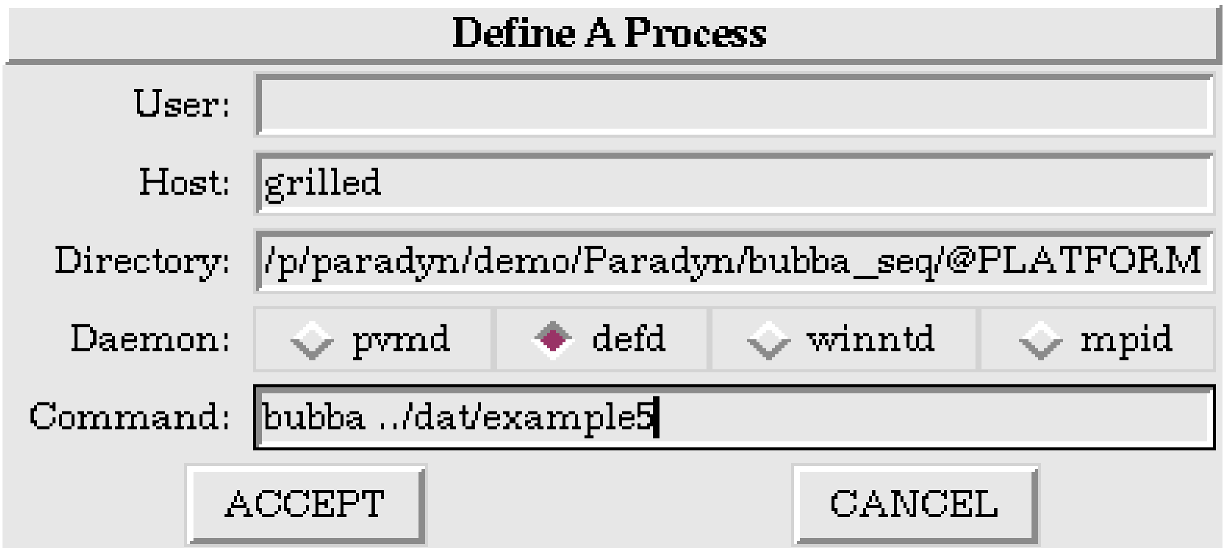

Figure 3: The Define A Process window specifying `bubba` application process

2.1.2 Starting an application process manually

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 (**`bubba`** or **bubba.exe**), the name of the machine (grilled), the name of the user (self), and the name of the daemon (defd)
2. **Processes:** A list of the process IDs of all the processes in the application. In this example, there is one pid (18904) corresponding to the process started on host grilled.
3. **Application status:** The current status of the application program (either RUNNING, PAUSED, or EXITED).
4. **grilled:** 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 examine the program executable(s), 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 machine names in the **Machine** hierarchy. Figure 5 shows the Where Axis with these new resources added. The Machine hierarchy contains the machine "`grilled.cs.wisc.edu`" under which is the process "`bubba{18904}`", and

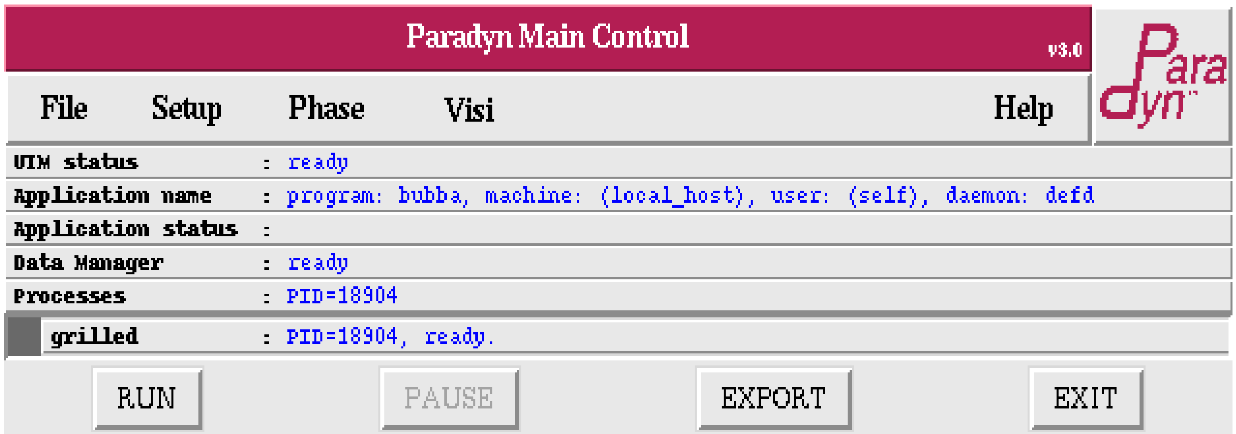

Figure 4: Paradyn Main Control window with `bubba` loaded and ready to run

the Code hierarchy contains several new entries corresponding to a source code modules. Double-

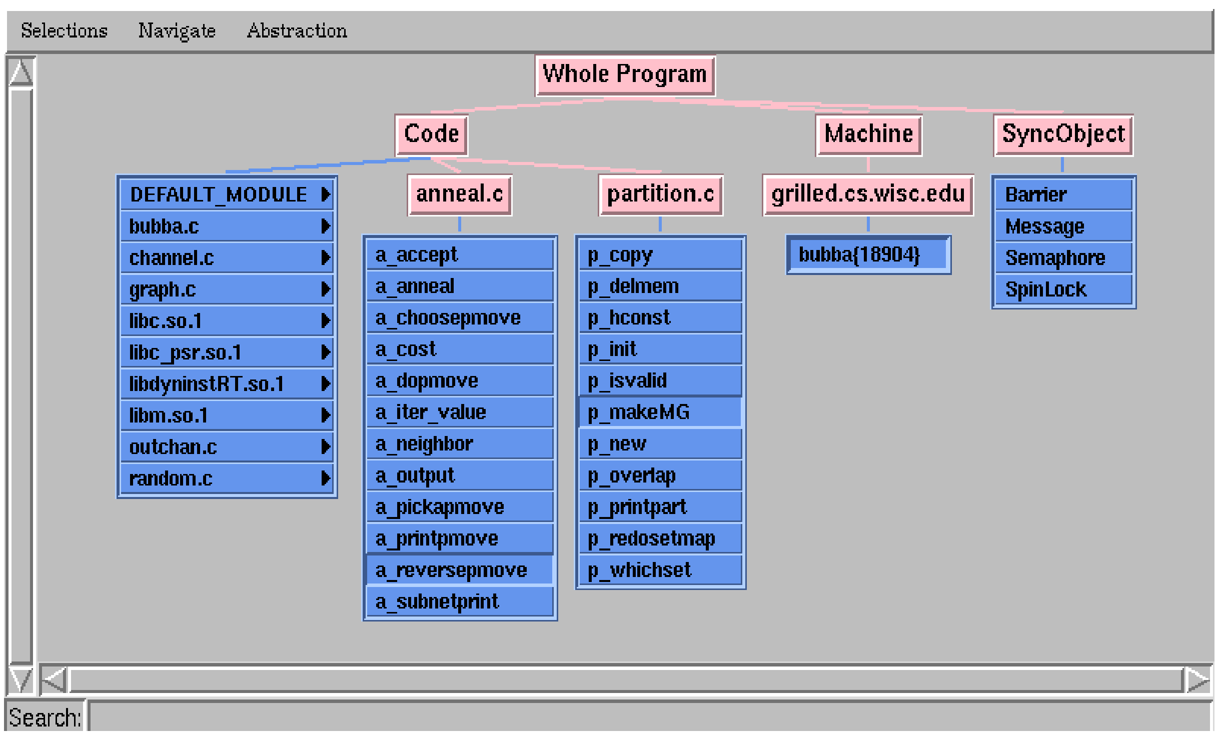

Figure 5: Where Axis after the `bubba` application process is loaded

clicking on nodes with a triangle on their righthand edge expands them to show the nodes they contain; double-clicking on a head-node folds it into its parent node. Single-clicking on nodes (including head-nodes) will be used later to select (sets of) resources for metric foci. Locating particular nodes can be achieved by typing a search string in the labeled field and then enter.

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 `bubba`, 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).

2.1.3 Starting an application process automatically

Paradyn can also start an application using a PCL specification file. Below shows a command to start the **bubba** application using a PCL file called `bubba.pcl`:

```
% paradyn -f bubba.pcl
```

The contents of the PCL file, `bubba.pcl`, are shown as follows:

```
// Paradyn configuration file for bubba (generic)

process bubba
{
        //host "localhost";
        dir "/p/paradyn/demo/Paradyn/bubba_seq/@PLATFORM";
        command "bubba ../dat/example5";
        daemon defd;
}
```

In the PCL file, the (optional) host, the directory, the command, and the daemon are specified as when starting an application manually, and on start-up Paradyn automatically loads and prepares this process ready for execution and analysis.

2.2 Viewing performance data

Before you run the application process, you may want to start a visualizer (or *visi*)[5]. For this application, we will start a time-histogram visualization to view CPU utilization for the application. In this section, we describe how to start a visualizer, and how to choose the set of metrics and parts of the program that a visualizer will display.

2.2.1 Starting a visualizer

To start a visualizer, 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 2.4 will discuss phases). Other visualizations allow metric data to be presented in tabular and barchart form, etc., though all visualizers may not be available on all platforms.

Once the visualization selection has been made, click on the **Start** button and Paradyn will display a metrics dialog. This dialog, shown in Figure 7, allows you to select the set of metrics to be displayed by the visualization.[6] In this example, we have selected *cpu* (CPU time) and *cpu_inclusive* (CPU inclusive). The *cpu* metric if applied to a function will exclude time spent in any function it calls, whereas the *cpu_inclusive* metric includes time spent in the selected function and the functions that are called by it.

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

5. Visualizers 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.
6. The metrics dialog shows all metrics defined for the current platform(s).

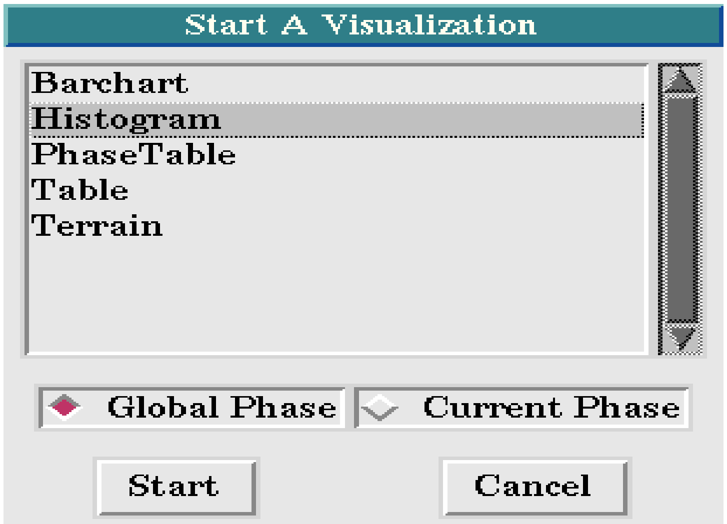

Figure 6: Selecting a Histogram visualization

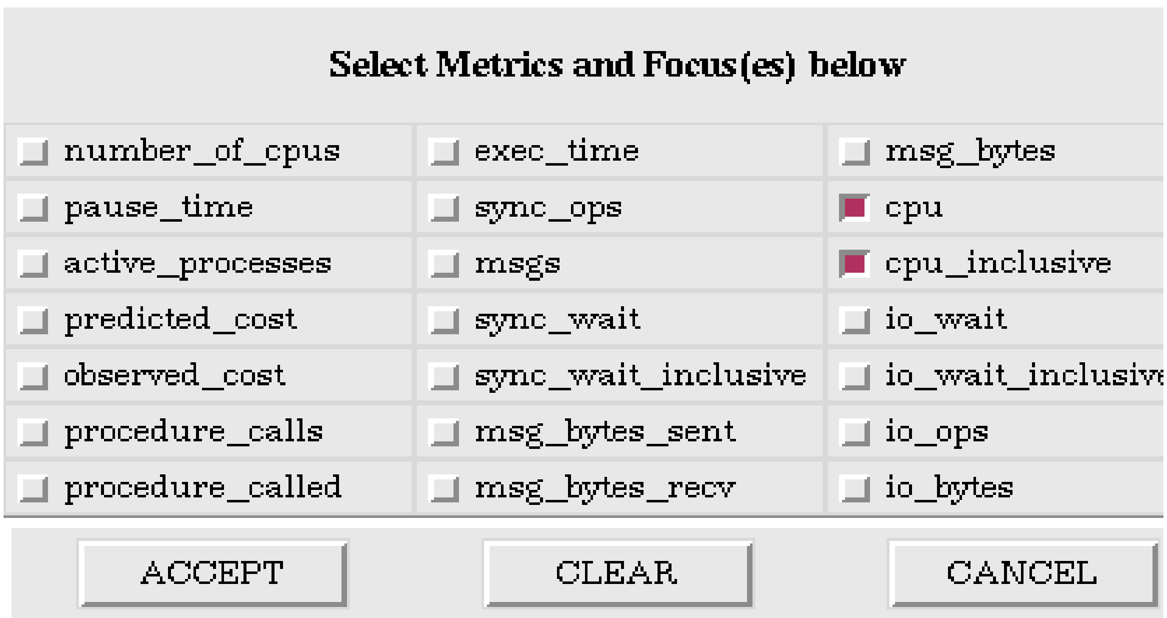

Figure 7: Metrics menu with "cpu" and "cpu_inclusive" selected

data can be collected. For example, if you select the node **bubba{18904}** from the Machine hierarchy, you limit data collection to the process **bubba{18904}**. If you select **p_makeMG** and **a_reversepmove** from the Code hierarchy, you limit data collection to function `p_makeMG` and `a_reversepmove`. 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 module selections, then you limit data collection to activity in the functions `p_makeMG` and `a_reversepmove` in the process `bubba{18904}`. This selection corresponds to two foci: the first focus is when the process 18904 is running in function `p_makeMG`; the second focus is when process 18904 is running in function `a_reversepmove`.

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 **CPU_INCLUSIVE**, and the resource nodes **bubba{18904}** and **p_makeMG** were selected, then Paradyn would try to enable four metric-focus pairs:

- *CPU* time for process 18904 when it is running in function `p_makeMG`.
- *CPU* time for process 18904 when it is running in function `p_makeMG`.
- *cpu_inclusive* time for process 18904 when it is running in function `p_makeMG`.
- *cpu_inclusive* time for process 18904 when it is running in function `p_makeMG`.

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.[7] If there are any metric-focus pairs that could not be enabled, Paradyn will display a message listing those pairs, and redisplay 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.

The time-histogram shown in Figure 8 is the result of selecting the metrics “cpu” and “cpu_inclusive” from the metrics menu and **bubba{18904}** and **p_makeMG** and **a_reversepmove** 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***.

2.3 Performance Consultant diagnosis

The Performance Consultant is the part of Paradyn that performs an automated hierarchical 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 aspect of the application’s execution. This aspect is specified as a point in a three dimensional search space defined by a Why Axis, Where Axis, and When Axis.

2.3.1 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. (Phases are

7. Metric data isn’t sampled or displayed before the application starts running or while it is paused.

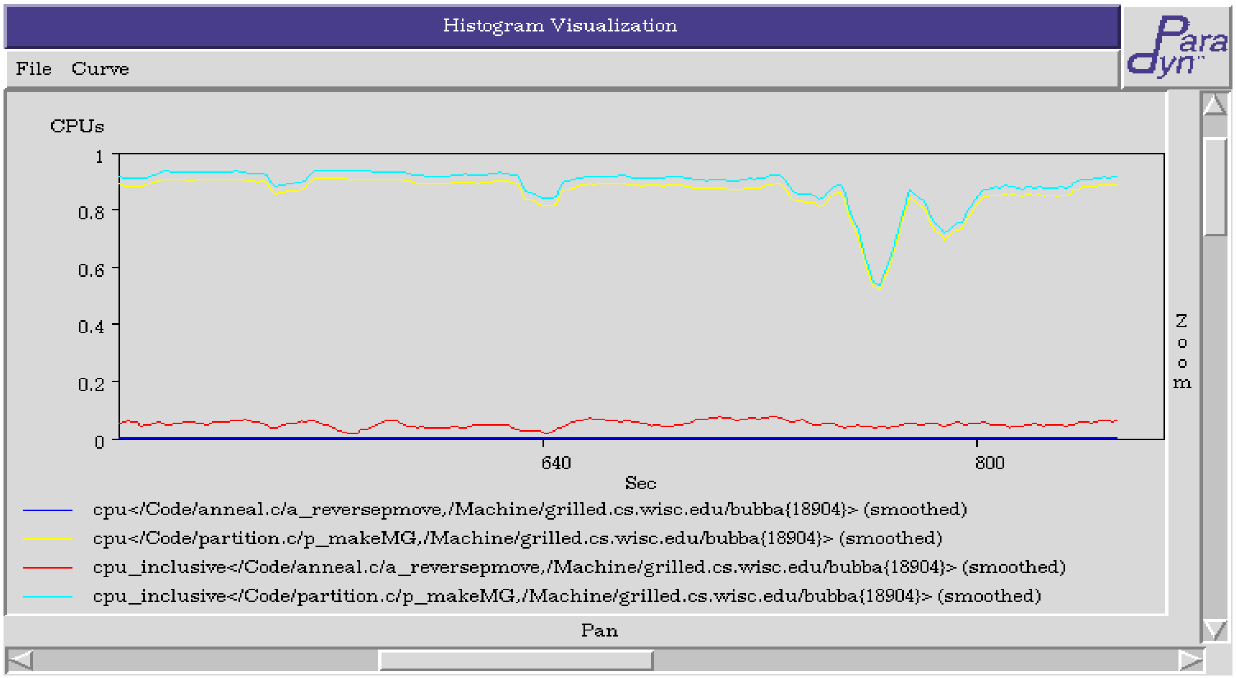

Figure 8: Histogram of global phase with “cpu” and “cpu_inclusive” for two foci

discussed later in Section 2.4.)

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.

2.3.2 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 watching the Search History Graph, we can see how the Performance Consultant itera-

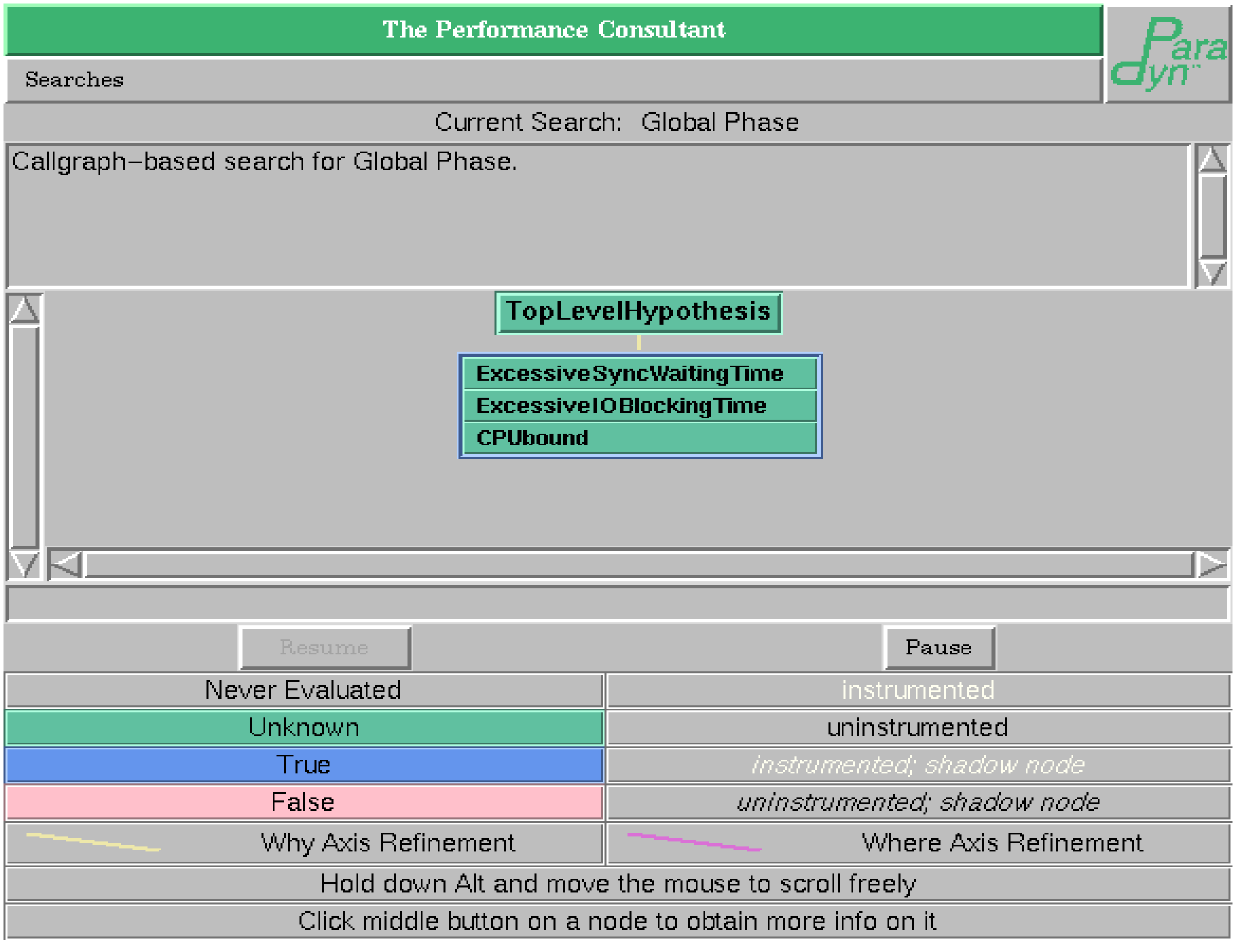

Figure 9: The Performance Consultant window

tively 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 right 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 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 nodes in the graph.

Figure 10 shows the search history graph during the search for a bottleneck in **`bubba`**. 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 cpu bottleneck in the application (the **CPUbound** node is true). Next, the synchronization bottleneck was isolated to the function main and machine grilled.cs.wisc.edu. 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 concurrently. The result after several such refinements is that the bottleneck is isolated to a specific procedure (`p_makeMG`). This means that the Performance Consultant found that there is CPU bottleneck in procedure `p_makeMG`. At this point, the Performance Consultant was unable to further refine the bottleneck. However, it will continue to evaluate true nodes in the graph.

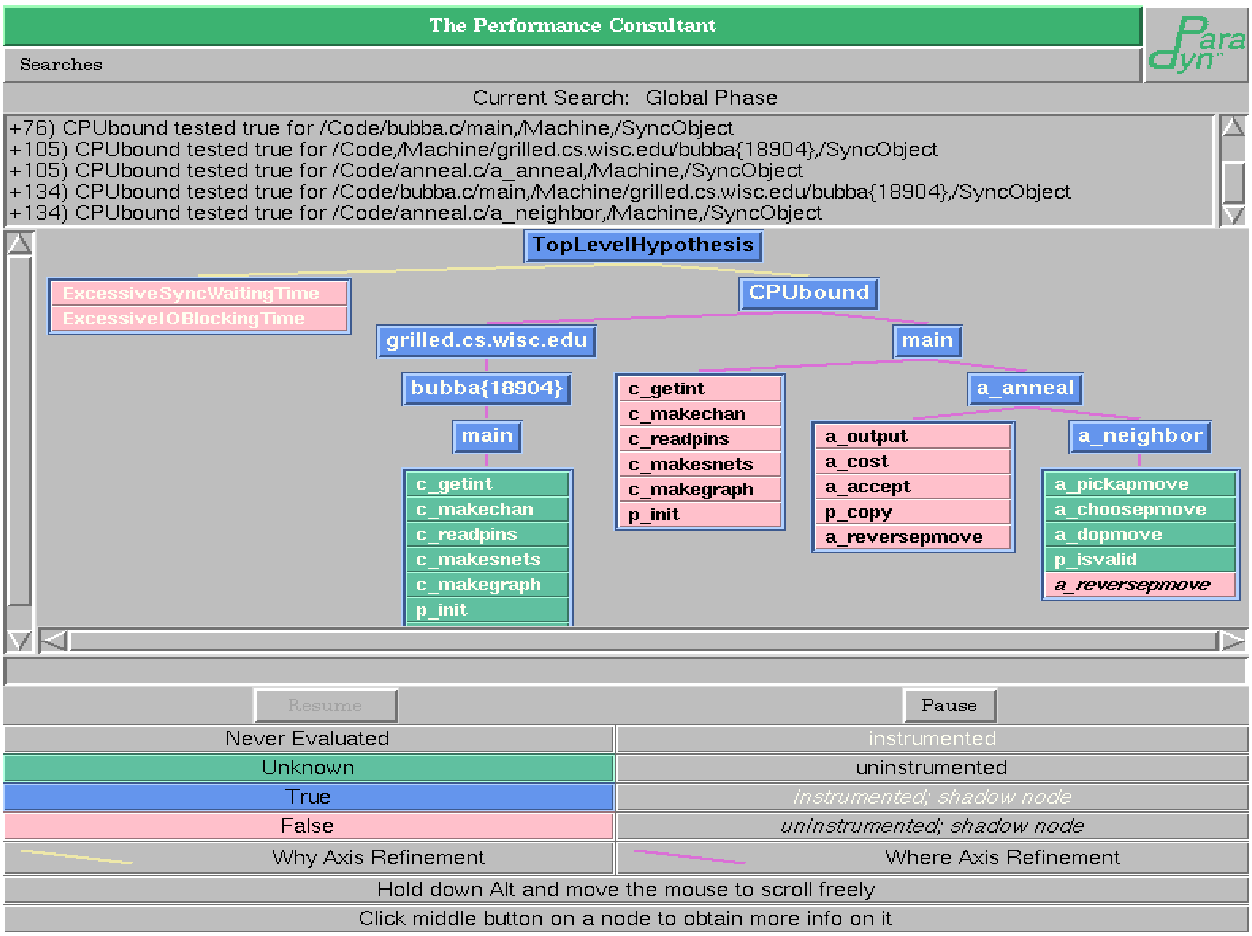

Figure 10: The Performance Consultant `bubba` exigency search

2.3.3 Investigating the Performance Consultant's diagnosis

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 almost all of the *cpu* time for **bubba** can be attributed to procedure `p_makeMG`.

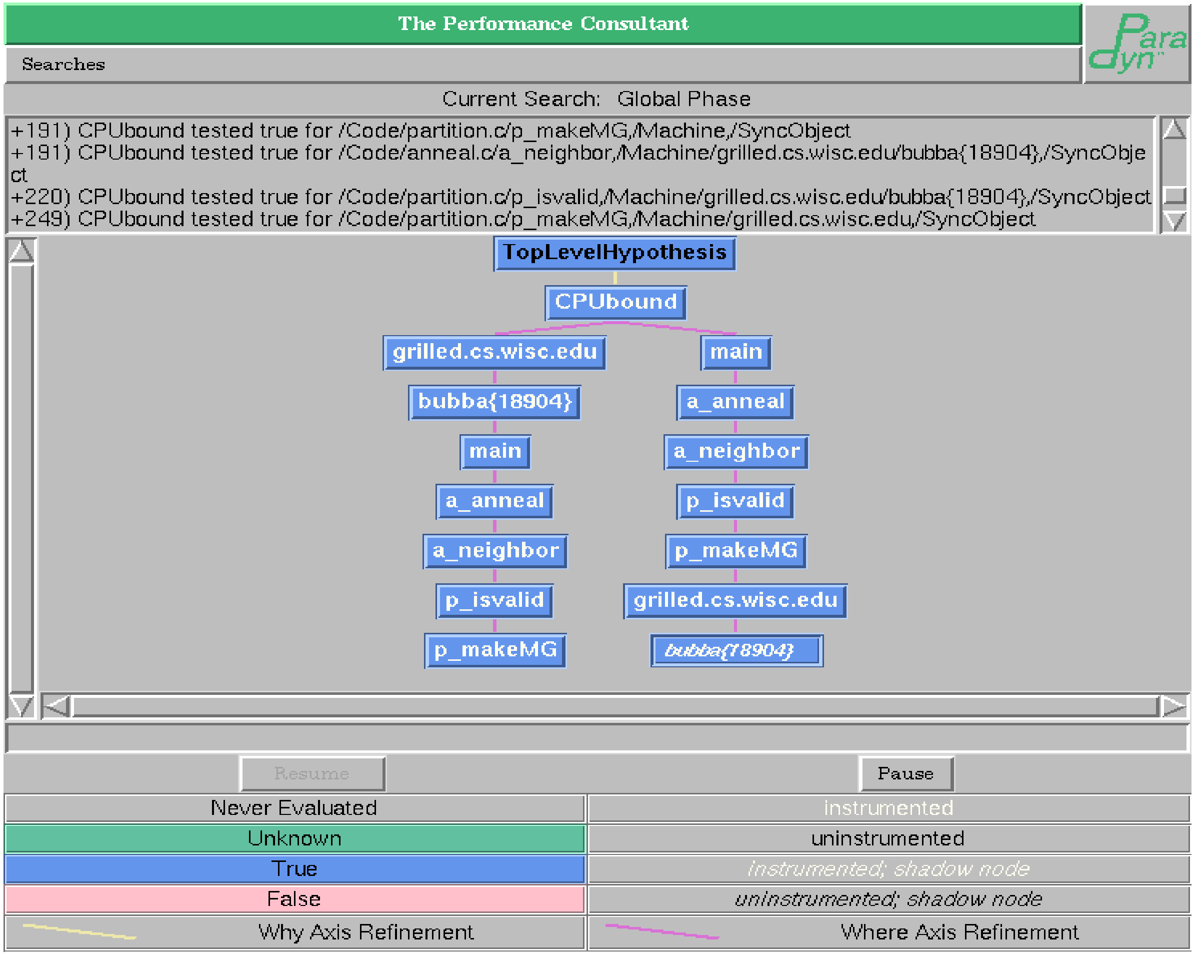

Figure 11: The Search History Graph showing only exigent `bubba` nodes

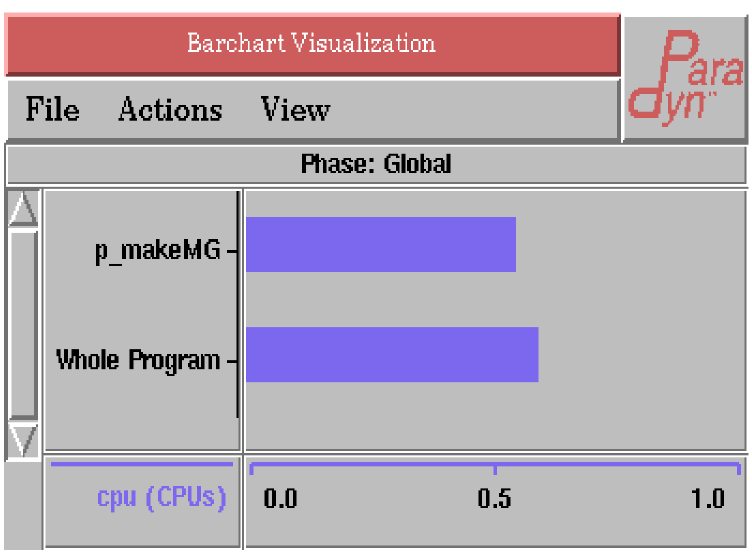

Figure 12: BarChart visi presenting selected `bubba` performance data

2.4 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.

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 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).

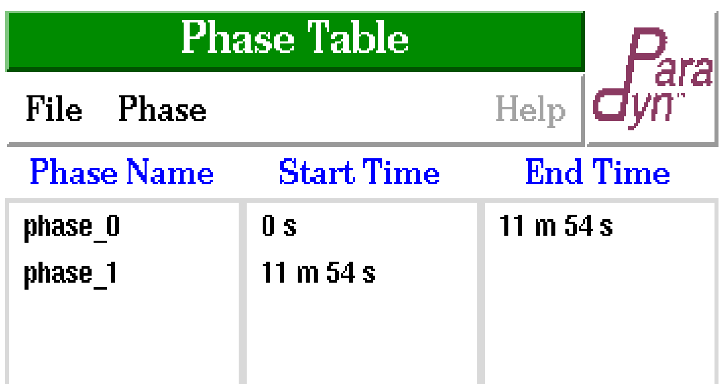

Figure 13: PhaseTable visi presenting phase durations.

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..

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.

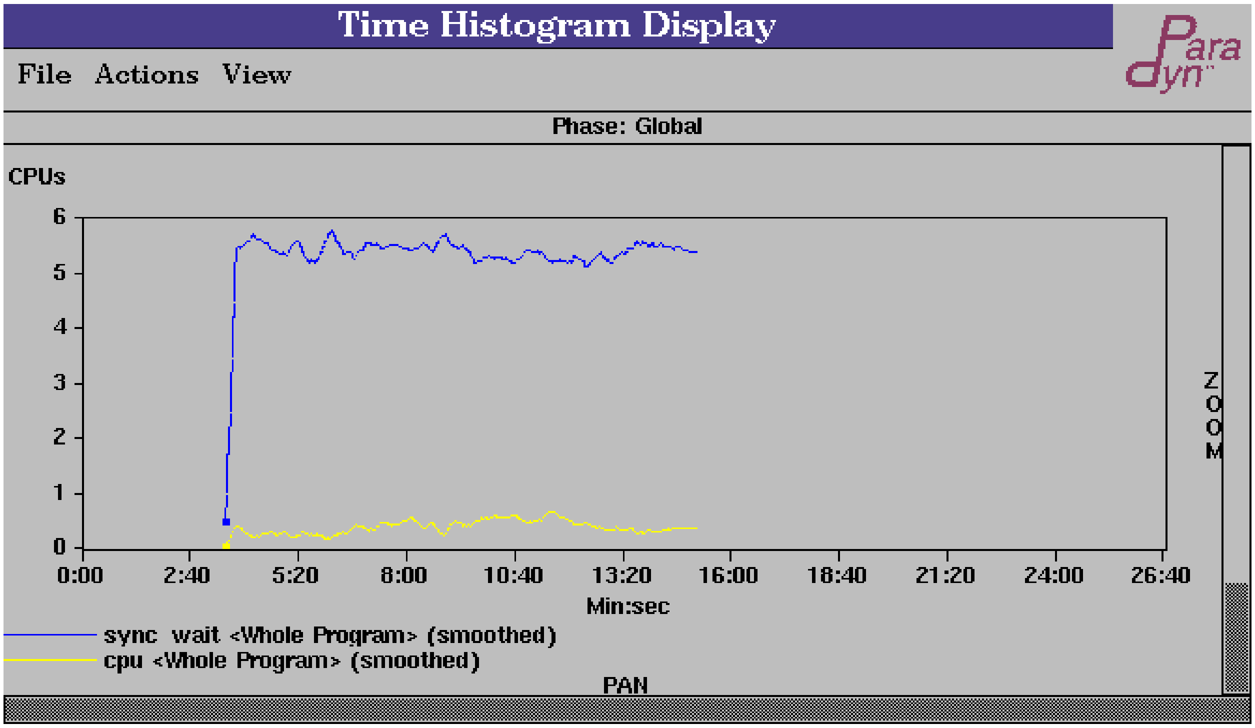

Figure 14: Histogram for global phase

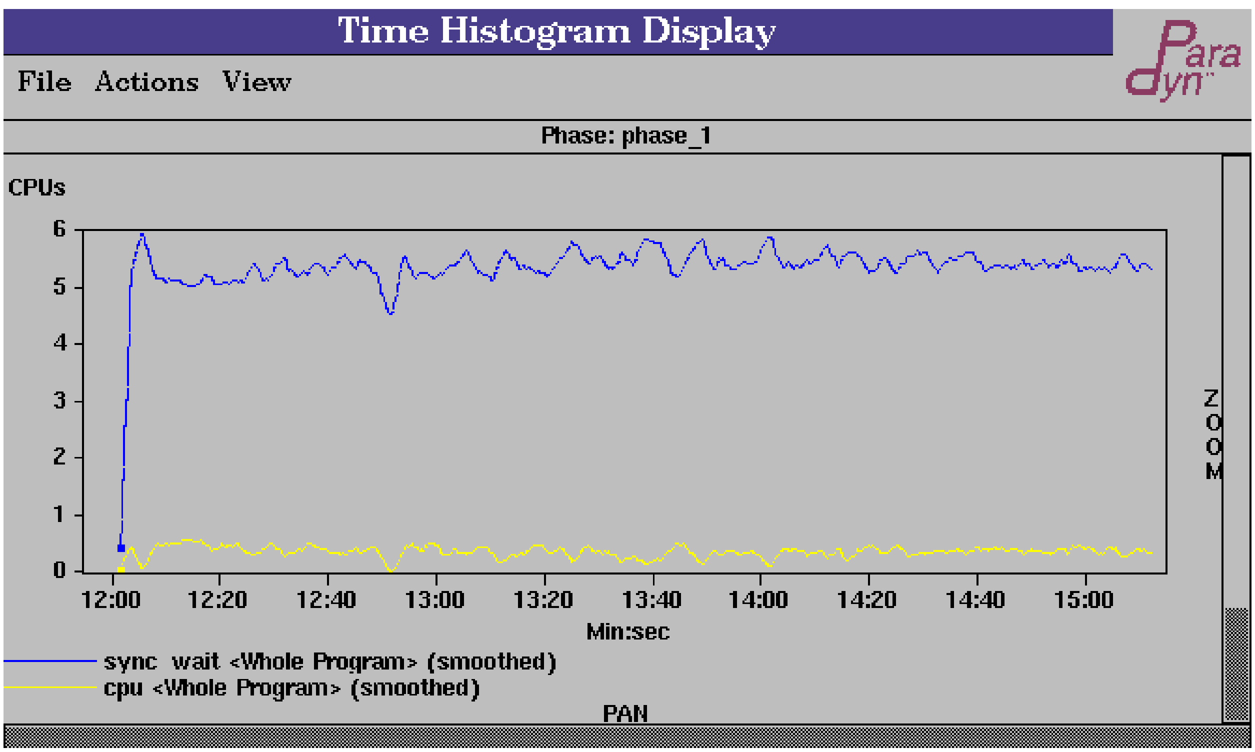

Figure 15: Histogram of current phase

3 MPI TUTORIAL - DECOMP_MPI

This tutorial section covers the use of Paradyn with a simple MPI application (**om3**) provided as part of the Paradyn binary distribution for platforms where MPI is supported. MPI is not yet supported by Paradyn on all platforms: see the ***Paradyn User Guide*** for details.

3.1 Running the MPI application

3.1.1 Start Paradyn and define the MPI application process

The first 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.

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 16. To describe the application 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 `/p/paradyn/applications/mpi/om3` before starting **`om3`**.
4. **Command:** This entry takes the unix command that will start the application program. The syntax for this command for launching MPI jobs will vary by platform. For MPICH, the entire command-line including the `mpirun` command and all of its appropriate arguments should be entered. For AIX, the POE job launcher `poe` can be entered or omitted. In this example we have entered "`mpirun -np 4 -machinefile hostfile om3_4node`", which specifies the executable file (**`om3_4node`**) with two command line arguments: the number of processes (`4`), and a file containing node names (hostfile).
5. **Daemon:** This option allows you to specify which version of the Paradyn daemon to run. Since this is an MPI application, the `mpid` 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.

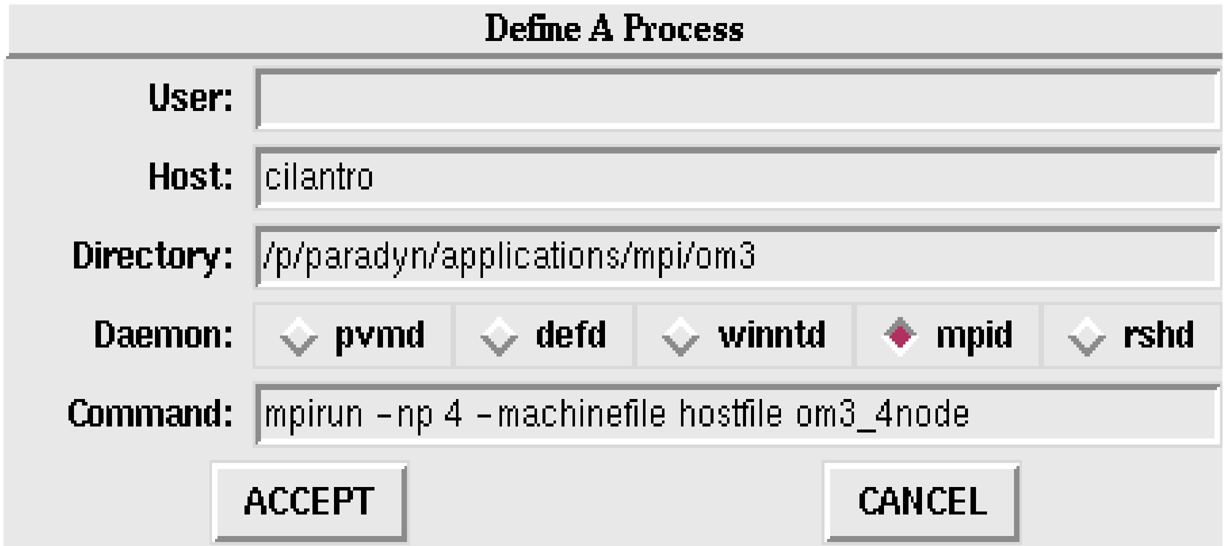

Figure 16: The Define A Process dialog for MPI om3

3.1.2 Start the MPI application process manually

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 17 (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 (in this case, **`mpirun`** is named), the name of the machine (c23), the name of the user (self), and the name of the daemon (rshd)
2. **Processes:** Typically Paradyn will indicate the process ids in this field. In the case of MPICH, this field is used to indicate that paradyn has identified the job as an MPICH job.
3. **Application status:** The current status of the application program (either READY, RUNNING, PAUSED, or EXITED).
4. **Hosts:** Status lines for each host. Once the application starts running these will display the status of each host (running, paused, or exited). In Figure 17 only the hostname c23is shown as we have not yet started the application.

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 **Machine** hierarchy. Figure 18 shows the new Where Axis with these new resources added. The Process hierarchy contains four new processes (one for each MPI process).

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 **`om3`**, or alternatively first define

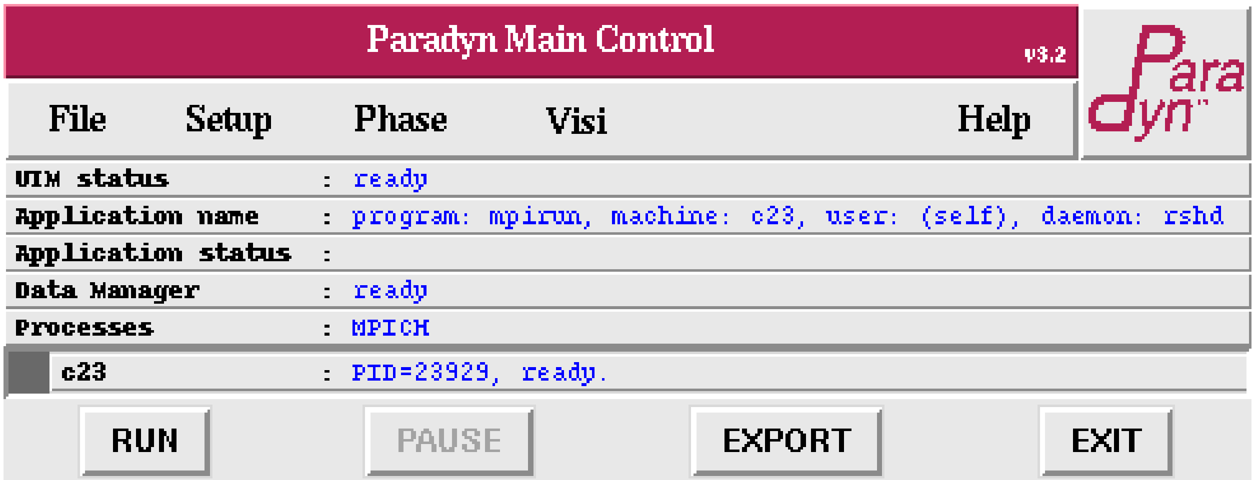

Figure 17: Paradyn Main Control window after the MPI application process is started

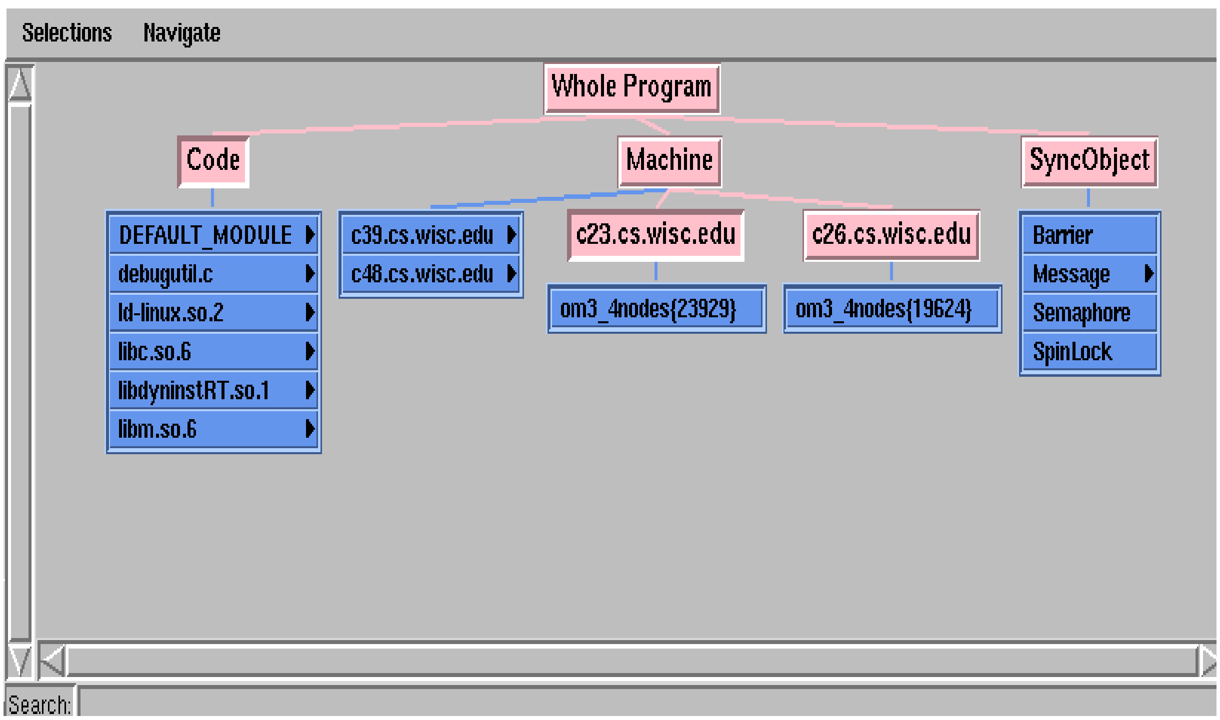

Figure 18: Where Axis after the om3 MPI application process is started

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).

3.2 Viewing performance data

Before you run the application process, you may want to start a visualizer[1]. 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 visualizer, and how to choose the set of metrics and parts of the program that a visualizer will display.

3.2.1 Starting a visualizer

To start a visualizer, 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 visualizer selected for the Global Phase (Section 2.4 discusses phases).

Once the visualization selection has been made, click on the **Accept** button and Paradyn will display a metrics menu appropriate for this MPI application. This menu, shown in Figure 19, allows you to select the set of metrics to be displayed by the visualization. In this example, we have selected *sync_wait_inclusive* (inclusive synchronization blocking time) and *cpu_inclusive* (inclusive CPU time).

To choose the parts of the program for which the metric will be collected, 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, selecting the nodes **om3_4nodes{23929}** and **om3_4nodes{19624}** from the Process hierarchy, limits data collection to these two processes (23929 on c23 and 19624 on c26). Selecting a module from the Code hierarchy limits data collection to that module. Figure 18 shows the Where Axis.

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 module selections, then you limit data collection to activity in a particular module only in processes 23929 and 19624. This selection corresponds to two foci: the first focus is when process 23929 is running in the module you selected; the second focus is when process 19624 is running in that module.

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_inclusive** and **sync_wait_inclusive**, and the resource nodes **om3_4nodes{23929}**, **om3_4nodes{19624}**, and **libm.so.6** were selected, then Paradyn would try to enable four metric-focus pairs:

- *CPU_inclusive* time for process 23929 when it is running in module `libm.so.6`.
- *CPU_inclusive* time for process 19624 when it is running in module `libm.so.6`.
- *sync_wait_inclusive* time for process 23929 when it is running in module `libm.so.6`.
- *sync_wait_inclusive* time for process 19624 when it is running in module `libm.so.6`.

1. Visualizers 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.

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.

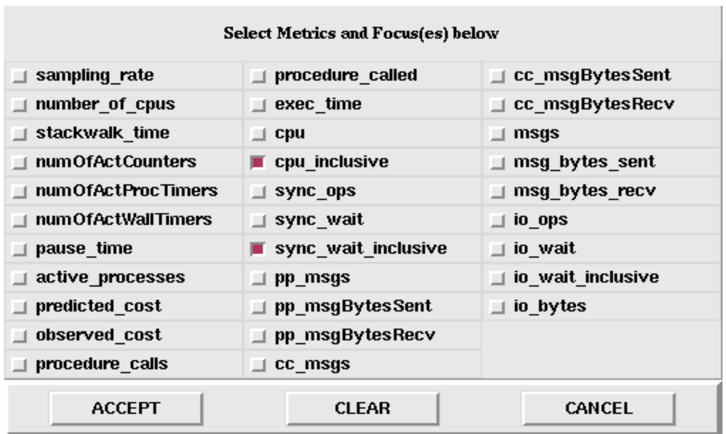

Figure 19: MPI metrics menu with “sync_wait_inclusive” and “cpu_inclusive” selected

The time-histogram shown in Figure 20 is the result of selecting the metrics “sync_wait_inclusive” and “cpu_inclusive” from the metrics menu with machine c23 selected in the WhereAxis.

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***.

3.3 Performance Consultant diagnosis

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.

3.3.1 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 initial 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.

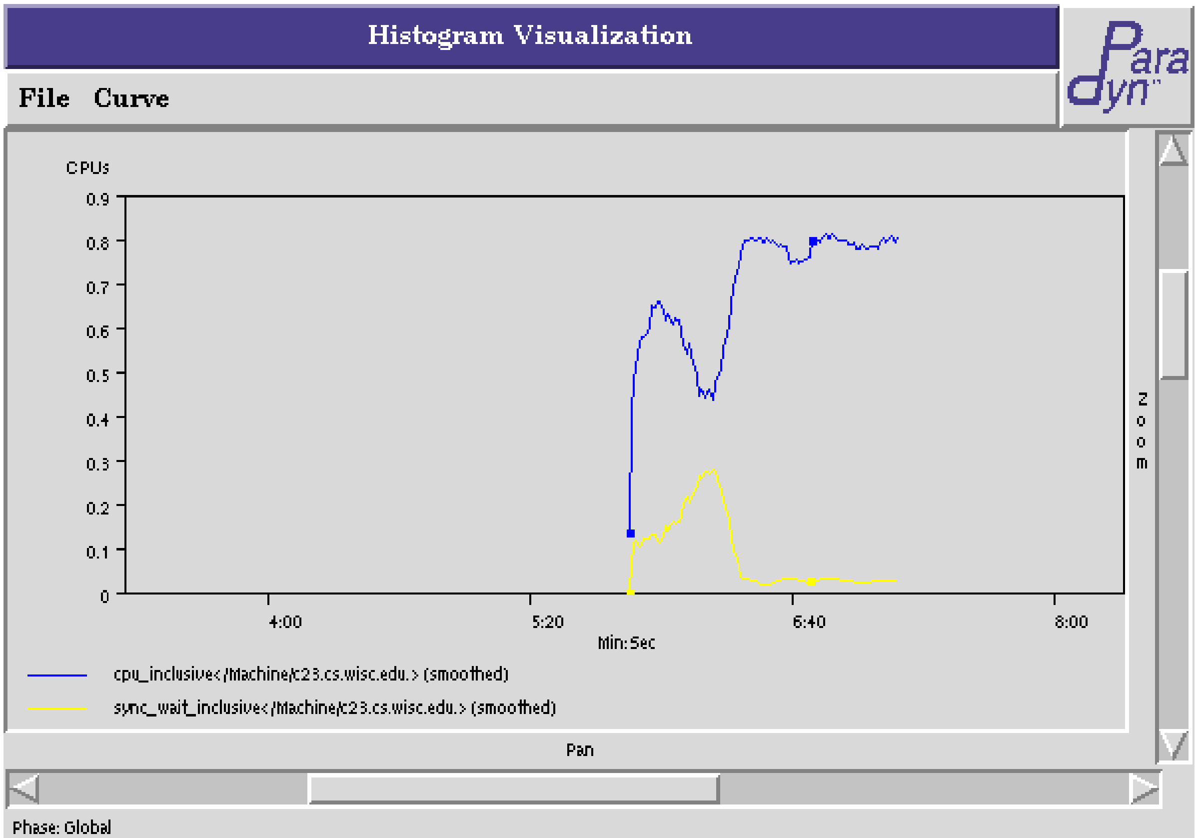

Figure 20: Histogram of global phase for "sync_wait_inclusive" and "cpu_inclusive"

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.
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 nodes and edges in the search history graph, and how to navigate around the window.

3.3.2 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 21 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 right 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 is displayed in the information line below the search history graph. For example, the information line below the search history graph in Figure 22 shows the hypothesis associated with the node representing the *sync_wait_inclusive* time for the whole program.

Figure 21 shows the search history graph during the search for a bottleneck in **`om3`**. 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). Note that this application is CPU-bound.

Figure 22 shows the search history graph after the search has progressed further. 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 the application is CPU bound (the **CPUbound** node is true). Next, the synchronization bottleneck was isolated to a specific function in the application (`main`). and to specific machines (**c23, c26, c39, c48**). The fact that thesenodes are siblings indicates that these refinements were done at the same time. These nodes were then further refined in parallel.

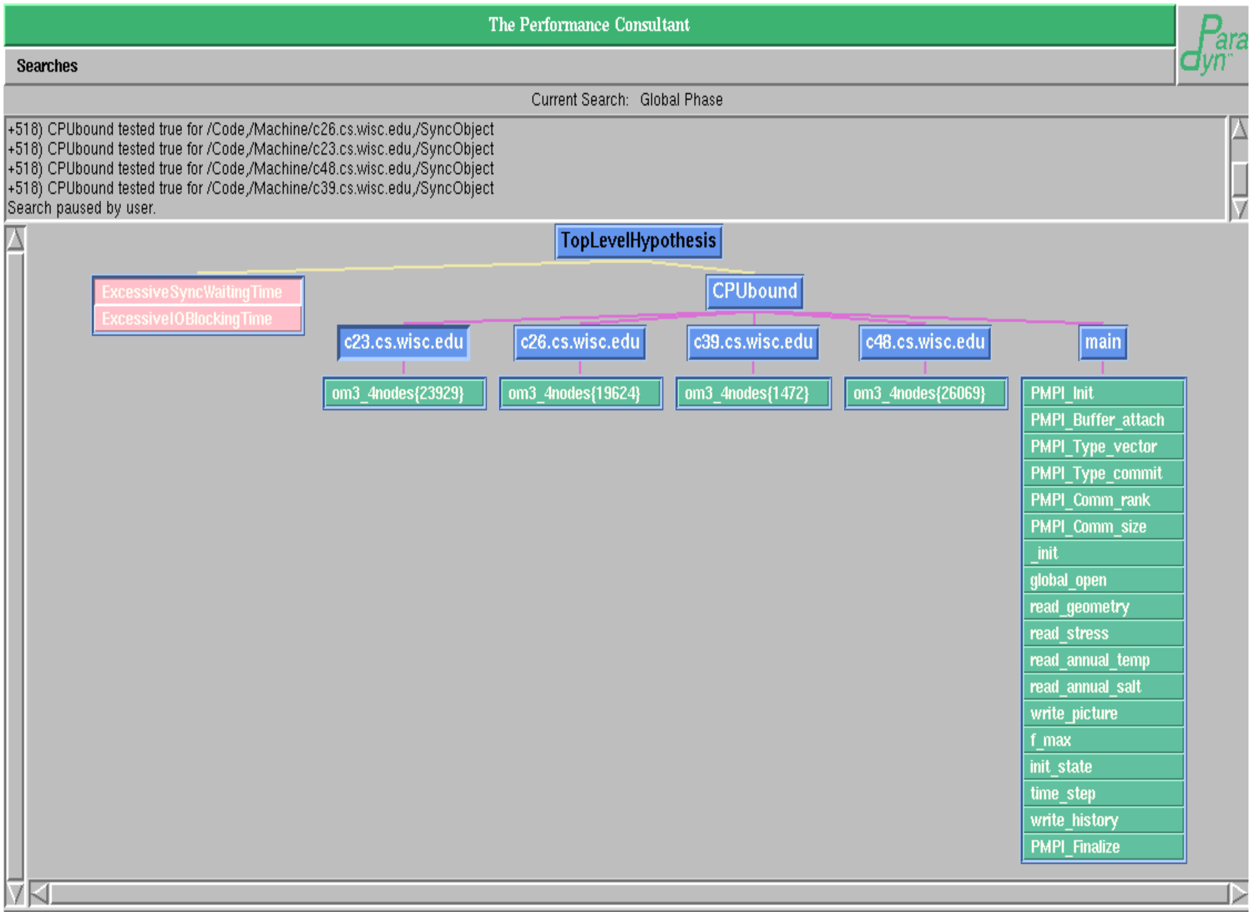

Figure 21: The Performace Consultant bottleneck search with MPI om3

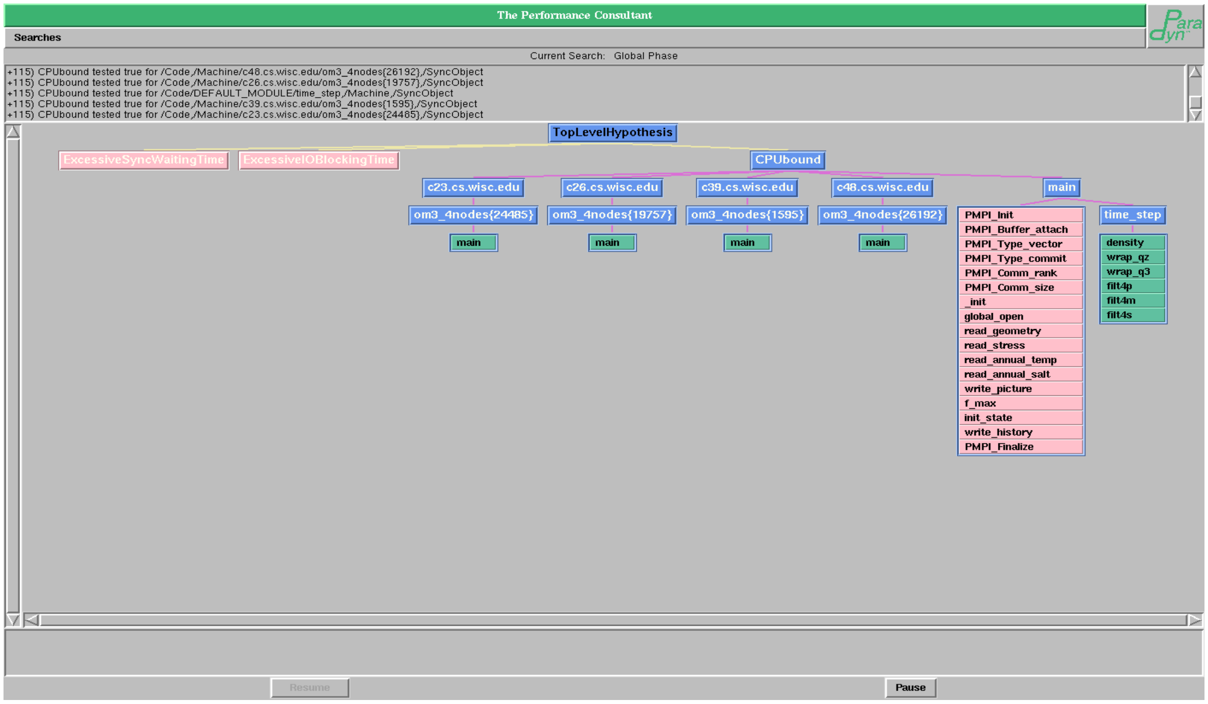

Figure 22: Search History Graph om3

4 FURTHER INFORMATION

This tutorial has not covered all of the features in Paradyn. It was intended to guide you through a few start-to-finish sessions with Paradyn, using the more common features. Note that some of the functionality shown in this tutorial differs from earlier versions of Paradyn, which are no longer supported. For a complete description of the features in Paradyn, and information on how to prepare applications for use with Paradyn, see the ***Paradyn User's Guide***.

4.1 Contacting the Paradyn developers

There are various ways to get in touch with the Paradyn developers. We are happy to try and answer questions and appreciate feedback.

e-mail: `paradyn@cs.wisc.edu`

The project e-mail address. Use this address for technical questions or requests.

Web: `http://www.cs.wisc.edu/~paradyn`

The project home page. From this page, you can find out how to get a binary or source version of Paradyn. You can also get updates and news on the current release of Paradyn.

FTP: `ftp://grilled.cs.wisc.edu/paradyn/`

The project ftp site. In the "paradyn" directory, you will find subdirectories containing the binary and source versions of the Paradyn release. Make sure to look at the README files!

FAX: +1 (608) 262-9777

Postal:
```
Paradyn Project
c/o Prof. Barton P. Miller
Computer Sciences Department
University of Wisconsin
1210 W. Dayton Street
Madison, WI  53706-1685
U.S.A.
```


■