HP PAK Performance Analysis Tools User's Guide

HP 9000 Computers

HEWLETT® PACKARD

B3476-90017

Printed in USA May 1997

Legal Notices

Contents

Contents

Preface

HP PAK Performance Analysis Tools User's Guide introduces the tools in the HP PAK Programmer's Analysis Kit. These include Puma, which analyzes program performance; TTV, which analyzes trace files produced by an instrumented thread library; and XPS, which displays the relative use of system resources by executing processes.

Comments

We welcome your comments on this manual. Please send electronic mail to `editor@ch.hp.com`, or send regular mail to:

MLL Learning Products
Hewlett-Packard Company
Mailstop: CHR 02 DC
300 Apollo Drive
Chelmsford, MA 01824

If you have any problems with the software, please contact your local Hewlett-Packard Sales Office or Customer Service Center.

Audience

This manual is written for programmers who use either C, C++, FORTRAN, or Pascal.

Manual Organization

Chapter 1	Provides an overview of the tools and instructions for starting them up.
Chapter 2	Provides information on Puma's underlying concepts.
Chapter 3	Describes Puma's command-line interface.
Chapter 4	Describes XPS.

Technical Summary

HP PAK 7.10 is an upgrade of HP PAK 7.0. The following list summarizes the differences HP PAK 7.10 and HP PAK 7.0.

- XPS is unchanged in this release.
- A new tool, TTV, has been added to HP PAK. TTV reads, synthesizes, and analyzes the trace files produced by the instrumented thread library (`libpthread_tr.sl` or `libpthread_tr.a`), and presents the resulting information in graphical form.
- Kernal thread support was added to Puma so that performance metrics can be displayed on a per-thread basis. This support is visible in the Pan/Zoom, Call Tree Analysis, and Playback windows.
- In Puma's Pan/Zoom Resource Use window, user and system CPU cycles are shown as actual counts rather than as percentages. `Exploded` `View Activity` buttons were moved to the bottom of the window.
- Puma's Call Tree Analysis textual and graphical windows were joined to a single display. You can toggle between textual (tabular) and graphical formats from one window.
- A tutorial was added to the Puma online help.
- Online help is available in Japanese as well as English.
- SJIS character support for pathnames was added.

Documentation Conventions

This manual uses the following symbolic conventions.

`literal values`	Bold monospaced words or characters in command formats and command descriptions represent commands or keywords that you must use literally. Path names are also in bold.
user-supplied values	Italic words or characters in command formats and command descriptions represent values that you must supply.
[ ]	Square brackets enclose optional items in formats and command descriptions.
|	A vertical bar separates items in a list of choices.
key	Type the corresponding key on the keyboard.
name(N)	An italicized word followed by a number in parentheses indicates a page and section number in the *HP-UX Reference*.
`Menu:Item`	A choice from the menu bar. For example, since `Exit` is on the `File` menu, the menu bar selection is written as `File:Exit`.

Printing History

B3476-90017	May 1997	Version 7.10
B3476-90016	July 1996	Version 7.0
B3476-90011	June 1995	Version 6.0

The printing date and part number at the top indicate the current edition. The printing date changes when a new edition is printed. (Minor corrections and updates which are incorporated at reprint do not cause the date to change.) The part number changes when extensive technical changes are incorporated.

We may issue a technical addendum or release notes to supplement this manual.

The software version number printed alongside the date indicates the version of HP PAK at the time this manual was printed.

HP Printing Division:

Hewlett-Packard Co.
19091 Pruneridge Ave.
Cupertino, CA 95014

1 Overview of HP PAK Tools

This chapter provides a brief overview of the HP PAK tools: Puma, TTV, and XPS. It describes how to start up the tools, and explains where to find more help.

Introduction to HP PAK

HP PAK consists of three tools that help you analyze the performance of your applications:

- Puma monitors the program counter, the call/return stack, and other performance statistics of executing programs. It saves the results in a data file. You can then view the data file in a variety of graphical formats.
- TTV (thread trace visualizer) reads, synthesizes, and analyzes the trace files produced by the instrumented thread library (`libpthread_tr.sl` or `libpthread_tr.a`), and presents the resulting information in graphical form.
- XPS (X Window System-based view of process statistics) displays the relative use of resources by processes at the system level.

NOTE Before invoking HP PAK tools, make sure that `/opt/langtools/bin` is in your search `PATH`. To access the manual pages, you must have the `/opt/langtools/share/man` directory in your `MANPATH`.

Using Puma

Puma is a tool that collects performance data from the system as a program runs. It saves the collected data to a file. You can then use Puma to analyze the information in the data file, or to play back the program's stack traces and thread states for close review.

Puma has three user interfaces:

- The graphical user interface, based on the conventions of OSF/Motif, is useful for interactive collection and analysis of program performance data.
- The command-line user interface is useful for collecting or analyzing data in a non-graphical, shell environment.
- The shell script interface is a special command-line interface that lets you execute a single Puma command from a shell or a shell script.

Invoking Puma's Graphical User Interface

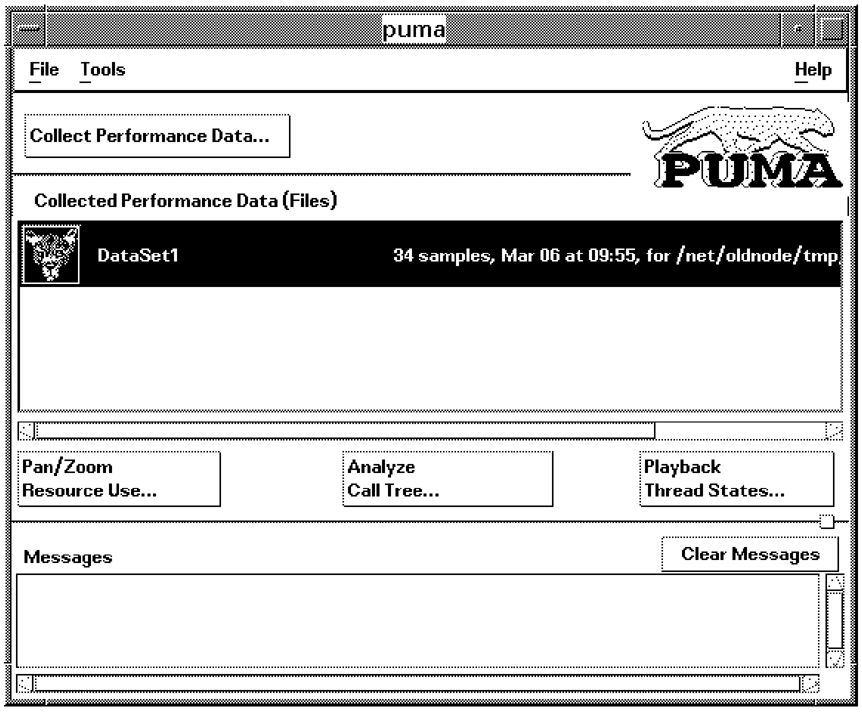
To invoke Puma's graphical user interface, enter the following command at the shell prompt:

```
puma
```

Puma displays its main window, as shown in Figure 1-1.

To exit Puma at any time, click on `Exit` in the `File` menu.

Figure 1-1 **The Main Puma Window**


puma
File
Tools
Help
Collect Performance Data...
PUMA
Collected Performance Data (Files)
DataSet1
34 samples, Mar 06 at 09:55, for /net/oldnode/tmp
Pan/Zoom Resource Use...
Analyze Call Tree...
Playback Thread States...
Messages
Clear Messages

The Puma main window is your starting point for all Puma activities. For example:

❑ The `Collect Performance Data` button brings up windows that guide you through the process of collecting performance data.

❑ The `Collected Performance Data (Files)` list box displays Puma data files that exist in the current working directory.

❑ The three buttons below the list box give you a choice of ways to display performance data:

- Select `Pan/Zoom Resource Use` to display a program's use of system resources over time in line-graph form. On systems that support kernel-threaded applications, you can also view how individual kernel threads use system resources.

    - Select `Call Tree Analysis` to display a given performance metric relative to the libraries and routines that executed during the sample set.
    - Select `Playback Thread States` to step through the stack trace of multi-threaded programs (on systems that support kernel-threaded applications).

❏ The `Messages` box displays any run-time messages generated by Puma.

For More Information

See Chapter 2, “Puma Concepts,” for more information on how Puma collects data and generates statistics.

See the *puma*(l) man page.

Detailed online help on all of Puma's functions is available from Puma's `Help` menus. The online help also includes a tutorial that guides you through data collection and analysis.

Here are a few tips on using Puma's online help:

- To browse the Puma Help volume, begin by selecting `Help:Introduction` in the main Puma window and then follow hyperlinks throughout the volume.
- To find help on using a particular window or dialog box, click on the `Help` menu or button in that display.
- To find help on a particular screen item (button, list box, etc.), select `Help:On Item` in the main Puma window and then click on the desired item in any Puma window.
- To find information on searching and printing helpfiles, select `Help:On Help` in the main Puma window.

Invoking Puma's Command-Line Interface

To invoke Puma's command-line interface, enter the `puma -text` command in a shell. The `puma>` prompt will be displayed. For example:

```
$ puma -text
puma>
```

To exit Puma, enter `exit` or `quit`.

For More Information

See Chapter 3, “Puma Command Quick Reference,” for more information on Puma commands and on the Puma command-line interface.

See Chapter 2, “Puma Concepts,” for more information on how Puma collects data and generates statistics.

See the *puma*(l) man page.

You can get online help from the command-line interface by entering one of the following commands at the `puma>` prompt:

`help`	Displays a list of help topics.
`help` *topic*	Displays syntactical help for the specified *topic*.
`man` *command*	Displays a full description of the specified *command*.

Using Puma's Shell Script Interface

You can invoke Puma from a shell or shell script to execute a single command. Enter the command `puma` and the name of the desired command as an *option*, that is, precede the command name with a hyphen. You can also supply arguments to the command option.

For example, to invoke and collect performance data on `my_program` (with arguments) from a shell or shell script, use a command like the following:

```
puma -monitor -invoke my_program arg1 arg2
```

As another example, suppose that `my_program` is already running, First, you need to find the Process ID (PID) with a command like the following:

```
ps | grep my_prog
1054
```

Then, you can invoke the following command line from a shell or shell script:

```
puma -monitor -existing 1054 -executable my_program
```

Notice that you cannot specify arguments for an existing process.

For More Information

See Chapter 3, “Puma Command Quick Reference,” for more information on Puma commands.

See Chapter 2, “Puma Concepts,” for more information on how Puma collects data and generates statistics.

See the *puma*(l) man page.

A Sample Program

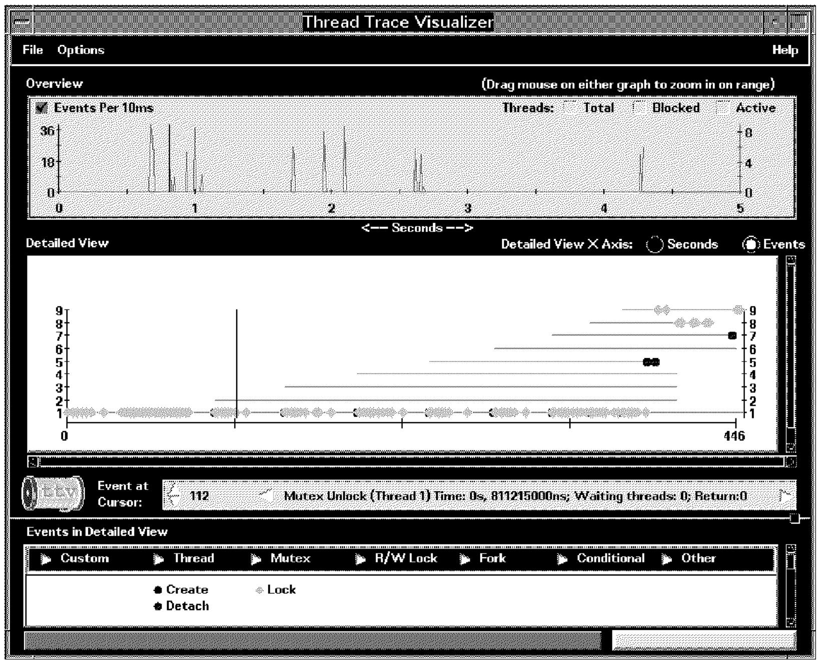
You can experiment with Puma by using the sample program, `vanderbilt`, that is included with HP PAK. This program is used in the online tutorial that illustrates most of Puma's features. Select `Help:Tutorial` from the main window of Puma's graphical user interface. `vanderbilt` should be installed in the following location:

```
/opt/langtools/hppak/examples/vanderbilt
```

Using TTV

The TTV (Thread Trace Visualizer) is a utility that reads the trace files produced by the instrumented thread library (`libpthread_tr.sl` or `libpthread_tr.a`). TTV provides a graphical format for the presentation and analysis of the data (see Figure 1-2).

Figure 1-2 **TTV Display**



```
112    Mutex Unlock (Thread 1) Time: 0s, 811215000ns; Waiting threads: 0; Return:0
```

To produce thread traces, you need to link your application to the instrumented thread library. For details on using the instrumented thread library and producing thread traces, refer to the file `/usr/include/sys/trace_thread.h`.

When you enter the `ttv` command, you can supply it with a list of trace files. For example, if your trace files are stored in the directory `my_traces`, and the threads of interest are for process 1542, you would invoke TTV as follows:

```
ttv my_traces/tr.1542.*
```

Alternatively, you can start ttv without a list of trace files. You can then load trace files from the `File` menu in the grapical user interface.

You can also invoke TTV from the Tools menu in the main Puma window.

The data from your trace files are presented in the two graphs in the TTV main window. You can change the scope and content of the graphs in many ways.

For More Information

Online help is available from TTV's `Help` menu or from the *ttv*(1) man page.

Using XPS

XPS provides a graphical display of CPU and I/O usage by the processes that are currently executing. The display is updated periodically.

To run XPS, enter

```
xps
```

at the shell prompt. An XPS screen, like the one in Figure 1-3, appears.

Scroll up and down in the XPS display using standard OSF/Motif tools (scroll bars or paging keys).

To end XPS's monitoring at any time, press **q** or **Ctrl-c** anywhere in the XPS display.

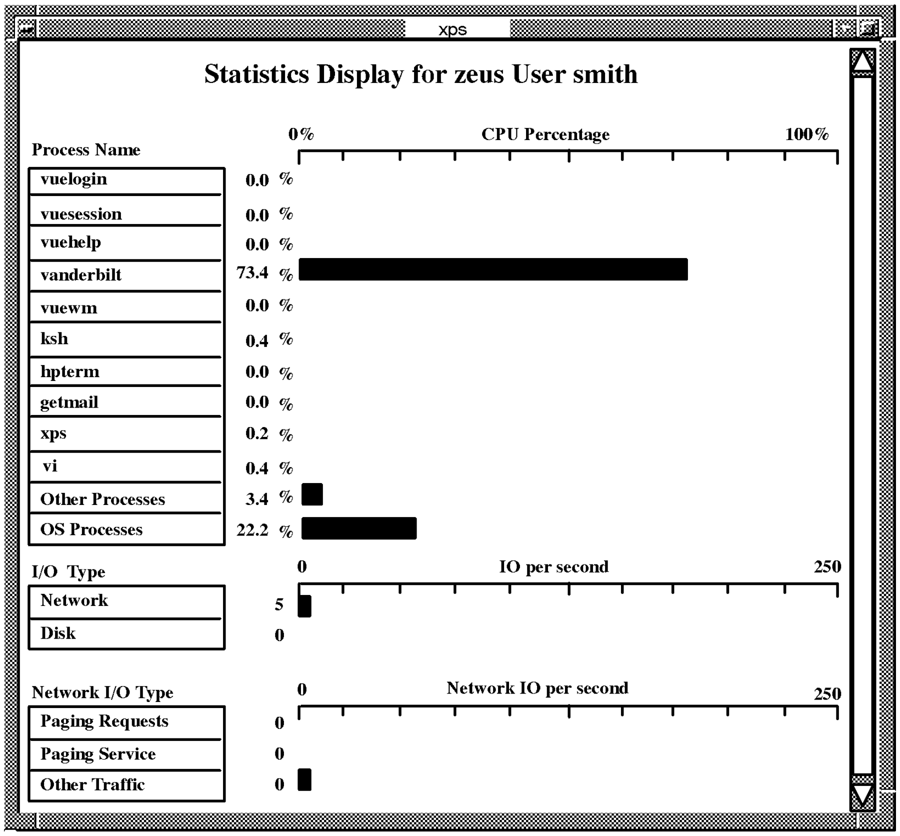
For More Information

See Chapter 4, “XPS Overview, Options, and Commands,” for detailed information on XPS.

You can also find information on XPS online in the *xps*(1) man page.

NOTE XPS is provided as an analysis tool for programmers. If you need to monitor system performance on an ongoing and regular basis, we recommend that you use the HP GlancePlus product.

Figure 1-3 **XPS Display**



Statistics Display for zeus User smith

Process Name		CPU Percentage (0% – 100%)
vuelogin	0.0 %	
vuesession	0.0 %	
vuehelp	0.0 %	
vanderbilt	73.4 %	
vuewm	0.0 %	
ksh	0.4 %	
hpterm	0.0 %	
getmail	0.0 %	
xps	0.2 %	
vi	0.4 %	
Other Processes	3.4 %	
OS Processes	22.2 %	

I/O Type		IO per second (0 – 250)
Network	5	
Disk	0	

Network I/O Type		Network IO per second (0 – 250)
Paging Requests	0	
Paging Service	0	
Other Traffic	0	

2 Puma Concepts

This chapter is a general description of how Puma gathers data and generates statistics.

Procedure Relationships

The terms that describe the relationship of a given procedure to other procedures in the same program are:

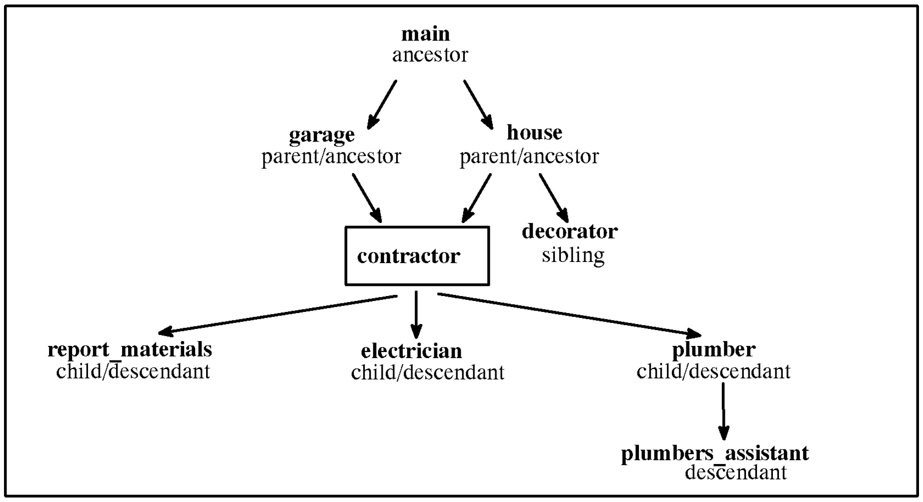
parent	A procedure that directly calls the given procedure.
ancestor	A procedure that calls the given procedure, either directly or through another procedure or procedures; all parents are also ancestors.
child	A procedure called directly by the given procedure.
descendant	A procedure that is called by the given procedure, either directly or through other procedures; all children are also descendants.
sibling	A procedure with the same parent as the given procedure.

Figure 2-1 illustrates these relationships in the program `vanderbilt` for the procedure `contractor`.

NOTE `vanderbilt` is a sample program included with HP PAK. It should be installed in the following location:

```
/opt/langtools/hppak/examples/vanderbilt
```

Figure 2-1 vanderbilt Procedure Relationships



To Puma, the relationships among procedures are *dynamic*, reflecting the possibility that sometimes *x* may call *y* and later *y* may call *x*. Puma analyzes your program as it is seen at run time; Puma does not base the analysis on the lexical structure of the program as seen at compile time.

For example, if a procedure calls an error-handling routine, Puma would report the error-handling routine as a child of the procedure only if, during the execution of the procedure, control actually branched to the error-handling routine *and* a sample was taken during the execution of the error-handling routine.

How Puma Gathers Data

To collect performance data, Puma performs the following steps at each sampling interval:

1. Stop the program.
2. Take a sample from the program's image.
3. Store the sample in a memory buffer.
4. Release the program, allowing it to resume execution.

A sample is a unit of data gathered by Puma. A sample contains stack trace information and statistical information about the current state of the program.

Stack Traces

A stack trace is a snapshot of the program's call/return stack. It is an ordered sequence of the procedures that are active when the sample is taken.

Puma considers a procedure to be *active* (on the stack) if control is within the procedure itself or in any of its descendants. When a procedure is active, Puma's later analysis will show that time is being spent *in-or-under* that procedure. When the program counter is in a procedure, Puma's analysis will show that time is being spent *in-only* that procedure.

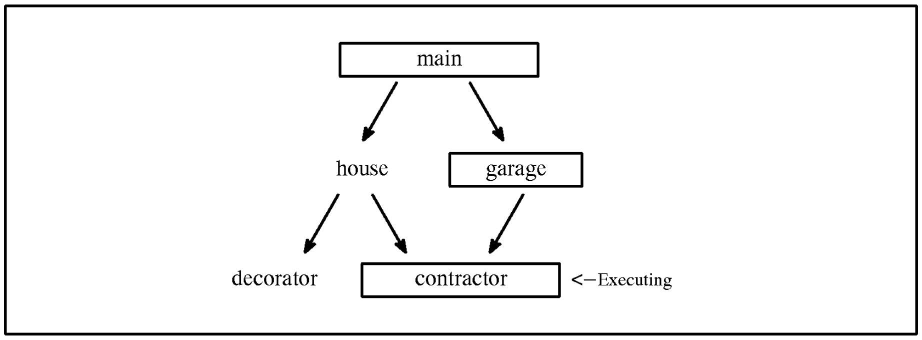
Figure 2-2 highlights the procedures that are active when `contractor` is executing, and `garage` has called `contractor`.

If Puma takes a sample when `contractor` is executing, the stack trace returned looks this way:

```
\main\garage\contractor
```

The last procedure in a stack trace is the currently executing procedure. The other procedures in the list are the executing procedure's ancestors.

Figure 2-2 **Active Procedures**



Statistical Information

Puma records statistical information concerning the program's resource use, including:

- time spent in user space ("user cycles")
- time spent in system space ("system cycles")
- major page faults
- minor page faults
- socket messages sent
- socket messages received
- signals received
- files open
- page swaps
- characters (bytes) of I/O

Data Files

Puma stores the samples it collects in a *data file*. You can have any number of data files, and any legal UNIX filename is acceptable.

By default, Puma names its data files `DataSet`*number*`.puma` and places them in the directory where Puma is executing. In graphical mode, Puma increments *number* each time data collection begins. In shell or command-line mode, Puma either creates or overwrites `DataSet1.puma`. Use the `-dat` option to create a data file with a different name.

When invoked in graphical mode, Puma displays icons for the data files that reside in the directory where it is executing, and that have the filename extension `.puma`. You can load data files with other filename extensions manually from within Puma.

To reduce the overhead cost of collecting data, Puma does not write data immediately upon collecting it. Instead, Puma stores data in memory and only writes to a data file when:

- you exit Puma
- you restart data collection
- you select some older data file for analysis

Sample Interval

The *sample interval* is the rate at which Puma takes data samples.

The sample interval refers to the amount of time that Puma allows the target program to run before suspending it to record information.

The phrase *samples per second* refers to the number of samples gathered for each second that the target program has been allowed to run. The number of samples gathered by Puma per elapsed (real) second depends upon the amount of overhead Puma incurs between each sample. Overhead includes the time it takes to stop the target, record information, update the graphical display, and restart the target.

By default, the sample interval is 100, which means that Puma gathers a sample every 100 milliseconds of the target process execution time (at a rate of 10 samples per second). This interval is under user control.

How Puma Generates Statistics

The information that Puma produces from a data file consists of statistical summaries for the run of the program. For example, Puma can report the percentage of samples in which each procedure was active (that is, how often control was in the procedure itself or in any of its descendants).

Puma can present statistical information in a variety of ways. For example, you can:

- Restrict Puma's analysis to a contiguous subset of the samples taken.
- View the statistics in a way that reflects the program's dynamic call structure.

The online documentation describes the various ways you can control the analysis.

How Puma Generates Procedure Names

In the Call Tree Analysis windows, you can direct Puma to display performance metrics by procedure (routine). This section describes the naming conventions that Puma uses when displaying procedure names.

When Puma can determine a procedure's name, it displays the name in case-correct format. For example:

```
XOpenDisplay
```

For procedures in dynamically loaded libraries, Puma cannot identify the procedure names unless you identify the library before data collection begins (using the `Options` menu on the Data Collection window). The default entry in the `Libraries` option is `*.c`. This entry causes Puma to collect data from all dynamically loaded libraries, but, without the actual library names, it does not load their symbol tables. Puma cannot, therefore, determine the names of the procedures defined in the libraries. In this situation, Puma displays the full pathname of the dynamically loaded library rather than the procedure name. For example:

```
/usr/shlib/libc.sl
```

There are two conditions under which Puma is unable to determine the name of the procedure associated with a call frame:

- The procedure's name was omitted from the binary, as dictated by the options used in compilation and linking.
- The procedure has a nonstandard call frame format (for example, for some assembly language procedures).

In this situation, Puma displays the virtual address of the procedure rather than its name. For example:

`va(`*xxxx*`)`

where *xxxx* is the `va` (virtual address) of the procedure.

How Puma Analyzes Recursive Procedures

When procedures in a program are recursive, the call structure at run time (the dynamic call structure) can be more complex than the order of procedure calls that is apparent before run time (the static call structure). The `Recursion Collapsing` area of the Call Tree Analysis window allows you to specify how you want Puma to collapse its reporting of recursive routines.

Recursion Collapsing

A program with recursion can have a deceptively large call tree. Showing the many paths to a given routine might give you more information than you need while obscuring important patterns of recursion.

Suppose, for example, you have a routine named `main` that calls routine `r`; routine `r` calls itself recursively to Level 3, and `r` also calls routine `s`. If you got one sample from each of the possible stack traces, you would have the following:

```
main r
main r s
main r r
main r r s
main r r r
main r r r s
```

A hierarchical analysis (in text-mode) might look like this:

```
Samples         Samples
In or Under     In Only
Raw Count       Raw Count     Level
------------    ---------     -----
6               0             1  main
6               1             2   r
1               1             3    s
4               1             3    r
1               1             4     s
2               1             4     r
1               1             5      s
```

(This analysis uses the raw count for clarity; the percentage or parent percentage would not reflect a strictly accurate picture, as the percentages are approximations. The in-only raw counts of the nested procedures add up to the in-or-under raw count of 6 for `main`.)

This analysis, although correct, does not properly summarize the program's behavior. It does not show that three of the six stack traces are of the form `main [r…] r` and three are of the form `main [r…] s`. Recursion collapsing remedies this. When you request recursion collapsing, Puma does not make a new node in a tree of call chains if it encounters a recursive call. Instead Puma creates a *recursive stub* that represents the recursion and refers to the place higher up the call chain where the same routine occurs. Puma then jumps up in the tree to that point and continues playing out the stack trace. In effect, call chains are truncated. The following sample analysis presents a collapse of the recursion in the earlier example.

```
Samples         Samples
In or Under     In Only
Raw Count       Raw Count      Level
-----------     ---------      ------
6               0              1  main
6               3              2   r
1 (2)           3              3    s
                               3    r...
```

The first line of this analysis is the same as the earlier one. The second line, for `r`, shows all three `main [r…] r` calls attributed in the in-only column to `r`, since levels have been collapsed. The third line shows three samples attributed to `s`. The notation `1 (2)` indicates one sample at this level and two at lower levels. In other words, there were three stack traces that passed through this point in the tree; one got there directly through the ancestors as shown, while two got there after some skipping through the recursive stubs. Note that only in-or-under data is split up according to whether recursive stubs were traversed; the in-only data is not. The fourth line, `r…`, represents the recursive stub. It indicates that the parent routine (`r` at Level 2) called the child (`r` at Level 3), and a stub was built pointing up to the parent.

Puma Recursion Collapsing Options

There are four degrees of recursion collapsing to choose from in a Puma analysis. They are, in order of increasing presence of recursion collapsing:

`No collapse`	Performs no recursion collapsing.
`Direct collapse`	Performs recursion collapsing for direct recursion (that is, if a routine calls itself directly).
`Conservative collapse`	Performs recursion collapsing unless doing so would omit any routine names out of the call chain.
`Full collapse`	Performs recursion collapsing whenever it encounters a routine for which there is a higher instance in the call tree.

No Collapse

As its name implies, the `No collapse` option causes Puma to perform no recursion collapsing at all. In the example above, `No collapse` would produce the first of the two analyses.

Direct Collapse

The `Direct collapse` option indicates that a recursive stub should be used only for direct recursion, that is, only if the prior instance of a routine in the call tree is the immediate parent of the current instance. For instance, the call chain `main a a a b` would have a stub built from the second `a` and would effectively be collapsed to `main a b`. However, the call chain `main a b a b` would *not* have recursive stubs built, because neither the recursive call to `a` nor the recursive call to `b` is direct.

Conservative Collapse

The `Conservative collapse` option builds recursive stubs in more circumstances than the `Direct collapse`. It builds a recursive stub if doing so would not lose any routine names out of the call chain; that is, if the routines being cut out are duplicated higher up in the stack trace. For instance, the call chain `main a b a b` would not have a recursive stub built at the recursive call to `a`, since the transformation from `main a b a` to `main a` loses the only instance of `b`. However, it would have a recursive stub at the recursive call to `b`, since the collapsing of `main a b a b` to `main a b` loses an instance of `a`, but leaves another one. As another example, the call chain `main a b c b a d c` would not have any recursive stubs using `Conservative collapse`. Building a stub at the recursive second call to `b` would lose `c`; building one at the recursive second call to `a` would lose `b` and `c`; and building one at the recursive second call to `c` would lose `d`.

Full Collapse

The `Full collapse` option causes a recursive stub to be built whenever there is any higher instance of a routine in the call tree. For example, for `main a b c b a d c`, the call chain that is discussed above, a stub would be created from the recursive call to `b`, and another at the recursive call to `a`. There would not be one at the recursive call to `c`, because after the other recursive jumps the other instance of `c` is no longer higher in the tree. In particular, the analysis just from this one stack trace would look like the following example.

```
Samples         Samples
In or Under     In Only
Raw Count       Raw Count      Level
-     -     -
1               0              1  main
1               0              2   a
1               0              3    b
1               0              4     c
                               5      b...
                               4     a...
0 (1)           0              3    d
0 (1)           1              4     c
```

When Puma encounters the second `b` in the call chain `main a b c b a d c`, it builds a recursive stub (at Level 5 in the analysis) up to the higher `b` (at Level 3). It then jumps up to that `b`. When it comes to the second `a`, Puma builds a recursive stub (the `a` at Level 4) as a child of the instance of `b` at Level 3, and pointing up to the instance of `a` at Level 2. For `d`, Puma builds an ordinary node (not a recursive stub). For the final `c` (the last line of the analysis), it does not build a recursive stub, because the first `c` is not an ancestor of the current node. The ancestors of the current node are `d` , `a` (at Level 2), and `main` (at Level 1). The last two lines have 0 as their direct value and 1 as their indirect value, because those nodes were reached only after traversing recursive stubs. When a node, such as `a` at Level 2, is reached directly and then again in the same stack trace through a recursive stub, the direct arrival counts; that is why these lines have a direct value of 1.

3 Puma Command Quick Reference

This chapter provides a quick summary of the commands available in the Puma command-line and shell script interfaces. .

Data Collection Commands

`mo[nitor]` [*monitor_options*]	Perform data collection, using the given options.
`sh[ow] mo[nitor]`	Show the current default data-collection options.
`se[t] mo[nitor]` *monitor_options*	Modify the default data-collection options.
`pr[ocesses]`	List the processes currently executing on the machine.

Data Analysis Commands

`an[alyze]` [*analyze_options*]	Generate an analysis, using the given options.
`sh[ow] an[alyze]`	Show the current default analysis options.
`se[t] an[alyze]` *analyze_options*	Modify the default analysis options.
`th[reads]`	List the threads for which data has been collected in the currently selected data file.

Miscellaneous Commands

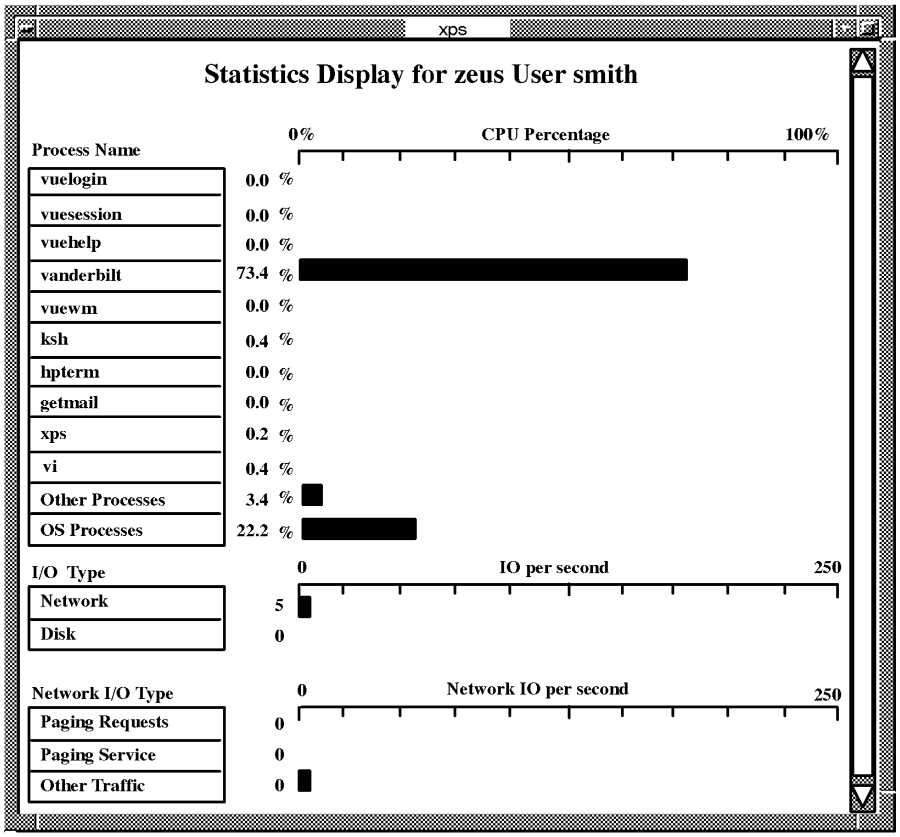
`e[xit]`	Exit from Puma command interface.
`he[lp]` [*command*]	Provide general help information (when no argument) or help information on a Puma command (when *command* is supplied).
`ma[n]` *command*	Provide a full description of the specified Puma command.
`q[uit]`	Exit from Puma command interface.
`sh[ow] al[l]` [>*filename*]	Show defaults for data-collection and analysis commands; optionally write them to *filename*.
`ve[rsions]`	Display the version number of Puma and whatever managers are loaded.

4 XPS Overview, Options, and Commands

This chapter contains a brief description of XPS, including information on its options and commands.

XPS Overview

Figure 4-1 **XPS Display**



```
Statistics Display for zeus User smith

Process Name              0%        CPU Percentage        100%
vuelogin           0.0 %
vuesession         0.0 %
vuehelp            0.0 %
vanderbilt        73.4 %
vuewm              0.0 %
ksh                0.4 %
hpterm             0.0 %
getmail            0.0 %
xps                0.2 %
vi                 0.4 %
Other Processes    3.4 %
OS Processes      22.2 %

I/O  Type                 0         IO per second         250
Network              5
Disk                 0

Network I/O Type          0      Network IO per second    250
Paging Requests      0
Paging Service       0
Other Traffic        0
```

XPS displays process and I/O usage dynamically, using a bar chart interface based on OSF/Motif.

For each process that you own, XPS displays a separate bar showing the relative percentage of CPU time being consumed by that process.

You can display other user processes and OS server processes individually, with a separate bar for each process. Or, you can collapse these processes into groups, with a single bar for each group. The options that control these and other features are listed in “XPS Options” on page 34 and online on the `xps(1)` man page.

When you invoke the `xps` command, you will see a display like that shown in Figure 4-1.

The `Process Name` section provides the names of your processes. By default, XPS groups all other user processes under `Other Processes` and all operating system processes under `OS Processes`. To expand `Other Processes` and display the processes individually, use the `-p` option. To expand `OS Processes` and display the processes individually, use the `-l1` option.

The `I/O Type` section provides the following information:

`Network`	The number of network messages sent or received per second within the sample interval.
`Disk`	The number of disk reads or writes per second within the sample interval.

The `Network I/O Type` section provides the following information:

`Paging Requests`	The number of requests to read or write per second from the monitoring machine within the sample interval.
`Paging Services`	The number of requests received to read or write per second from other machines within the sample interval.
`Other Traffic`	The number of network messages sent or received per second that are unrelated to paging requests and paging services within the sample interval.

NOTE XPS is provided as an analysis tool for programmers. If you are interested in monitoring system performance on an ongoing and regular basis, we recommend that you use the HP GlancePlus product.

XPS Options

Specify *options* as a single option or as a list of options separated by spaces. The available options are:

`-r` *n*	Update the display every *n* seconds. By default, the display is updated every 4 seconds.
`-p`	Expand the Other Processes group to show a bar for each process. Other processes are all user processes other than the processes that you own. By default, XPS groups all these processes and displays them with a single bar.
`-os`	Group all OS server processes and display them with a single bar. This is the default.
`-l1`	Expand the OS Processes group to show a bar for each process.
`-m`	Show a bar for idle (that is, unused) CPU time. XPS calculates this value as 100% minus the sum of all the current process times. By default, this bar is not displayed.
`-a`	Show all information (same as `-p -l1 -m`).
`-large`	Use large font for display. This is the default.
`-small`	Use small font for display.

XPS Window Commands

While XPS is running, you can use various keys to move within the window, to get help, and to exit.

Table 4-1 shows the definitions for various keys while XPS is running.

Table 4-1 **XPS Key Definitions**

Task	Predefined Key
Scroll backward 1/2 window	**Prev Page Up**
Scroll forward 1/2 window	**Next Page Down**
Scroll backward 1 line	**Shift-Up Arrow**
Scroll forward 1 line	**Shift-Down Arrow**
Move to top	**Ctrl-T**
Move to bottom	**Ctrl-B**
Help	**Ctrl-H Shift-H**
Exit	**Ctrl-C Ctrl-N Q q**

Glossary

Active A state of a procedure. Puma considers a procedure to be active if control is in the procedure itself or in any of its descendants.

Ancestor A procedure that calls the given procedure, either directly or through another procedure or procedures. Contrast with *parent*.

Child A procedure called directly by the given procedure. Contrast with *descendant*.

Conservative collapse An analysis specification that instructs Puma to perform recursion collapsing unless doing so would omit any routine names out of the call chain.

Context The collective name for Puma's target program, the directory where the target program executes, and the pathname of the file to which Puma writes the collected data. Puma context also includes any search directories you have provided for target source files.

Count A Puma analysis mechanism that produces a raw count (as opposed to a percentage) for a given statistic in the analysis.

Data file A file in which Puma stores the data that it collects from a process. Data files are used during analysis and playback.

Descendant A procedure that is called by the given procedure, either directly or through other procedures. Contrast with *child*.

Direct collapse An analysis specification that instructs Puma to perform recursion collapsing only for direct recursion (that is, when a routine calls itself directly).

Flat report A Puma analysis that indicates the overall percentage of time spent in-only and/or in-or-under each procedure, regardless of the procedure's ancestors. A flat report does not reflect the dynamic calling structure of a program. Contrast with *Hierarchical report*.

Flow See *Program flow*.

Full collapse An analysis specification that instructs Puma to perform recursion collapsing whenever it encounters a routine for which there is a higher instance in the call tree.

Glossary

Granularity The degree of detail in which Puma displays analysis data. Data may be shown separately for each routine or library of a program; or data may be aggregated and associated with the top-level caller. The former granularity is called "in-only"; the latter is called "in-or-under."

Hierarchical report A Puma analysis that reflects the dynamic calling structure of a program, giving a structured breakdown of where the program spends its time. Contrast with *Flat report*.

In-only A term used to refer to execution time spent within a procedure but not within its descendants. Contrast with *in-or-under*.

In-or-under A term used to refer to execution time spent within a procedure and/or its descendants. Contrast with *in-only*.

Interval See *sample interval*.

Level cutoff A specification that limits the nesting depth of a Puma hierarchical analysis to a certain number of levels. The default is a level cutoff of 64.

Library As a Puma data-collection option, the term *library* refers to a dynamically loaded library; that is, a collection of executable routines bound together. Library routines are loaded into the target program when one of the routines is called by the target program.

Limit The lowest percentage of execution time included in a Puma display. The default limit is 1.

Monitor To collect performance data from an executing program.

No collapse An analysis specification that instructs Puma to collapse none of the recursion of a program, but instead to show the full dynamic tree.

Overhead The time spent by Puma recording and displaying a sample during data collection.

Parent A procedure that directly calls the given procedure. Contrast with *ancestor*.

Percentage A Puma analysis mechanism that produces a statistical value as a percentage of the value for the whole set of samples being analyzed.

Glossary

Percent cutoff A specification that restricts entries in a Puma analysis to procedures consuming more than a certain amount of execution time. By default, the percent cutoff is 0, which, in effect, instructs Puma to include information about every procedure that consumes 0% or more of the execution time (that is, every procedure that occurred in any stacktrace that Puma recorded).

Performance statistics A set of metrics that Puma collects from a program run to aid in analyzing performance. These include time spent in user space ("user cycles"), time spent in system space ("system cycles"), major page faults, minor page faults, socket messages sent, socket messages received, signals received, files open, page swaps, and characters (bytes) of I/O.

Play back To use data accumulated while monitoring a program to examine the sequence of the execution of the program.

Procedure cutoff A cutoff that excludes calls beneath a given procedure from the analysis.

Program call tree A graphical image of the currently active stack trace and the siblings of each procedure in that stack trace.

Program counter A register in the CPU that contains the address of the next instruction to be executed.

Program flow The procedure path through which the execution of a program has gone to bring the program counter to its current location.

Process A binary executed by the CPU. Each process is executed independently.

Recursion Any situation in which procedure calls are circular; for example, when a procedure calls itself, when a procedure calls the procedure that called it, or when a procedure calls farther back up the call chain.

Glossary

Recursion collapsing A Puma analysis mechanism for reporting on the recursion of a program. When you request recursion collapsing, Puma may not make a new node in a tree of call chains if it encounters a recursive call (depending on the type of recursion collapsing you specify). Instead Puma creates a recursive stub that represents the recursion and refers to the place higher up the call chain where the same routine occurs. Puma then jumps up in the tree to that point and continues playing out the stack trace.

Resources See *Performance statistics*.

Sample The data obtained from one interruption of an executing program. The sample includes statistical information and a stack trace.

Sample interval The period of time between samples taken during data collection.

Samples per second The number of samples Puma gathers per second that a target program has been allowed to run during data collection.

Sibling A procedure with the same parent as the given procedure.

Stack Trace The sequence of procedure calls leading to the program counter at any given time.

Step To reconstruct the execution of a program one sample at a time.

Target program A program from which Puma collects performance data.

Thread An independent stream of program execution. A program may be made up of one or more threads; multiple threads in a program cooperate in solving a common problem.

Virtual CPU time The time the processing unit spends in the user's program, excluding time spent waiting for I/O or executing other programs.

Wall-clock time The total time taken for program execution, including disk and memory accesses, I/O, and operating system overhead.

Index

Index

Index