VMS Software

VSI OpenVMS

VSI DECset for OpenVMS Guide to Performance and Coverage Analyzer

VSI DECset for OpenVMS Guide to Performance and Coverage Analyzer

VMS Software

Preface

This guide explains how to use the VSI Performance and Coverage Analyzer (PCA) on the DECwindows user interface. The guide describes the following two components of PCA:

- The Collector, which gathers various kinds of performance and test coverage data on your program

- The Analyzer, which processes and displays that data graphically

To use the character-cell terminal version of the PCA Collector or Analyzer in place of DECwindows, enter the appropriate command to define the terminal display logical:

```
$ DEFINE PCAC$DECW$DISPLAY " "
$ DEFINE PCAA$DECW$DISPLAY " "
```

To return to the DECwindows version of the PCA Collector or Analyzer, deassign either logical name, as follows:

```
$ DEASSIGN PCAC$DECW$DISPLAY
$ DEASSIGN PCAA$DECW$DISPLAY
```

In addition, you can specify your interface at the command line, as follows:

- /INTERFACE (invokes the character-cell format)

- /INTERFACE=CHARACTER_CELL

- /INTERFACE=DECWINDOWS

- /INTERFACE=MOTIF

1. About VSI

VMS Software, Inc., (VSI) is an independent software company licensed by Hewlett Packard Enterprise to develop and support the OpenVMS operating system.

VSI seeks to continue the legendary development prowess and customer-first priorities that are so closely associated with the OpenVMS operating system and its original author, Digital Equipment Corporation.

2. Intended Audience

This guide is intended for experienced programmers and technical managers who are concerned with the performance of their application programs. PCA serves as a flexible and reliable tool that you can use to:

- Analyze the performance characteristics of your applications

- Analyze the test coverage characteristics of the tests you run on your applications to determine what code paths the tests are executing.

3. Document Structure

This guide contains the following chapters:

- Chapter 1 introduces PCA and describes a methodology for using PCA effectively.

- Chapter 2 shows how to use the PCA Collector to gather performance and coverage information about your application.

- Chapter 3 shows how to use the PCA Analyzer to analyze and interpret the performance and coverage information the Collector has gathered.

- Chapter 4 shows how you can interpret the elapsed and process time information to locate general performance bottlenecks.

- Chapter 5 shows how you can interpret execution counts information to locate specific performance bottlenecks.

- Chapter 6 explains how to find which parts of your application are covered or not covered by tests.

- Chapter 7 shows how you can determine how your application uses system resources: system services and I/O services.

4. Related Documents

The following documents might be helpful when using PCA:

- *VSI DECset for OpenVMS Installation Guide* gives instructions for installing PCA.

- *VSI Fortran Performance Guide* details the performance features of the VSI Fortran-High Performance Option (HPO) and discusses ways to improve the run-time performance of VSI Fortran programs.

- *VSI DECset for OpenVMS Performance and Coverage Analyzer Reference Manual* describes all the commands available in PCA.

- *VSI DECset Performance and Coverage Analyzer for OpenVMS Command-Line Interface Guide* describes how to use the command-line interface to PCA.

5. References to Other Products

Some older products that VSI DECset components previously worked with might no longer be available or supported by VSI. Any reference in this manual to such products does not imply actual support, or that recent interoperability testing has been conducted with these products.

Note

These references serve only to provide examples to those who continue to use these products with VSI DECset.

Refer to the Software Product Description for a current list of the products that the VSI DECset components are warranted to interact with and support.

6. OpenVMS Documentation

The full VSI OpenVMS documentation set can be found on the VMS Software Documentation webpage at https://docs.vmssoftware.com.

---

7. VSI Encourages Your Comments

You may send comments or suggestions regarding this manual or any VSI document by sending electronic mail to the following Internet address: `<docinfo@vmssoftware.com>`. Users who have VSI OpenVMS support contracts through VSI can contact `<support@vmssoftware.com>` for help with this product.

8. Conventions

VMScluster systems are now referred to as OpenVMS Cluster systems. Unless otherwise specified, references to OpenVMS Cluster systems or clusters in this document are synonymous with VMScluster systems.

The contents of the display examples for some utility commands described in this manual may differ slightly from the actual output provided by these commands on your system. However, when the behavior of a command differs significantly between OpenVMS Alpha and Integrity servers, that behavior is described in text and rendered, as appropriate, in separate examples.

In this manual, every use of DECwindows and DECwindows Motif refers to DECwindows Motif for OpenVMS software.

The following conventions are also used in this manual:

Convention	Meaning
Ctrl/ *x*	A sequence such as **Ctrl/** *x* indicates that you must hold down the key labeled Ctrl while you press another key or a pointing device button.
PF1 *x*	A sequence such as PF1 *x* indicates that you must first press and release the key labeled PF1 and then press and release another key or a pointing device button.
Return	In examples, a key name enclosed in a box indicates that you press a key on the keyboard. (In text, a key name is not enclosed in a box.)
…	A horizontal ellipsis in examples indicates one of the following possibilities: • Additional optional arguments in a statement have been omitted. • The preceding item or items can be repeated one or more times. • Additional parameters, values, or other information can be entered.
. . .	A vertical ellipsis indicates the omission of items from a code example or command format; the items are omitted because they are not important to the topic being discussed.
( )	In command format descriptions, parentheses indicate that you must enclose the options in parentheses if you choose more than one.
[ ]	In command format descriptions, brackets indicate optional choices. You can choose one or more items or no items. Do not type the brackets on the command line. However, you must include the brackets in the syntax for OpenVMS directory specifications and for a substring specification in an assignment statement.
[l]	In command format descriptions, vertical bars separate choices within brackets or braces. Within brackets, the choices are options; within braces, at least one choice is required. Do not type the vertical bars on the command line.

Convention	Meaning
{ }	In command format descriptions, braces indicate required choices; you must choose at least one of the items listed. Do not type the braces on the command line.
bold text	This typeface represents the introduction of a new term. It also represents the name of an argument, an attribute, or a reason.
italic text	Italic text indicates important information, complete titles of manuals, or variables. Variables include information that varies in system output (Internal error *number*), in command lines (*/PRODUCER= name*), and in command parameters in text (where *dd* represents the predefined code for the device type).
UPPERCASE TEXT	Uppercase text indicates a command, the name of a routine, the name of a file, or the abbreviation for a system privilege.
`Monospace type`	Monospace type indicates code examples and interactive screen displays. In the C programming language, monospace type in text identifies the following elements: keywords, the names of independently compiled external functions and files, syntax summaries, and references to variables or identifiers introduced in an example.
-	A hyphen at the end of a command format description, command line, or code line indicates that the command or statement continues on the following line.
numbers	All numbers in text are assumed to be decimal unless otherwise noted. Nondecimal radixes—binary, octal, or hexadecimal—are explicitly indicated.

Chapter 1. PCA Overview

This chapter provides information on the following topics:

* What PCA does

* The two phases of using PCA: collecting and analyzing information

* How to obtain online help while you are running PCA

* How to adopt a general methodology for using PCA effectively to locate performance bottlenecks in your applications and determine their causes

1.1. What PCA Does

PCA collects performance and coverage information on your application and lets you analyze that information to learn about the run-time characteristics of your application, such as:

* Where your application spends the most time

* What parts of your application are executed the most

* What areas of your application are executed (or not) by tests

* What parts of your application are compute-bound or I/O-bound

* What the dynamic call relationships are among the routines in your application

* Where page faults occur

* Where your application uses system services

* Where your application executes vector instructions

This information helps you pinpoint important qualities of your application, such as the location and nature of performance bottlenecks, or how effective your test coverage is.

1.2. The Two Phases of Using PCA— Collecting and Analyzing Information

When you use PCA, you perform two basic tasks:

* Collect performance or coverage information on your running application. This is referred to as the **collection phase**.

* Analyze and interpret the collected information. This is referred to as the **analysis phase**.

PCA provides two components that correspond to these tasks: the **Collector**, which collects performance and coverage information about your application while the application is running; and the **Analyzer**, which you run as a standalone utility to examine and interpret the information that the Collector has gathered.

Figure 1.1 shows the basic cycle of PCA usage: collecting information on your application, then analyzing the results.

Figure 1.1. PCA Run-Time Usage
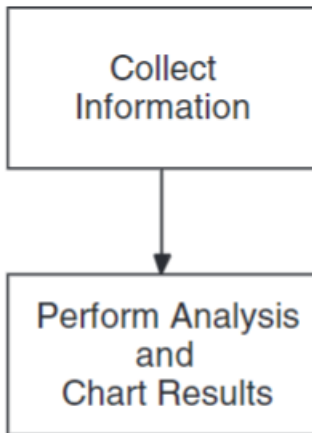
1.2.1. The Collector

The Collector gathers performance and coverage information about your running application and stores that information in a data file.

When you run the Collector, PCA displays the Collector main window on the screen, as shown in Figure 1.2. You can then do the following:

- Use the Collector main window to choose the type of information you want to collect.

- Start the collection, and continue the normal execution of your application.

Chapter 2 gives complete details on using the Collector.

Figure 1.2. Collector Main Window
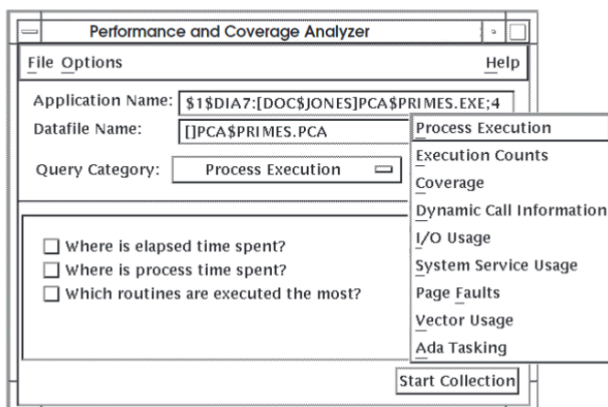
1.2.2. The Analyzer

After your application completes its execution cycle, you can use the Analyzer to view and interpret the performance or coverage information that has been gathered. The Analyzer main window is shown in Figure 1.3.

Figure 1.3. Analyzer Main Window
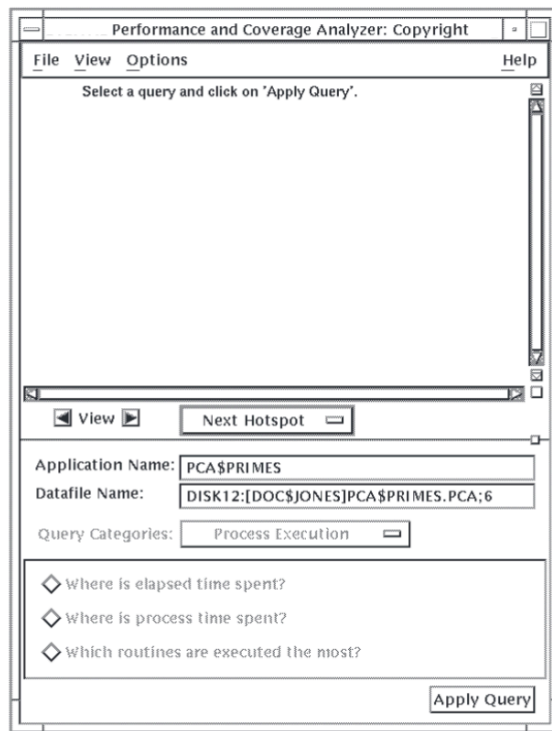
The Analyzer displays performance and coverage information graphically in types of **charts**:

- Histograms or tables

- Annotated source listings

The Analyzer also provides several navigational functions so you can easily traverse your application's structure to locate **hotspots**, places in your application that have generated the most collection activity.

Chapter 3 explains how to invoke and use the Analyzer.

1.3. Using the Context-Sensitive Online Help

To get help with PCA commands, or any of the qualifiers or parameters used with these commands, type HELP, followed by the command or topic.

For the DECwindows interface, to get help on any PCA screen object, do the following:

1. Position the pointer on the desired object.

2. Press and hold the Help key while you press MB1.

3. Release both keys.

PCA also provides another way to get context-sensitive help. Do the following:

1. Choose On Context in the Help menu.

2. Click on any object in the Analyzer window for which you want help.

When you invoke context-sensitive help, PCA displays the Help window shown in Figure 1.4.

Figure 1.4. PCA Help Window
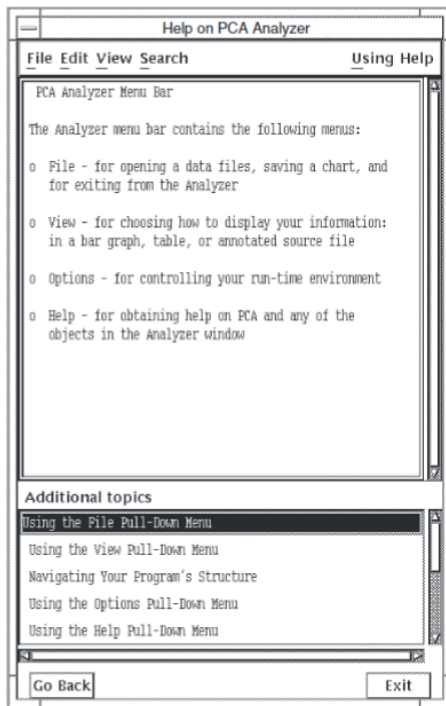
There are two panes in the Help window: a large pane on top, and a small pane below, labeled Additional topics. The large pane describes the screen object on which you clicked. The small pane gives additional topics that are related to the main help topic. You can see the help associated with an additional topic by double clicking on the topic.

1.4. Adopting a PCA Usage Methodology

Although operating PCA is easy, using PCA efficiently and in a way that lets you methodically identify and correct your performance problems involves planning. This guide does not intend to prescribe the best way to use PCA, but hopefully you will find some of the suggestions useful in developing your own usage methodology. You might also find it appropriate to modify some of the suggestions in this guide for your specific applications.

The examples in this guide are based on the general-usage methodology shown in Figure 1.5.

Figure 1.5. PCA Usage Methodology

A general PCA usage methodology can involve the following steps:

1. Set up a testing environment.

    Your first task is to create separate directory areas for each of the following types of files:

    -   Test programs and PCA data files

    -   Copies of source code files

    -   Locally built executable files

2. Create tests.

   Before you begin any performance or coverage testing, plan your tests. When your Collector-linked application starts its execution cycle, you should have tests for exercising all the major functions of your application. In addition, you should have tests that individually exercise different functions in your application, enabling you to focus on specific locations in your application. PCA provides the coverage query category for collecting and examining information that tells you how much of your application's code is exercised by tests.

   Allow for a variation in the size of your tests. For example, if you are testing a database application, your initial test might perform 10 define record and 10 delete record operations. Your second test might perform 100 of the same operations, and a third, 1000 such operations. PCA lets you analyze the changes in the database application's behavior under the differing test conditions, giving further clues to the causes of performance bottlenecks.

   VSI Test Manager for OpenVMS is useful for managing and executing PCA tests. See *Using Using VSI DECset for OpenVMS Systems* for information on using PCA with VSI Test Manager for OpenVMS.

3. Collect application execution information.

   Your first collection runs should involve gathering process execution information, such as the following:

   - Where elapsed time is spent

   - Where process time is spent

   - Which routines are called the most

   This type of information typically provides a broad, high-level view of the general locations of performance bottlenecks. By comparing elapsed time with process time information, you can see where your application is spending idle time. By finding which routines are called the most, you can find which routines require the most performance testing.

4. Chart and analyze the results of your tests.

   After each collection run, invoke the Analyzer and chart the result of the collected information. Take note of those locations that generated a lot of collection activity. You might want to go back to Step 2 and create tests that focus on those hotspots to gain more information.

5. Repeat the collection and analysis steps with more detailed tests.

   After your initial performance and coverage results indicate the general locations of your performance bottlenecks, create new tests that focus specifically on those bottlenecks. For example, if you have a routine that appears to be consuming a lot of elapsed time but not much process time, you might find that the routine is I/O-bound and is frequently in a device wait loop. You can create tests that focus on the I/O activity in that routine that will show specifically which I/O instructions are creating problems.

6. Correct the performance problems.

   After you have determined what the performance bottlenecks are, correct your code accordingly.

7. Verify performance improvements.

After you have fixed your code, verify that it works by repeating the collection and analysis cycle. Make sure your fixes do not introduce new problems.

1.5. Customizing Your Video Display

The PCA DECwindows interface enables you to change the default values for window sizes and font sizes, to provide more control of multiple PCA display windows.

The PCA default values for text fonts, window sizes, and other window resources are contained in the following files, found in the directory DECW$SYSTEM_DEFAULTS:

- PCAC.DAT – PCA Collector resource file

- PCAA.DAT – PCA Analyzer resource file

Create a local copy of these files to the directory DECW$USER_DEFAULTS,then read the value descriptions in the resource files and modify the values to your preferences.

Chapter 2. Running the Collector

To collect performance and coverage information on your application,do the following:

1. Invoke the Collector.

2. Specify the performance or coverage information you want on your application.

3. Start the data collection run.

4. Exit the Collector.

This chapter explains how to do these steps and lists all thetypes of information you can collect on your application.

2.1. Invoking the Collector

1. To invoke the Collector, do the following:

2. You can invoke the Collector on an image linked for debugging by defining the LIB$DEBUG logical to be SYS$LIBRARY:PCA$COLLECTOR.EXE. This causes the OpenVMS image activator to invoke the Collector as a debugger. You can then enter Collector commands. First link the user program, as follows:

    ```
    $ LINK/DEBUG PCA$PRIMES.OBJ
    ```

3. *Define the LIB$DEBUG logical.*

    Type the following commands to define the LIB$DEBUG logical:

    ```
    $ LINK/DEBUG PCA$PRIMES.OBJ
    $ DEFINE LIB$DEBUG PCA$COLLECTOR.EXE
    ```

    To return to the OpenVMS Debugger, deassign the logical name LIB$DEBUG, as follows:

    ```
    $ DEASSIGN LIB$DEBUG
    ```

4. *Run the program.*

    To initiate performance or coverage data collection, run the executable image of your application, as follows:

    ```
    $ RUN PCA$PRIMES.EXE
            PCA Collector Version 4.6
    PCAC>
    ```

    When the Collector window is displayed on the screen, you have successfully invoked the Collector.

2.2. Specifying Performance or Coverage Information to Collect

This section explains how to specify the types of performance or coverage information you want to collect on your application. The following steps are explained:

• Opening a data file to store the performance and coverage information gathered by the Collector

- Specifying the performance or coverage information you want to collect by choosing a query

- Customizing the collection run to suit your specific requirements

2.2.1. Opening a Data File

Using the DECwindows interface, after the Collector main window is displayed on your screen, by default the Collector automatically opens a data file. If a data file with the same name as your application already exists, the Collector opens that data file and appends new collection information to it. If no such data file already exists, the Collector creates one, and assigns it the name of the application with a file type of `.PCA`.

You can also open a data file in one of the following ways:

- You can choose New Datafile... from the File menu to open a new data file. You typically do this if:

  - You want to give the data file a name other than the default.

  - You want to overwrite an existing data file and not append new collection information to it.

- You can choose Open Datafile... from the File menu to open an existing data file. You typically do this if this is not the first collection run on your collector-linked application, and you want to keep all the performance and coverage information for a particular application in one data file.

Although it is not required in most cases, you should keep all the performance and coverage information for an application in one data file.

Figure 2.1 shows the File menu and the New Datafile dialog box.

Figure 2.1. Opening a Data File
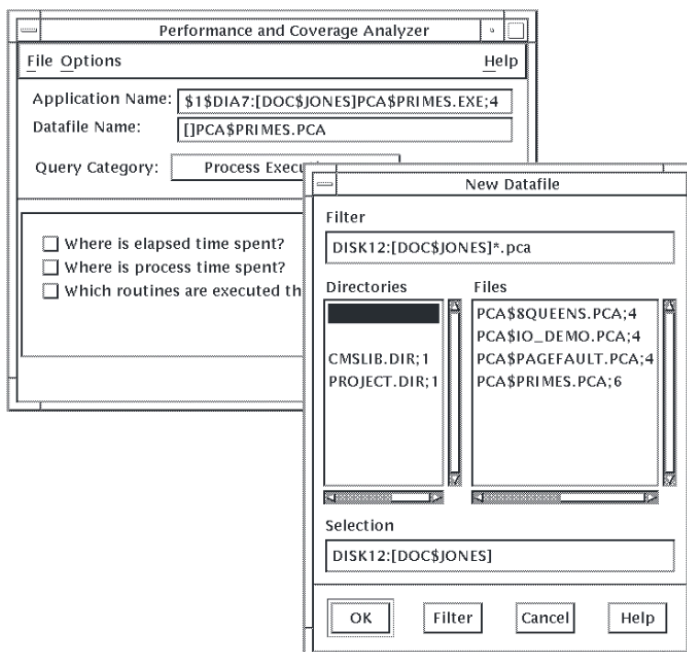
2.2.2. Choosing a Query

This section describes how to choose a **query** using the DECwindows interface. A query is a specific question appearing in the Collector main window that corresponds to a specific type of information to

collect. For example, the query that corresponds to elapsed time information is "Where is elapsed time spent?" A **query category** is a grouping of associated queries. For example, the Process Execution query category contains three queries that pertain to the basic run-time characteristics of your application: where the most elapsed and process time is spent, and what routines are executed the most.

Use the following steps to specify what kind of performance or coverage information you want to collect:

1. Choose a query category from the Query Category menu.

    Click on the box to the right of the Query Category label. A menu of available query categories is displayed. When you choose a category, the Collector displays the associated queries in the query pane at the bottom of the window.

    Figure 2.2 shows the Collector main window, Query Category menu, and query pane. By default, the Collector displays the Process Execution queries because this category is typically the best place to begin analyzing your application's performance.

2. Choose a specific query in the query pane.

Figure 2.2. Collector Query Pane
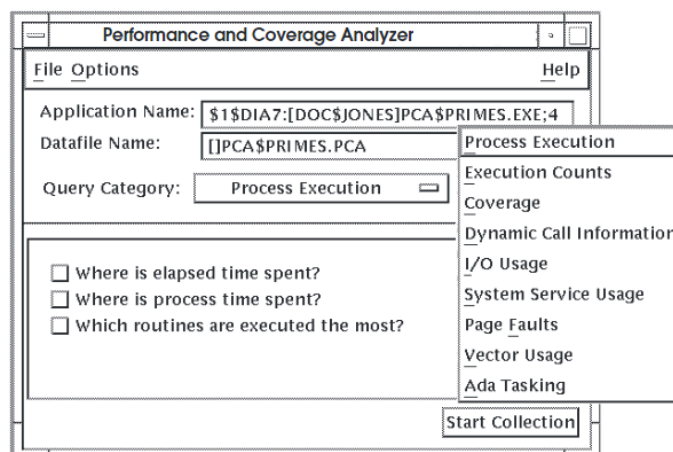
2.2.3. Customizing the Collection Run

The Collector lets you make the following customizations to your collections:

*   The sampling interval for elapsed time or process time

*   The collection elements for information such as execution counts and coverage information

Use the Options menu to get access to the customization functions, as shown in Figure 2.3. This pull-down menu's Vector Range option is available only on OpenVMS VAX systems.

Figure 2.3. Collector Options Menu
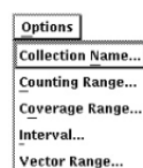
2.2.3.1. Customizing the Sampling Interval for Elapsed or Process Time

When the Collector gathers information on where your application is spending the most time, it uses an algorithm that involves noting the currently executing program address at regular intervals of time. By default, the interval is 10 milliseconds.

To customize the sampling interval for elapsed or process time, do the following:

1. Choose the Interval... menu item from the Customize menu.

2. In the Set Sampling Rate dialog box, click on the menu next to either the Elapsed Time or Process Time sampling rate labels.

3. Choose a sampling rate in the displayed menu.

Figure 2.4 shows the Set Sampling Rate dialog box.

Figure 2.4. Set Sampling Rate Dialog Box
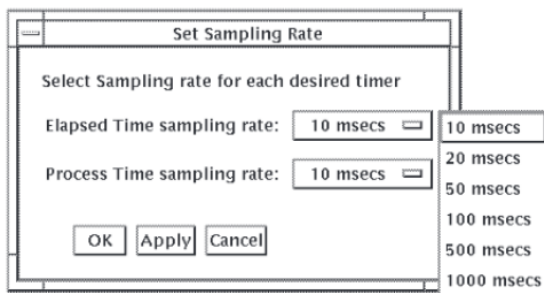
2.2.3.2. Specifying Collection Elements

You can focus a collection on specific modules, specific routines, or all lines of your application when you are collecting information such as the following:

• Counting information

• Coverage information

Focusing collection activity on specific program elements in your application is useful once you have narrowed down the locations of your performance bottlenecks, or if you have a particularly large application.

Figure 2.5 shows the Specific Elements list box, which is displayed when you choose Specific Elements... in the Counting Range, Coverage Range, or Vector Range dialog boxes. To specify a module or routine in the Specific Elements list box, click on the module or routine name, then click on OK.

Figure 2.5. Specific Elements List Box
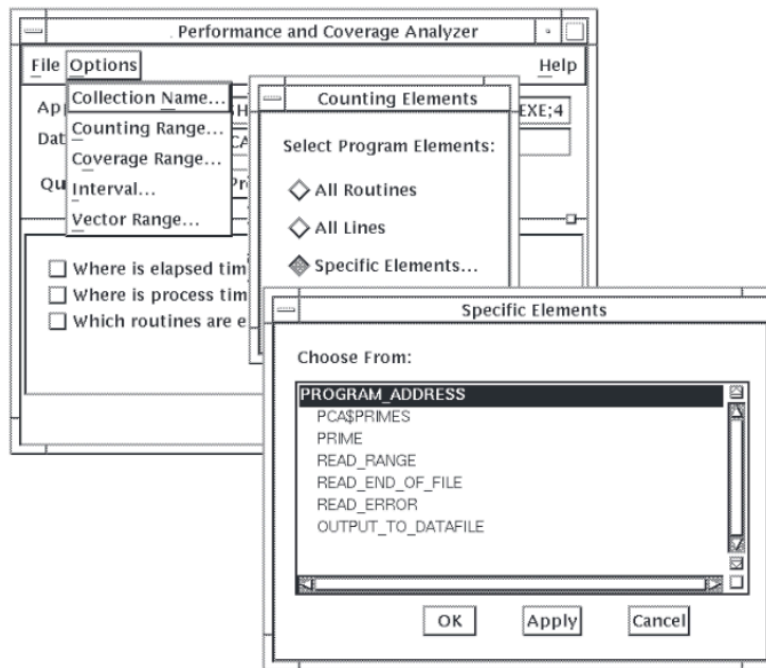
2.3. Starting the Collection Run

After you invoke the Collector and optionally set the data file and collection type, enter the GO command to start data collection. To run your program without collecting data, use the /NOCOLLECT qualifier with the GO command.

To start the collection using the DECwindows interface, click on Start Collection in the Collector main window.

The Collector main window disappears and the application begins execution. As the execution proceeds, the Collector gathers information. When the application completes its normal execution cycle, the collection is also completed.

2.4. Exiting the Collector

To end the Collector session before you enter the GO command, enter the EXIT command or press Ctrl/ Z. If the data collection has already started, you cannot enter an EXIT command; you must press Ctrl/Y to stop the collection run.

Caution

If you press Ctrl/Y, do not use the DCL STOP command immediately afterwards. If Ctrl/Y interrupts the Collector when the Collector is executing supervisor-mode code, a subsequent STOP command might cause a supervisor-mode exception that kills your entire process and logs you out. To avoid this, execute another program after pressing Ctrl/Y, or type EXIT. The Collector's exit handlers then successfully close out the data collection.

To stop Collector output to your terminal, such as SHOW command output, press Ctrl/C. The Collector aborts the command and returns to DCL level.

Repeat the RUN command to invoke the Collector again and collect additional data on your program's performance or coverage. If you want the same file to contain data collections from many executions of the same program, use the /APPEND qualifier with the SET DATAFILE command, as follows:

```
PCAC>  SET DATAFILE/APPEND PCA$PRIMES.PCA
```

You can run your program without collecting data by issuing the GO command with the /NOCOLLECT qualifier.

The Collector allows you to collect different kinds of data in the same collection run, but you should be cautious in doing so. Values can be distorted if different kinds of data are collected in the same collection run. The Collector provides informational messages warning about potential conflicts.

2.5. Query Categories

This section lists the query categories available with PCA and describes each individually. The query categories are as follows:

- Process Execution

- Execution Counts

- Coverage

The query category Vector Usage applies to OpenVMS VAX systems only. All other categories apply to both OpenVMS VAX and OpenVMS Alpha systems.

2.5.1. Process Execution Query Category

The Process Execution query category includes queries to show where your application spends the most elapsed and process (or CPU) time, and which routines are executed the most. The Process Execution queries are as follows:

- *Where is elapsed time spent?*

- *Where is process time spent?*

- *Which routines are executed the most?*

OpenVMS Alpha systems offer an additional "Where are unaligned data exceptions?" query. By comparing where your application spends the most elapsed and process time, you can determine where your application spends idle time.

2.5.2. Execution Counts Query Category

The Execution Counts query category includes queries to show the exact number of times specific program locations are executed. The Execution Counts queries are as follows:

- *Which lines are executed the most?*

- *Which routines are executed the most?*

- *Which modules are executed the most?*

Examining execution counts information helps you to identify inefficient algorithms, or excessive calls to other routines or system services in your application.

2.5.3. Coverage Query Category

The Coverage query category contains queries to show which parts of your application are executed (or not) during each collection run. The Coverage queries are as follows:

- *What percentage of the application was covered?*

- *Where is the application covered?*

- *Which routines were completely NOT covered?*

- *Where is the application acceptably not covered?*

Designing tests that execute all the code paths of your application is important in determining how your application performs under the circumstances for which the application is intended.

Collecting coverage information is useful for determining if the tests you are using during your collection runs are adequately covering all your application's code paths. In addition, collecting coverage information can tell you how to change your tests to give acceptable code path coverage if the tests initially do not adequately exercise all code paths.

2.5.4. Dynamic Call Information Query Category

The Dynamic Call Information query category contains queries to show the calling relationships among routines in your application. The Dynamic Call Information queries are as follows:

- *What is the call tree structure?*

- *Which routines call which other routines?*

- *Which routines make the most calls?*

By determining the basic, run-time calling relationships among the routines in your application, especially determining which routines are executed most, you can determine if your application is calling routines efficiently.

2.5.5. I/O Usage Query Category

The I/O Usage query category contains queries to count information about all the I/O your program performs. The I/O Usage queries are as follows:

- *Where is I/O being done?*

- *What I/O system services are being used?*

- *Where are I/O reads being used?*

- *What is the number of I/O read counts?*

- *Where are I/O writes being used?*

- *What is the number of I/O write counts?*

- *Where are I/O reads and writes used?*

- *What files is I/O done to?*

- *What file virtual block numbers are used?*

- *What file keys are used?*

- *What are the record sizes being used?*

By comparing I/O usage information with process execution and executions counts information, you can determine if your application is using I/O services excessively or inefficiently.

2.5.6. System Service Usage Query Category

The System Service Usage query category contains queries to show which system services your application calls, how often it calls them, and which program locations do the calling. The System Service Usage queries are as follows:

- *Where are system services used?*

- *What system services are used?*

By comparing system service usage information with process execution and executions counts information, you can determine if your application is using system services excessively or inefficiently.

2.5.7. Page Faults Query Category

The Page Faults query category contains queries to show where a page fault occurs in your application and which program address caused it. The Page Faults queries are as follows:

- *Where are the page faults occurring?*

- *What are the addresses causing page faults?*

Page fault information is useful in determining how your application is using system resources in general. For example, page fault information can tell you why your application is consuming a lot of time in system space.

2.5.8. Vector Usage Query Category

The Vector Usage query category contains queries to show information on vector processor usage in your application. The Vector Usage queries are as follows:

- *Where is vector processing time spent?*

- *Where are vector instructions executed?*

- *What vector instructions are used?*

- *What are the vector lengths used?*

- *What is the number of elements set in the vector masks?*

- *What is the distribution of vector register usage?*

- *What vector strides are used?*

- *Where are the largest number of vector operations used?*

If you have an application running on a computer with a vector processor, you can use the information from the Vector Usage category to determine how efficiently your application is using the vector processor. In addition, you can determine how you have split the processing of your application between the scalar and vector processors in your computer.

2.5.9. Ada Tasking Query Category

The Ada Tasking query category contains queries related to showing information on all context switches in Ada multitasking applications. The Ada Tasking queries are as follows:

- *What tasks are performing context switching?*

- *What are the Ada tasking names used?*

- *What are the Ada tasking types used?*

- *What are the Ada task priorities used?*

Chapter 3. Running the Analyzer

After you collect information, you must analyze it; that is, choose the same queries as in the collection phase and interpret the results.

This chapter explains the following steps in analyzing the collected information:

1. Invoke the Analyzer.

2. Open the performance data file (if you did not specify one in the command line to invoke the Analyzer)

3. Choose a query to specify the information to be displayed.

4. Navigate through the Analyzer display (change the view in the display, traverse a chart, or zoom in on the display for more detailed information).

5. Exit the Analyzer.

3.1. Invoking the Analyzer

To invoke the Analyzer, type the following command at the system prompt:

```
$PCA [DATAFILE]
```

In the command line, the optional argument datafile represents the name of the performance data file you want to analyze.

After you invoke the Analyzer, PCA displays the Analyzer main window.

Note

If a valid DECwindows display is defined, PCA defaults to the DECwindows environment. However, the character-cell environment is also available. Invoke one, or switch between the two, with the following commands:

$ **PCA /INTERFACE=DECWINDOWS [DATAFILE]**

$ **PCA /INTERFACE=CHARACTER_CELL [DATAFILE]**

3.2. Opening the Performance Data File

After you have invoked the Analyzer, open the data file created during the collection phase using one of the following options:

• Specify the name of the data file in the Analyzer invocation command, shown in Section 3.1

• Choose Open Datafile... from the File menu in the Analyzer main window.

If you choose the latter option, PCA displays the Select PCA Datafile dialog box shown in Figure 3.1.

To select a performance data file in the Select PCA Datafile dialog box, do the following:

1. Click on the data file you want to open in the Files list box, or type the name of the data file directly into the Selection box.

2. Click on OK.

The Select Datafile dialog box is a standard DECwindows file selection box. To get help on using the file selection box, choose the On Context option under the Help menu. With your cursor in the form of a question mark, click on a button or field label to view the online help explanation for the selected item.

Figure 3.1. Selecting a Data File
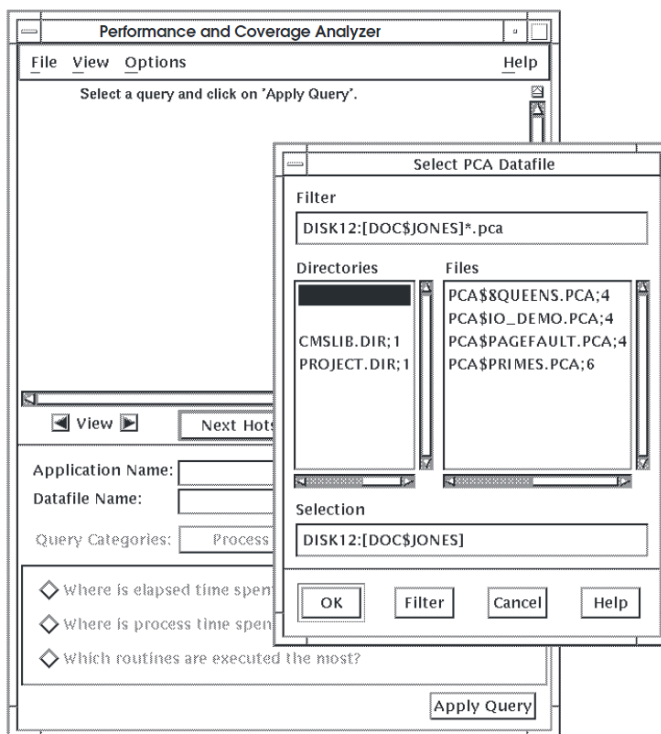
3.3. Specifying Information to Be Displayed

After you open a data file, you display the information contained in it by choosing a query. Figure 3.2 shows the Analyzer query panel using the DECwindows interface.

Figure 3.2. Analyzer Main Window Query Panel
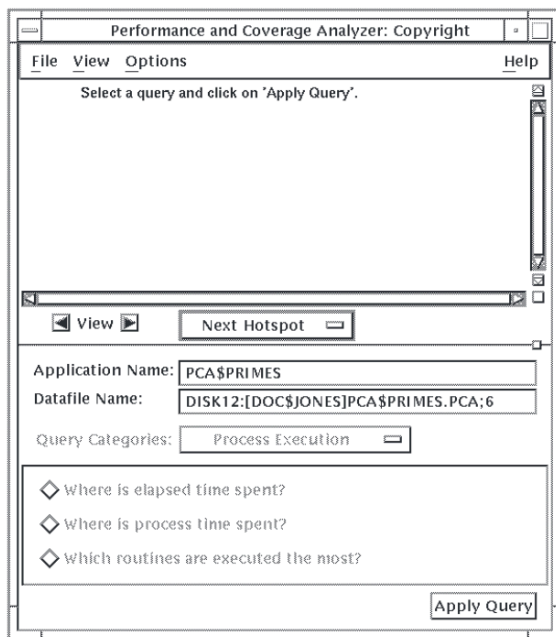
By default, the Analyzer checks the data file and sets the query category to what was collected in the data file. Queries for which no information was collected are grayed out. To change the query category, use the Query Category menu.

Once you have chosen the query for which you want to see information, click on the Apply Query button. The Analyzer displays a chart that graphically shows information in response to the query you have chosen.

The default chart displayed by the Analyzer depends on the query. For example:

- A histogram of routines for elapsed or process time

- An annotated source listing for execution counts of lines in your program

You can change charts with the View menu at the top of the Analyzer window, or with the View arrow buttons in the middle of the window.

3.4. Navigating Through the Analyzer Charts

The Analyzer facilitates navigating through the charts and locating areas of more detailed information. The basic object the Analyzer locates is called a **performance hotspot**, or "hotspot." A hotspot is a line or area of code that generated collection activity for the current query.

When you display a histogram or table in the Analyzer, by default the Analyzer sorts the entries in the display in descending order of collection activity. Next to each entry in the histogram or table, the Analyzer also displays the percentage of total collection activity generated by that entry.
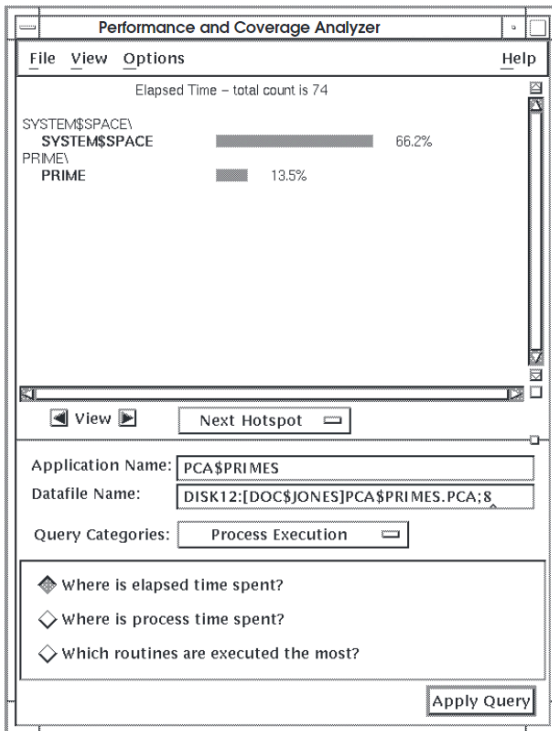
3.4.1. Changing Charts

The Analyzer provides three types of charts in which to display information:
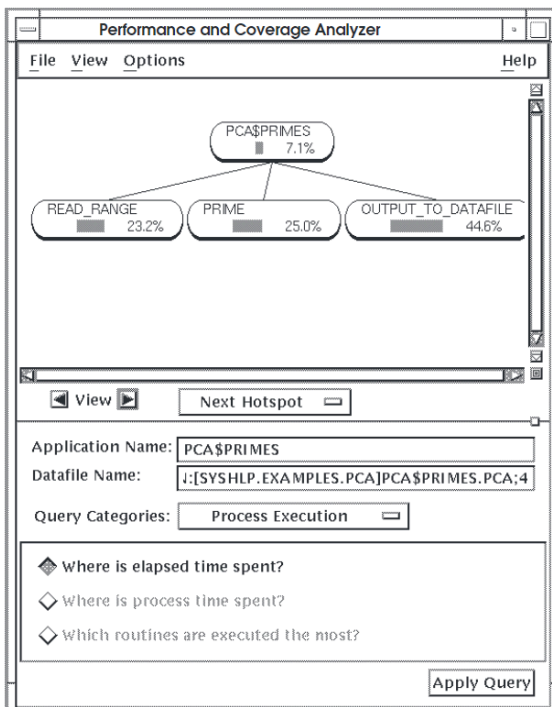
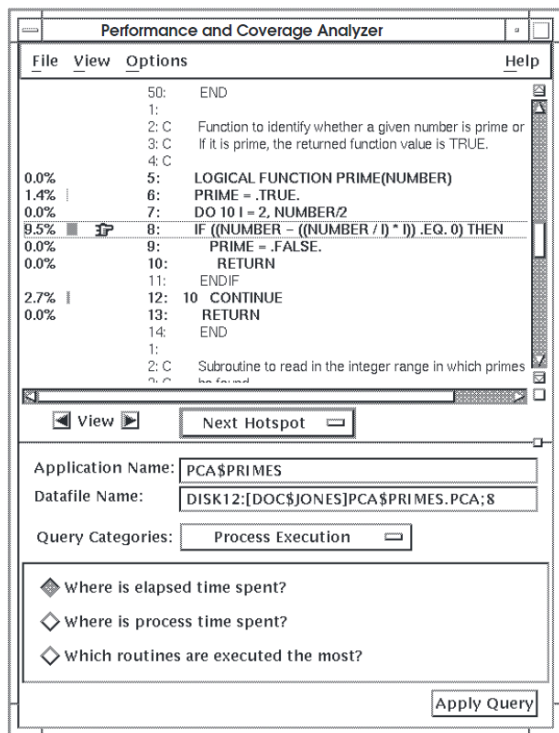- Histograms and tables

- Call trees

- Annotated source listings

Figure 3.3, Figure 3.5, and Figure 3.5 show, respectively, a sample histogram, call tree, and annotated source listing.

Figure 3.3. Sample Histogram

Figure 3.4. Sample Call Tree

Figure 3.5. Sample Annotated Source Listing

3.4.2. Traversing Charts

The Analyzer provides four **traversal** functions in the Navigate menu that let you quickly move around the Analyzer charts:

• Next—Positions the display at the next hotspot, in descending order.

• First—Positions the display at the hotspot that generated the most collection activity.

• Previous—Positions the display at the previous hotspot, in ascending order.

• Current—Positions the display at the currently selected chart entry. This function is useful if you have scrolled ahead of the current hotspot and want to return to it.

3.4.3. Adjusting the Granularity of Information in the Displays

The Analyzer provides two buttons on either side of the label titled "View" above the query panel. These buttons, a left arrow and a right arrow, let you adjust the granularity of the current display, as follows:

• Left arrow view button—Redraws the current chart to display less granularity of information. For example, if the current chart is an annotated source listing of your application, clicking on the left arrow causes the display to show only the routines in your application.

  If you select an entry in the current chart, the left arrow view button applies only to that entry.

• Right arrow view button—Redraws the current chart to display more granularity of information. For example, if the current chart is a histogram of routines, clicking on the right arrow view button causes the display to show individual lines of code.

If you select an entry in the current chart, clicking on the right arrow view button applies only to that entry.

You can increase the granularity of information only from views of modules or routines; you can decrease the granularity of information only from views of lines or routines.

3.5. Stopping Terminal Output or Exiting the Analyzer Session

To stop the Analyzer's terminal output, such as long output from a SHOW or LIST command, press Ctrl/C. The Analyzer aborts the current output operation and prompts for a new command.

To end the Analyzer session, enter the EXIT command or press Ctrl/Z.

Chapter 4. Determining an Application's Basic Run-Time Characteristics

This chapter shows how to use PCA to determine the following run-time characteristics of your application:

- Which modules, routines, or lines consume the most elapsed time and process time

- Which modules, routines, or lines consume the most idle time

- Which routines are called the most

By examining these characteristics, you can determine where your performance bottlenecks are, what your I/O service wait times are, what your page faulting time is, and so on.

To get this information, use the Process Execution query category, which contains the following queries:

- *Where is elapsed time spent?*

- *Where is process time spent?*

- *Which routines are executed the most?*

4.1. Collecting Process Execution Information

To collect process execution information on your application using the DECwindows interface:

1. Compile your application for use with the OpenVMS Debugger.

2. Link your application for use with the Collector.

3. Invoke the Collector.

4. Open a performance data file.

5. Choose the Process Execution query category from the Query Category option menu.

6. Choose each of the queries in the Process Execution category, one at a time, in individual collection runs.

7. Click on Start Collection.

For more information on performing these steps, see Chapter 2.

The rest of this chapter explains how to use the Analyzer to do the following:

- Analyze and interpret elapsed time and process time information.

- Compare elapsed time and process time information to make initial findings on where your application is spending idle time.

- Locate the routines that are called the most in your application.

4.2. Analyzing Elapsed Time Information

To analyze elapsed time information on your application using the DECwindows interface, do the following:

1. Invoke the Analyzer and specify the name of the data file containing the information you want to see. (See Chapter 3 for details on completing this step.)

2. Choose the Process Execution query category.

3. Choose the query "Where is elapsed time being spent?"

4. Click on Apply Query.

Figure 4.1 shows a histogram of elapsed time information for the application PCA$PRIMES.
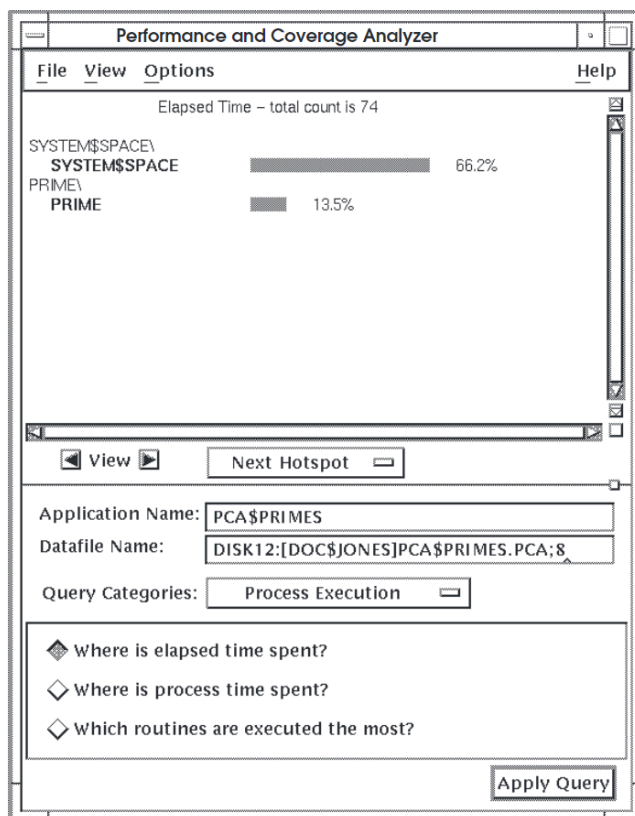
Figure 4.1. Sample 1 of Elapsed Time Histogram

Figure 4.1 shows the following elapsed time information:

- 66.2 percent of all elapsed time is spent in SYSTEM$SPACE.

- 13.5 percent of all elapsed time is spent in the subroutine PRIME.

- No elapsed time information is recorded for any of the other modules or subroutines in PCA $PRIMES, including the following:

- `OUTPUT_TO_DATAFILE`

- `READ_RANGE`

- `PCA$PRIMES`

- `SYSTEM$SERVICE`

- `READ_END_OF_FILE`

- `READ_ERROR`

- `READ_RANGE`

4.2.1. Elapsed Time Information

Elapsed time information might not be collected for some routines for several reasons:

- The routines are not called often enough.

- The test run was inadequate to cover all the code paths in the application.

- The elapsed time collection interval was inappropriate for the particular test run or the application.

- The routines are never called due to a logic error in the application.

To get more elapsed time information on routines that initially do not generate elapsed time information, do the following:

- Redesign your tests.

  You can use the Coverage query category to choose queries that give you specific information on routine and line test coverage of your application. Thus, you can see whether your tests are calling routines that report zero elapsed time.

- Create tests that run longer.

4.2.2. Elapsed Time in System Space

Some applications appear to spend a lot of time in system space, as the histogram in Figure 4.2 shows.

Figure 4.2. Sample 2 of Elapsed Time Histogram
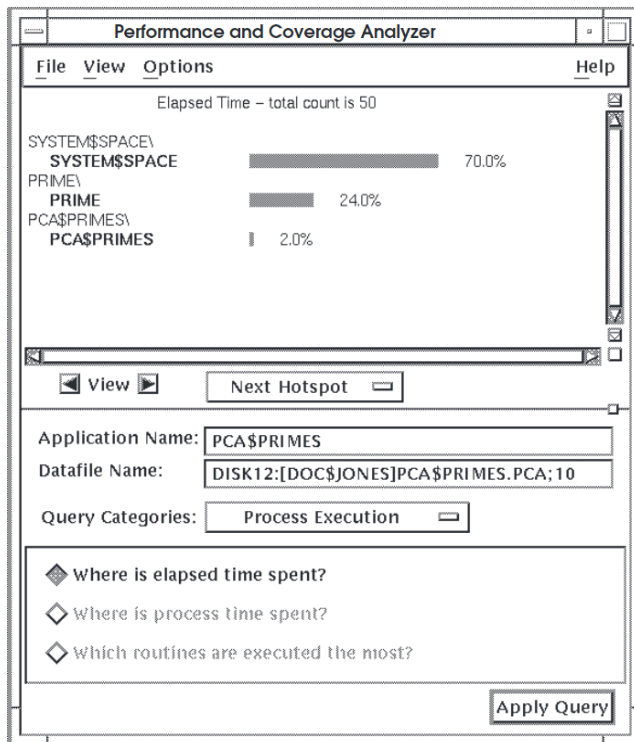
4.3. Analyzing Process Time Information

The next step is to collect process time information and compare it to the elapsed time information.

To analyze process time information on an application using the DECwindows interface, do the following:

1. Choose the Process Execution query category.

2. Choose the query "Where is process time being spent?"

3. Click on Apply Query.

Figure 4.3 shows two histograms: the one on the left is for elapsed time for PCA$PRIMES the other is for process time information on the same program.
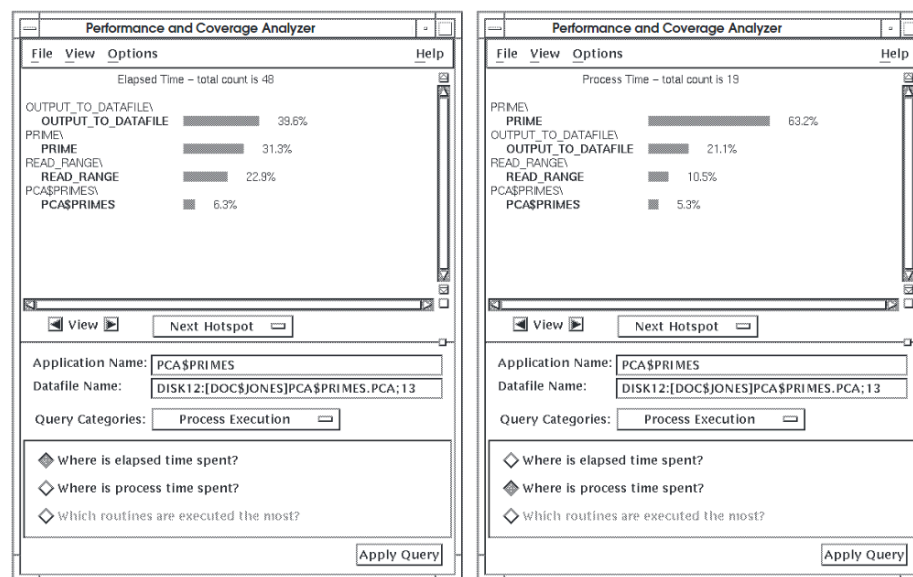
Figure 4.3. Process Time Histogram

Figure 4.3 shows the following process time information:

- 63.2 percent of all process time is spent in subroutine `PRIME`.

- 21.1 percent of all process time is spent in subroutine `OUTPUT_TO_DATAFILE`.

- 10.5 percent of all process time is spent in subroutine `READ_RANGE`.

- 5.3 percent of all process time is spent in module `PCA$PRIMES`.

Finding Idle Time in an Application

By comparing elapsed time with process time information collected for `PCA$PRIMES`, you can make the following initial conclusions:

- A significant difference exists between the amount of elapsed time and process time spent in subroutines `PRIME` and `OUTPUT_TO_DATAFILE`.

  Routine `OUTPUT_TO_DATAFILE` generates significantly more elapsed time than process time information. This indicates that there is a sizable amount of idle time in that routine due, perhaps, to an I/O wait loop.

  By comparison, routine `PRIME` consumes the majority of process time, but considerably less elapsed time. This might indicate that `PRIME` is computation-bound due to an inefficient algorithm.

- Routine `READ_RANGE` and module PCA$PRIMES are not generating a lot of process or elapsed time information. It is premature to assume there are no performance problems with these code locations, however; different test conditions might turn up significantly different results. As with routines that do not generate any collection activities, check your tests to see if they are exercising all the code paths, or taking into account all the different conditions under which the application is likely to be used.

- The other routines in `PCA$PRIMES` are not generating any collection activity for process time information.

4.4. Analyzing Routine Calling Relationships

For more clues on the basic run-time characteristics of your application, analyze the routine calling relationships. This, in conjunction with elapsed and process time information, can help you determine if a performance bottleneck is caused by excessive calls to one or two particular routines.

To analyze routine calling relationships in an application using the DECwindows interface, do the following:

1. Choose the Dynamic Call Information query category.

2. Choose the queries "Which routines are called the most?" or "Which routines make the most calls?"

3. Click on Apply.

To get complete information on the routine calling relationships in your application, get information from both queries mentioned in the previous list.

4.4.1. Analyzing Elapsed Time Information in an Annotated Source Listing

When you see elapsed time information displayed in a histogram, you can get abroad measure of what routines are consuming the most elapsed time. You can increase the granularity of the elapsed time information in the Analyzer by displaying an annotated source listing.

Although the current chart can be a histogram, you can view elapsed time information in an annotated source listing. Use the DECwindows interface by clicking on the right arrow view button.

Chapter 5. Finding an Application's Most Executed Parts

This chapter shows how to use PCA to collect and analyze execution count information. Collecting execution count information tells you how many times each line in a module or routine is executed. This information can help you find the exact location and possible causes of performance bottlenecks.

The topics covered in this chapter include the following:

- Collecting execution counts information

- Analyzing and interpreting execution counts information

To collect execution counts information, use the Execution Counts query category, which contains the following queries:

- *Which lines are executed the most?*

- *Which routines are executed the most?*

- *Which modules are executed the most?*

5.1. Collecting Execution Counts Information

To collect execution counts information on specific modules or routines in your application using the DECwindows interface, do the following:

1. Compile your application for use with the OpenVMS Debugger.

2. Link your application for use with the Collector.

3. Define the Collector as your process debugger.

4. Invoke the Collector.

5. Open a performance data file.

6. Choose the Execution Counts query category from the Query Category option menu.

7. Choose Set Counting Range... from the Customize menu, as shown in Figure 5.1.

**Figure 5.1. Selecting Execution Counts for a Specific Routine**
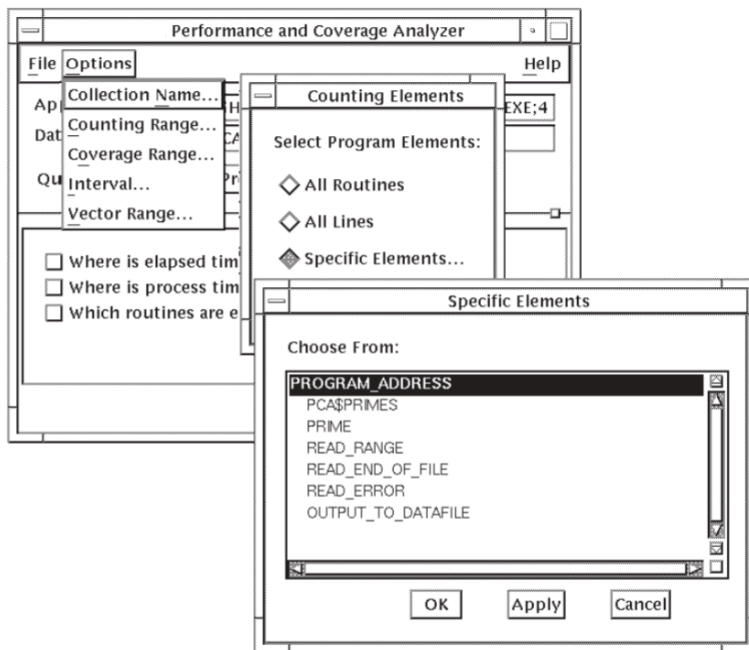
8. Click on Specific Elements... in the Counting Elements dialog box.

9. In the Specific Elements dialog box, select the routine for which you want to collect execution counts information. Double click on entries in this dialog box to expand them to individual routines.

10. Click on OK in the Specific Elements dialog box.

11. Click on OK in the Counting Elements dialog box.

12. Click on "Which lines are executed the most?" in the query panel.

13. Click on Start Collection.

5.2. Analyzing Execution Counts Information

To analyze execution counts information on your application using the DECwindows interface, do the following:

1. Invoke the Analyzer and specify the name of the data file containing the information you want to look at. (Chapter 3 describes this step.)

2. Choose the Execution Counts query category.

3. Choose the query "Which routines are executed the most?" in the Execution Counts query panel.

4. Click on Apply Query.

Figure 5.2 shows a histogram of execution count information for each routine in the application PCA $PRIMES.

Figure 5.2. Execution Count Information By Routine
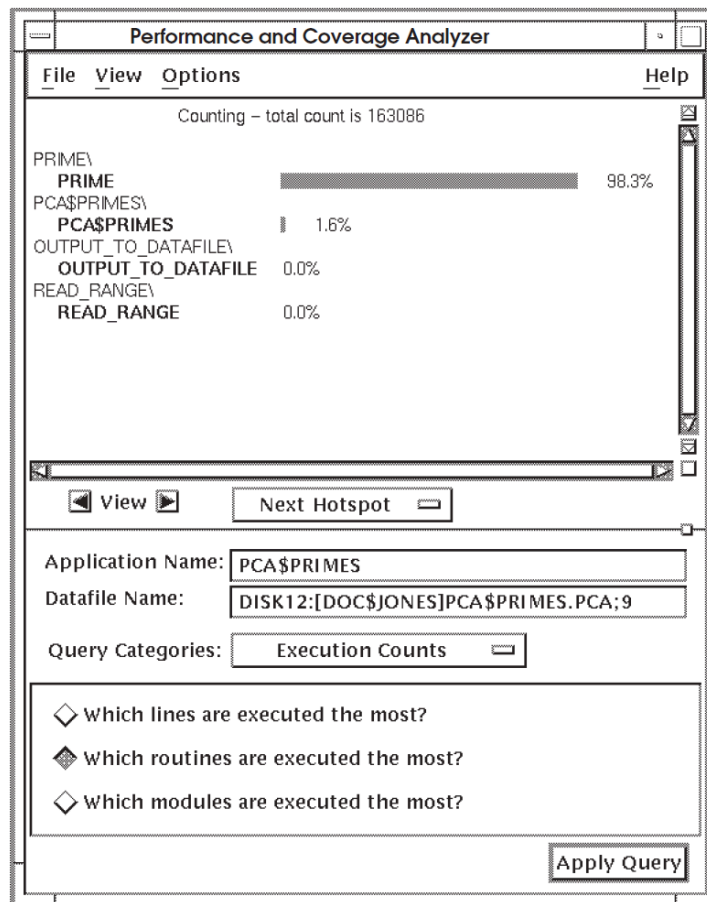
To see counting information by line, click on the right arrow view button. The Analyzer displays an annotated source listing that gives detailed line counting information, as shown in Figure 5.3.
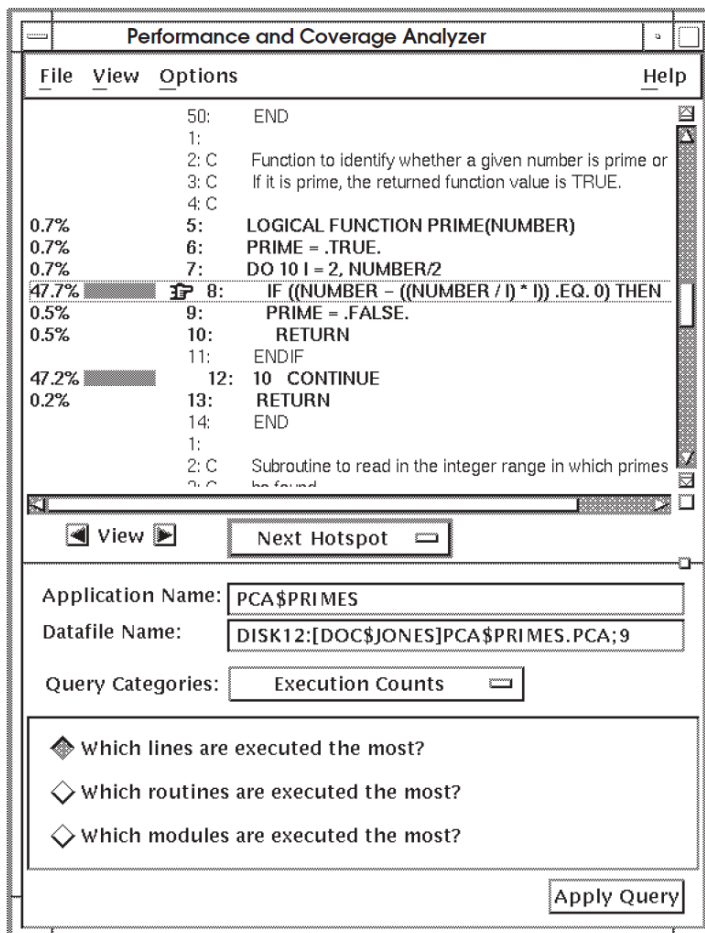
Figure 5.3. Line Count Information

Figure 5.3 shows that lines 8 and 12 account for almost 95 percent of all the executed lines in routine PRIME. When a high percentage of line count information is concentrated in one loop, you should evaluate whether you can rewrite the loop to be more efficient.

Look at the following information when analyzing line count information:

- Examine loops.

  If a loop is generating a high amount of counts, examine the code structure of the loop carefully. For example, look for redundant variable assignments or inappropriate algorithms, such as inefficient sort or search routines.

- Compare elapsed time information with counting information.

  If a code location is consuming a lot of time but not generating a lot of line count activity, search for the reason. For example, your program could be spending a lot of time waiting for I/O services. For more information on finding where your application uses system services,see Chapter 7.

- Examine the relationship between the number of times the routine is called and the line count information for the lines in the routine.

  For example, if a routine is called 2,000 times and each line in the routine is executed 2,000 times (that is, each line is executed once each time the routine is called), the performance characteristics of that routine are probably good. However, you should look at why the routine is called 2,000 times and see whether that number is appropriate.

By contrast, if a routine is called only once but each line in the routine is executed 2,000 times, you should investigate why. The routine itself might have inefficient algorithms in it, depending on what the routine's function is.

Finally, if you have a routine that is called 500 times and each line in the routine is called 2500 times, you should investigate the routine and where the routine is being called from.

Chapter 6. Determining Test Coverage

This chapter explains how to use PCA to collect and analyze the following test coverage information on your application:

- Where your application is covered by tests

- Where your application is not covered by tests

- Where your application is acceptably not covered by tests

With this information, you can determine whether all the code paths in your application are being executed satisfactorily.

To collect and analyze test coverage information, use the Coverage query category, which contains the following queries:

- *What percentage of the application was covered?*

- *Where is the application covered?*

- *Which routines were completely NOT covered?*

- *Where is the application acceptably not covered?*

6.1. Collecting Test Coverage Information

To collect information on how your application is covered by tests using the DECwindows interface, do the following:

1. Compile your application for use with the OpenVMS Debugger.

2. Link your application for use with the Collector.

3. Define the Collector as your process debugger.

4. Invoke the Collector.

5. Open a performance data file.

6. Choose the Coverage query category from the Query Category option menu.

7. Choose a query in the Coverage query panel.

8. If you have a particularly large application and you want test coverage information on a particular module or routine, use the Customize menu and choose the Set Coverage Range... menu item.

9. Click on Start Collection.

6.2. Analyzing Test Coverage Information

To analyze test coverage information on your application using the DECwindows interface, do the following:

1. Invoke the Analyzer and specify the name of the data file containing the information you want to see. (Chapter 3 describes this step.)

2. Choose the Coverage query category.

3. Choose one of the queries in the coverage query panel, as described in the following sections.

6.2.1. Showing Where Your Application Is Covered By Tests

To show where your application is covered by tests, do the following:

• Choose the query "Where is the application covered?"
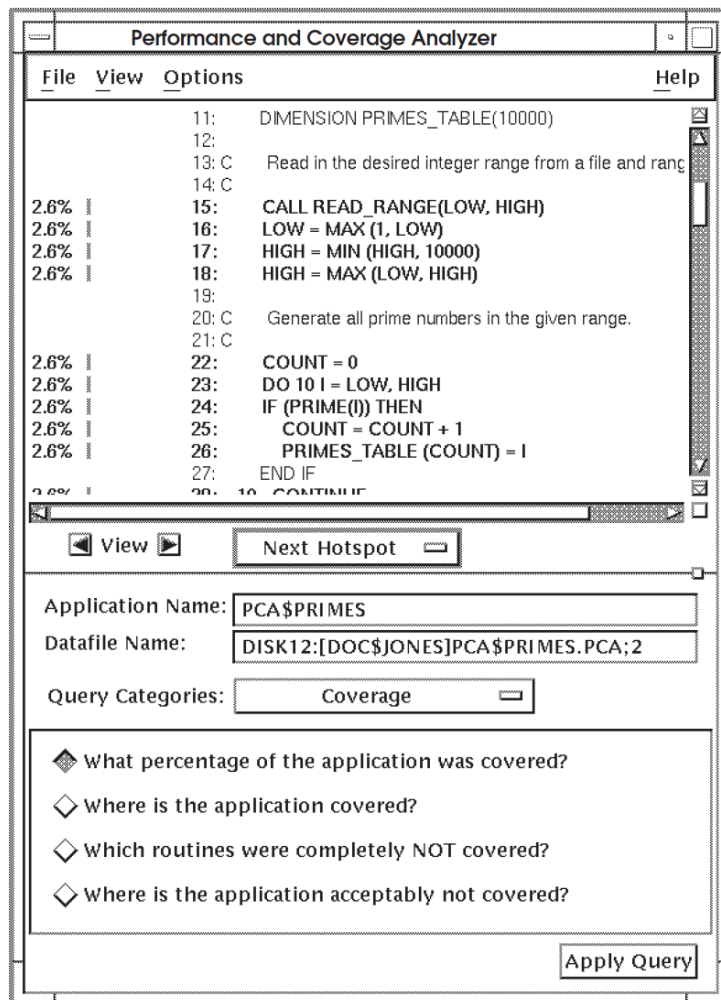
• Click on Apply Query.

The Analyzer displays a histogram showing the total test coverage among all the routines in your application. Figure 6.1 is an example test coverage histogram of the application PCA$PRIMES.

Figure 6.1. Test Coverage Histogram
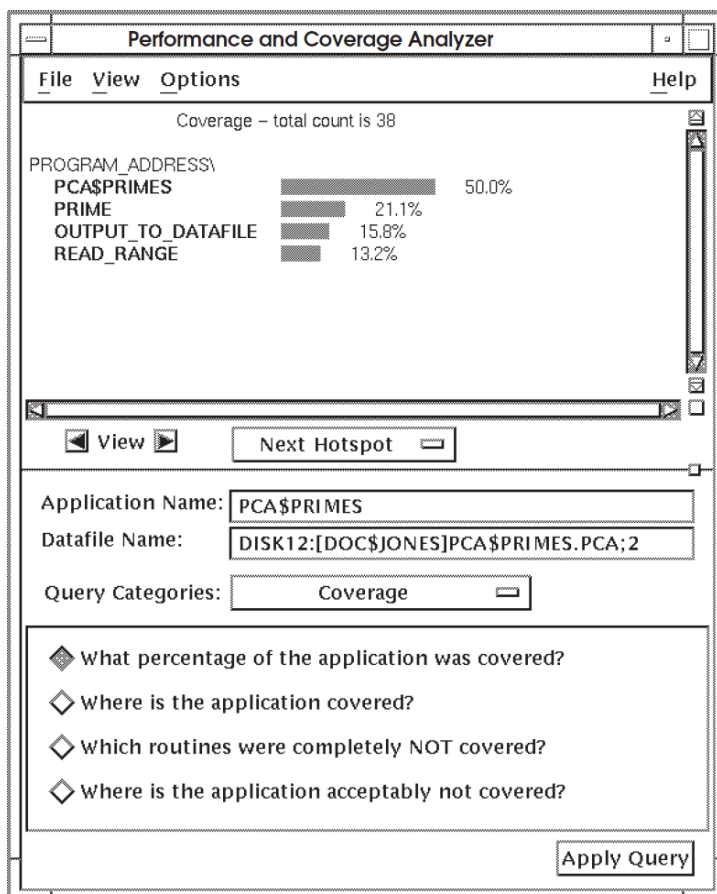
In Figure 6.1, the bars and percentage figures for each line represent the percentage of coverage points for that line, with respect to the total number of lines covered.

For a breakdown of test coverage by line, click on the right arrow view button. Figure 6.2 is an example of an annotated source listing showing which lines were covered by tests.

**Figure 6.2. Test Coverage Information By Line**

In Figure 6.2, the Analyzer displays a nonzero percentage figure to the left of lines that were covered, and a zero percentage figure next to the lines that were not covered.

6.2.2. Showing Where an Application Is Not Covered By Tests

To show where your application is not covered by tests, do the following:

- Choose the query "Which routines were completely NOT covered?"

- Click on Apply Query.

The Analyzer displays a histogram showing only those routines that received no test coverage at all. Figure 6.3 is an example histogram that shows routines that were completely not covered by tests in the application PCA$PRIMES.

Figure 6.3. Test Noncoverage
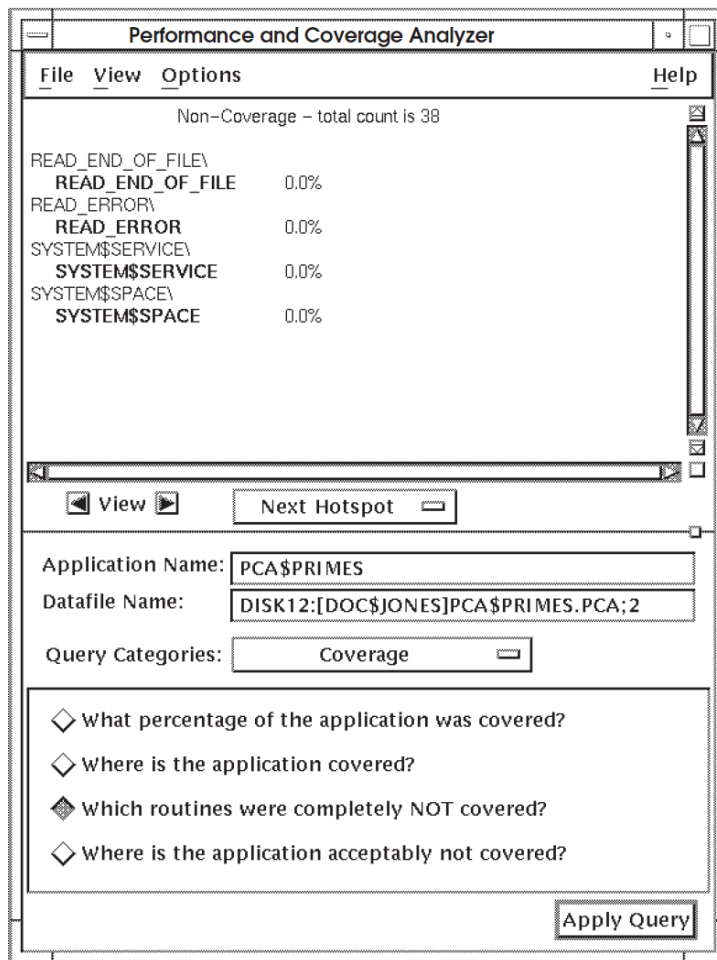
For a breakdown of test noncoverage by line in a particular routine, click onthat routine in the display pane and click on the right arrow View button. To see the first occurrence of a noncovered line, click on Next Hotspot in the Navigate menu. Figure 6.4 is an example of an annotated source listing that shows which lines were not covered by tests. The display points to the first occurrence of a noncovered line.

Figure 6.4. Test Noncoverage Information By Line
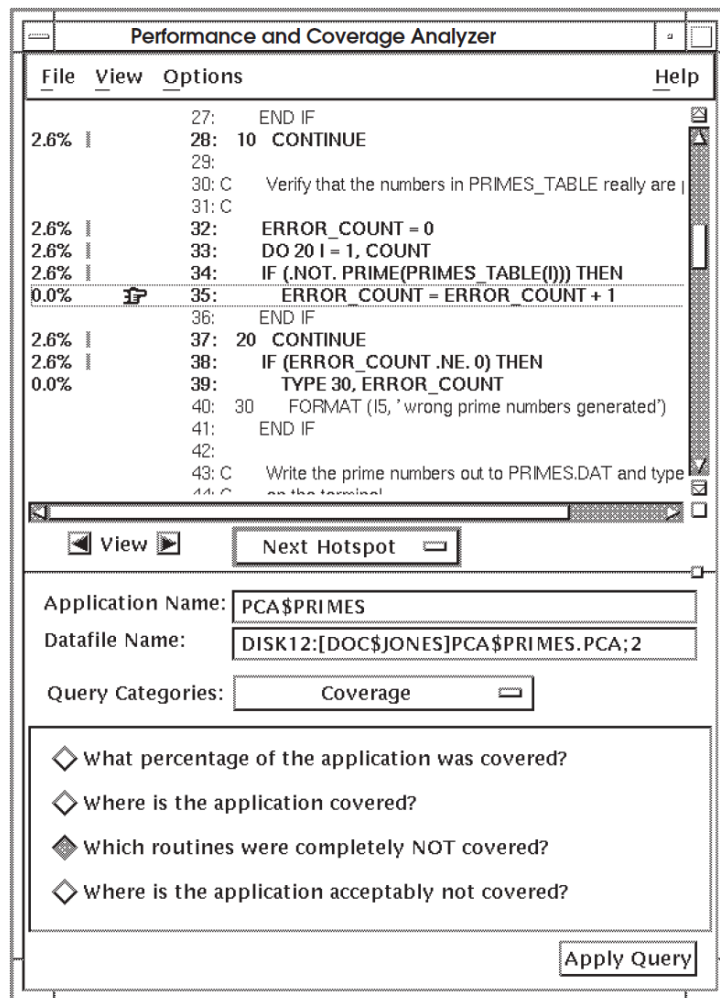
Chapter 7. Determining How Your Application Uses System Resources

This chapter shows how to use PCA to gather the following information about how your application uses system resources, including the following:

- System services use

- I/O services use

- Page fault information

- Ada tasking

Collect system resource usage information if any of the following conditions exists in your application's performance:

- Your application has locations where it is spending idle time—places where the elapsed time is higher than the process time.

- Your application is spending a lot of time in system space.

By examining system resource usage information, you can get clues as to where and how you need to optimize your use of system services and I/O operations.

To obtain system services, I/O usage, page fault, and Ada tasking information on your application, you need to choose queries in the System Service Usage, I/O Usage, Page Faults, and Ada Tasking query categories, respectively. The System Service Usage query category contains the following queries:

- *Where are system services used?*

- *What system services are used?*

The I/O Usage query category contains the following queries:

- *Where is I/O being done?*

- *What I/O system services are being used?*

- *Where are I/O reads being used?*

- *What is the number of I/O read counts?*

- *Where are I/O writes being used?*

- *What is the number of I/O write counts?*

- *Where are I/O reads and writes used?*

- *What files is I/O done to?*

- *What file virtual block numbers are used?*

- *What file keys are used?*

- *What are the record sizes being used?*

The Page Faults query category contains the following queries:

- *Where are the page faults occurring?*

- *What are the addresses causing page faults?*

The Ada Tasking query category contains the following queries:

- *What tasks are performing context switching?*

- *What are the Ada tasking names used?*

- *What are the Ada tasking types used?*

- *What are the Ada task priorities used?*

7.1. Collecting System Services, I/O Use, and Page Fault Information

To collect system services and I/O information on your application using the DECwindows interface, do the following:

1. Compile your application for use with the OpenVMS Debugger.

2. Link your application for use with the Collector.

3. Define the Collector as your process debugger.

4. Run your application.

5. Open a performance data file.

6. Choose the System Service Usage query category from the Query Category option menu.

7. Choose one of the queries in the System Service Usage query panel.

8. Click on Start Collection.

9. In subsequent collection runs, repeat Steps 1 through 4, then choose the I/O Usage and Page Faults query categories. Choose one of the queries, then click on Start Collection.

For more information on performing these steps, see Chapter 2.

The remaining sections in this chapter show how to use the Analyzer to understand the nature of your application's use of system services and I/O services.

7.2. Analyzing System Service Usage Information

To analyze system service information on your application using the DECwindows interface, do the following:

1. Invoke the Analyzer and specify the name of the data file containing the information you want to see. (Chapter 3 describes this step.)

2. Choose the System Service Usage query category.

3. Choose the query "Where are system services used?"

4. Click on Apply.

When you analyze system service usage information, look for the following:

*   Where system services are being used. Check whether your application is calling system services from several different locations. If so, you might want to centralize where your application uses system services.

*   What system services are being used. If a system service is I/O-oriented—for example, SYS$GET or SYS$PUT—focus on your I/O usage activity.

*   Whether you have system services that are consuming a lot of elapsed time. If so, you might improve performance by optimizing your use of those services.

7.2.1. Finding Where System Services Are Being Used

Choosing the query "Where are system services used?" shows you where system services are used in your application. You can see that information by routine or by line in a histogram or table. Figure 7.1 is a histogram showing by routine where system services were used in the application PCA$PRIMES.
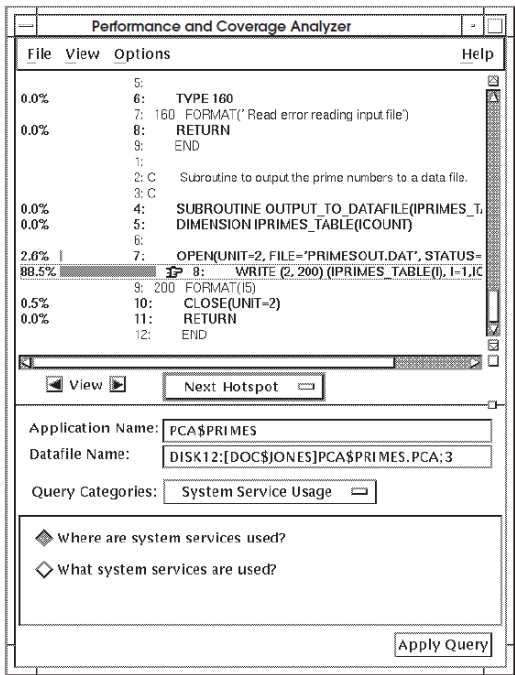
Figure 7.1. Histogram Showing Where System Services Are Used
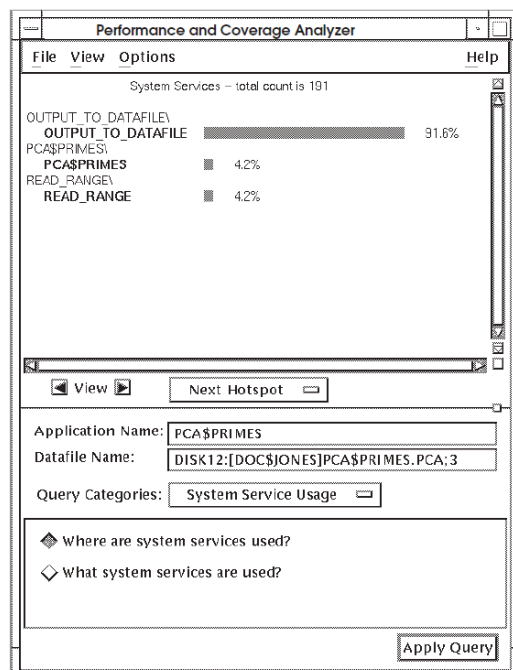
In Figure 7.1, nearly all the system services are used in the routine OUTPUT_TO_DATAFILE.

To get more information on system services usage, click on the right arrow View button. Figure 7.2 shows an annotated source listing of system service usage for each line in the application PCA $PRIMES.
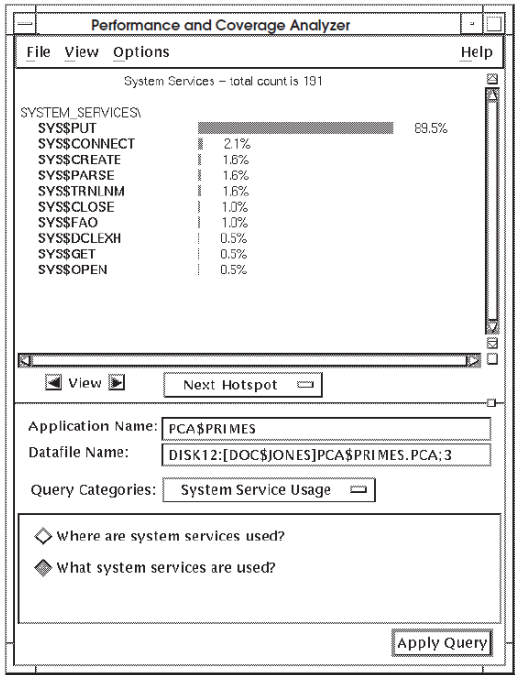
Figure 7.2. Annotated Source Showing Where System Services Are Used

7.2.2. Finding What System Services Are Being Used

Choosing the query "What system services are used?" causes the Analyzer to display a histogram showing what system services are being used in your application, as shown in Figure 7.3.

Figure 7.3. Histogram Showing What System Services Are Used

7.3. Analyzing I/O Usage Information

To analyze I/O usage information in your application using the DECwindows interface, do the following:

1. Invoke the Analyzer and specify the name of the data file containing the information you want to look at. (Chapter 3 describes this step.)

2. Choose the I/O Usage query category.

3. Choose the query "Where is I/O being done?"

4. Click on Apply.

When you analyze I/O usage information, look for the following:

• Where I/O services are being used

• What I/O services are being used

• Whether you have I/O services that generate a lot of idle time

• Whether you have I/O services that generate a lot of execution counts information

7.3.1. Finding Where I/O Operations Are Being Performed

When you choose the query "Where is I/O being done?" the Analyzer displays a histogram showing the routines that are making the most use of I/O operations. Figure 7.4 shows where I/O operations are being performed in the application PCA$PRIMES.

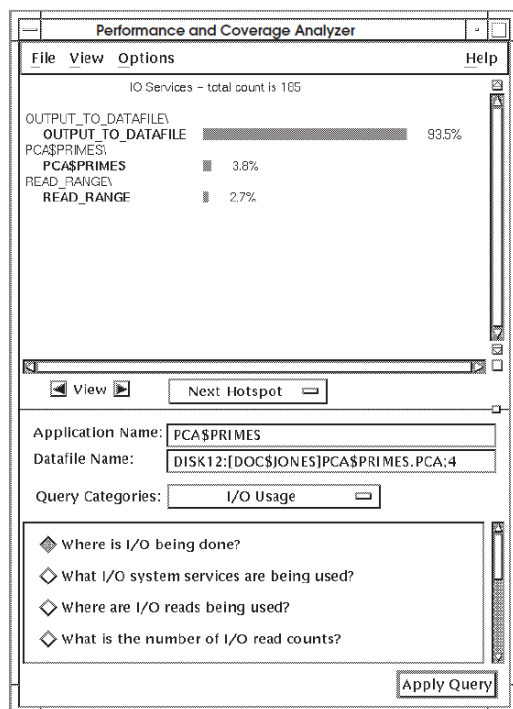**Figure 7.4. Histogram Showing Where I/O Operations Are Being Performed**

Figure 7.4 shows that the routine OUTPUT_TO_DATAFILE is performing 93.5 percent of all the I/O operations in PCA$PRIMES. If your application has a routine in which a large percentage of I/O operations are being performed, get more information on that routine.

If your application performs I/O operations throughout numerous locations and in several different routines, consider centralizing the location of the I/O operations. Compare elapsed time, process time,

and execution count information with the I/O information for that routine. If a routine is consuming idle time in addition to using a lot of I/O services, find out what I/O services are being used.

7.3.2. Finding What I/O Operations Are Being Performed

When you choose the query "What I/O system services are being used?" the Analyzer displays a histogram showing which I/O operations are being used most in your application. Figure 7.5 shows the I/O operations being used the most in the application `PCA$PRIMES`.

Figure 7.5. Histogram Showing What I/O Operations Are Being Performed
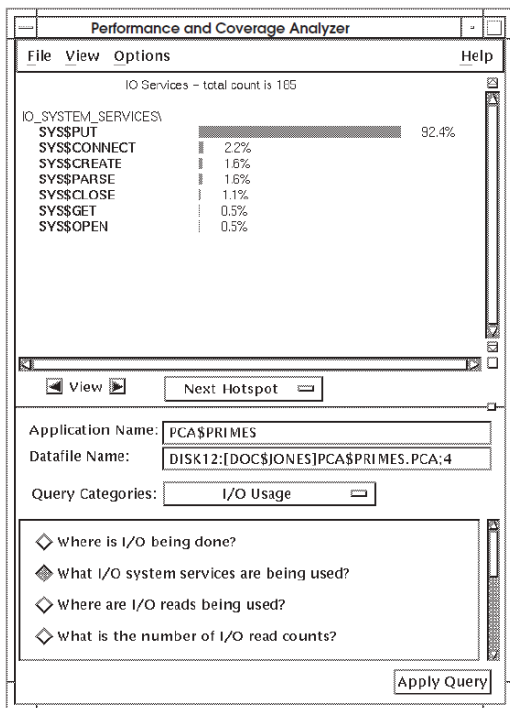
In Figure 7.5, `SYS$PUT` represents 92.4percent of all I/O services used in `PCA$PRIMES`. If your application has a high usage of one I/O service, check to see how it is being used. In this example, check to see where you are sending output to a file and see if that location is generating a lot of idle time. If so, you might look at ways to make that algorithm more efficient, or optimize use of the I/O output statement.

To get more information on the use of output, check the following:

- Where I/O writes are being performed

- The number of I/O write counts

- What files output is being sent to

- Record size information

To get this information, click on the appropriate query in the Analyzer query panel.

7.4. Analyzing Page Fault Information

To analyze page fault information in your application using the DECwindows interface, do the following:

1. Invoke the Analyzer and specify the name of the data file containing the information you want to look at. (Chapter 3 describes this step.)

2. Choose the Page Faults query category.

3. Choose the query "Where are the page faults occurring?"

4. Click on Apply.

When you analyze page fault information, look for the following:

• Where page faults are occurring

• The address of the instruction that caused the page fault

7.4.1. Finding Where Page Faults Are Occurring

When you choose the query "Where are the page faults occurring?"the Analyzer displays a histogram showing the routines that generates the most page faults. Figure 7.6 shows a sample histogram.

Figure 7.6 shows that more than 25 percent of the page faults are occurring in system space. This might indicate that the page faults are caused by system service or I/O calls. To get more information, choose the query "What are the addresses causing page faults?"

Figure 7.6. Histogram Showing Where Page Faults Are Occurring
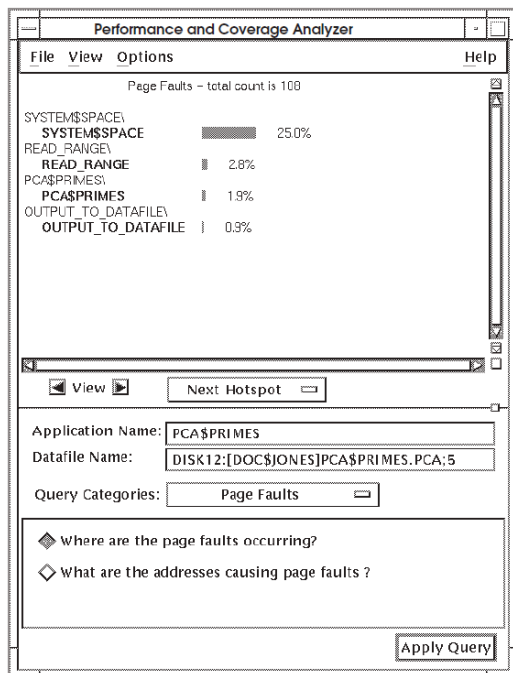
7.4.2. Finding What Instructions Are Causing Page Faults

When you query "What are the addresses causing page faults?", the Analyzer displays the addresses of the instructions that are causing the page faults.

Figure 7.7 shows sample addresses of instructions that cause page faults.

Figure 7.7. Addresses of Instructions Causing Page Faults
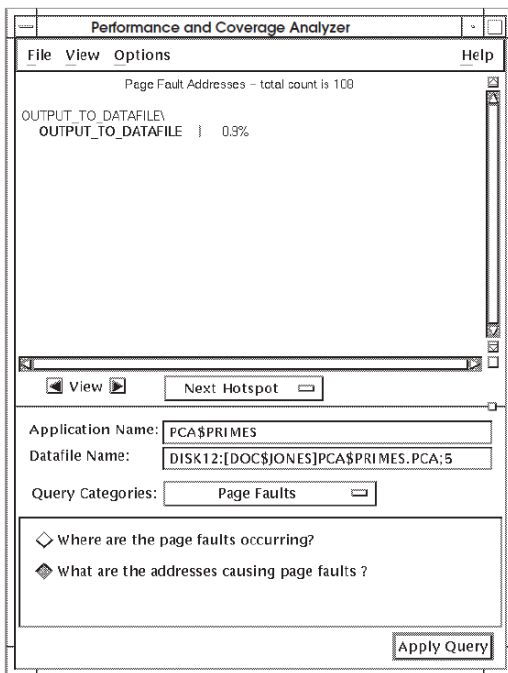
7.5. Analyzing Ada Tasking Information

To analyze Ada tasking information in your application using the DECwindows interface, do the following:

1. Invoke the Analyzer and specify the name of the data file containing the information you want to look at. (Chapter 3 describes this step.)

2. Choose the Ada Tasking query category.

3. Choose the query "What tasks are performing context switching?"

4. Click on Apply.

When you analyze Ada tasking information, look for the following:

- What tasks are performing context switching?

- What Ada tasking names are used?

- What Ada tasking types are used?

- What Ada task priorities are used?

7.5.1. Finding What Tasks Are Performing Context Switching

When you choose the query "What tasks are performing context switching?",the Analyzer displays a histogram showing the percentage of each task that performs context switching.

Figure 7.8 shows that Run 1 and Run 3 represent context switching 17.6 percent of Task 1. Similarly, Run 3 represents the same percent of context switching of Task 3 and Task 4. To get more information, choose the query "What are the Ada tasking names used?"

Figure 7.8. Histogram Showing What Tasks Are Performing Context Switching
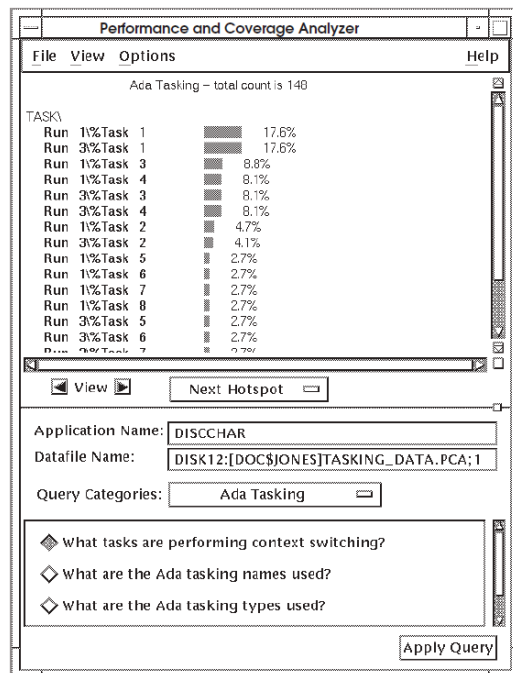
7.5.2. Finding What Ada Tasking Names Are Used

When you query "What are the Ada tasking names used?", the Analyzer displays the percentage of each task name used, as shown in Figure 7.9.
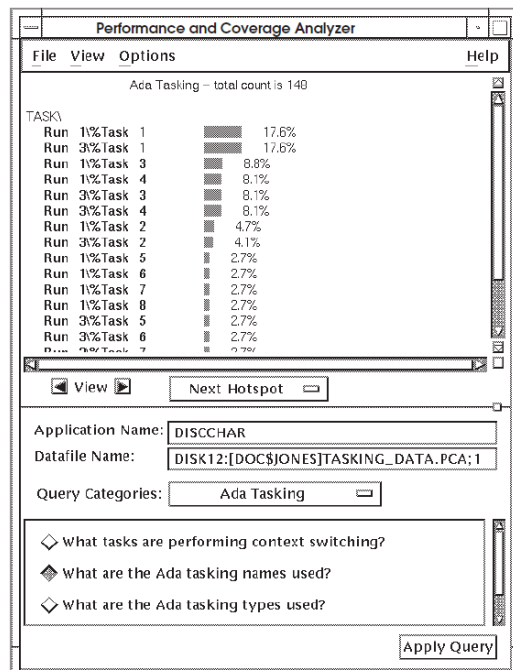
Figure 7.9. Histogram Showing What Ada Tasking Names Are Used

Figure 7.9 shows that Run 1 and Run 3 represent 17.6 percent of Task 1. Similarly, Run 3 represents the same percent of Task 3 and Task 4. For more information, query "What are the Ada tasking types used?"

7.5.3. Finding What Ada Tasking Types Are Used

When you query "What are the Ada tasking types used?", the Analyzer displays the percentage of each task type used, as shown in Figure 7.10.

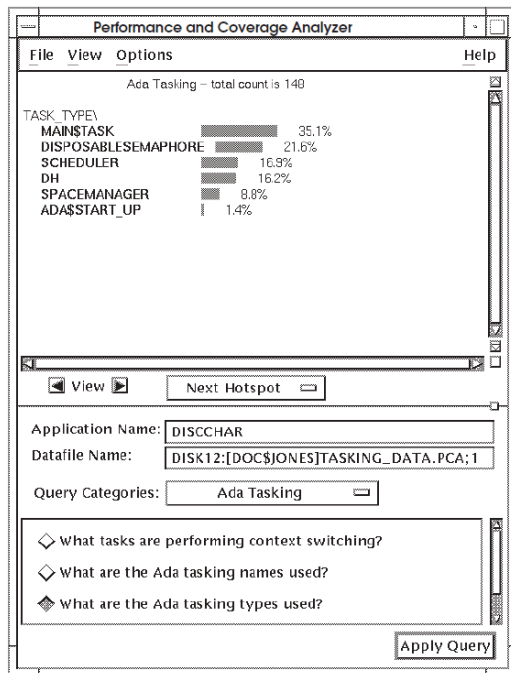**Figure 7.10. Histogram Showing What Ada Tasking Types Are Used**

Figure 7.10 shows that MAIN$TASK represents 35.1 percent and that ADA$START_UP represents only 1.4 percent of all Ada tasking used in DISCCHAR. To get more information, choose the query "What are the Ada task priorities used?"

7.5.4. Finding What Ada Task Priorities Are Used

When you query "What are the Ada task priorities used?", the Analyzer displays the percentage of task priorities used, as shown in Figure 7.11.
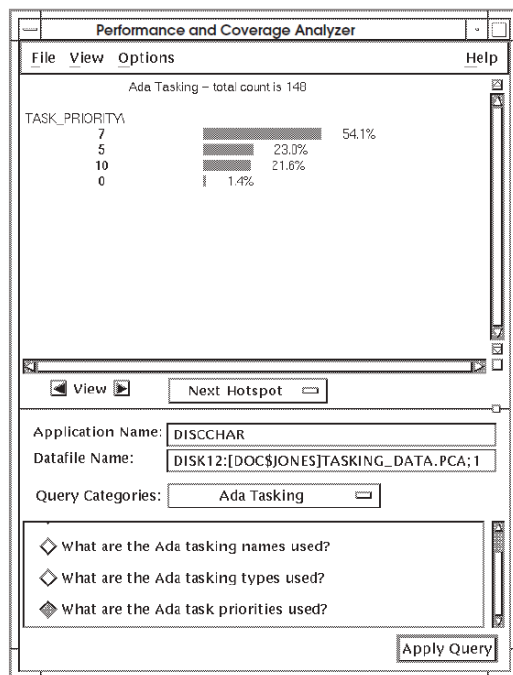
Figure 7.11. Histogram Showing What Ada Task Priorities Are Used

Figure 7.11 shows that task priority 7 represents 54.1 percent, task priority 5 represents 23.0 percent, task priority 10 represents 21.6 percent, and task priority 0 represents only 1.4 percent of all Ada tasking used in DISCCHAR.