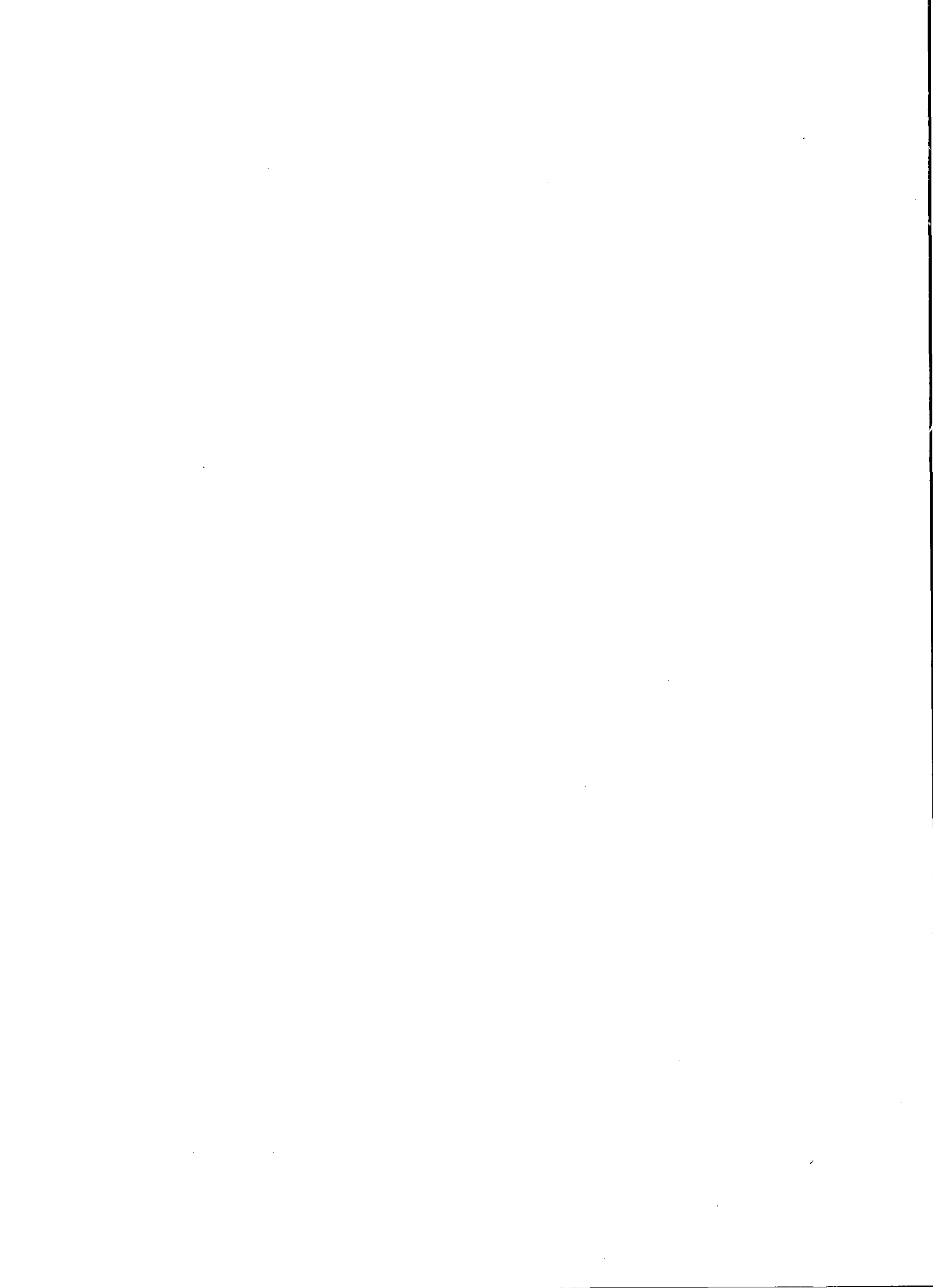
CONVEX CXpa User's Guide

Third Edition

CONVEX

CONVEX COMPUTER CORPORATION

CONVEX Computer Corporation
3000 Waterview Parkway
P.O. Box 833851
Richardson, TX 75083-3851
United States of America
(214)497-4000



CONVEX CXpa User's Guide

Order No. DSW-251

Third Edition
March 1993

CONVEX Press
Richardson, Texas
United States of America

CONVEX CXpa User's Guide

Order No. DSW-251

Copyright ©1993 CONVEX Computer Corporation
All rights reserved.

This document is copyrighted. This document may not, in whole or part, be copied, duplicated, reproduced, translated, electronically stored, or reduced to machine readable form without prior written consent from CONVEX Computer Corporation.

Although the material contained herein has been carefully reviewed, CONVEX Computer Corporation does not warrant it to be free of errors or omissions. CONVEX reserves the right to make corrections, updates, revisions or changes to the information contained herein. CONVEX does not warrant the material described herein to be free of patent infringement.

UNLESS PROVIDED OTHERWISE IN WRITING WITH CONVEX COMPUTER CORPORATION (CONVEX), THE PROGRAM DESCRIBED HEREIN IS PROVIDED AS IS WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. SOME STATES DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES. THE ABOVE EXCLUSION MAY NOT BE APPLICABLE TO ALL PURCHASERS BECAUSE WARRANTY RIGHTS CAN VARY FROM STATE TO STATE. IN NO EVENT WILL CONVEX BE LIABLE TO ANYONE FOR SPECIAL, COLLATERAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, INCLUDING ANY LOST PROFITS OR LOST SAVINGS, ARISING OUT OF THE USE OR INABILITY TO USE THIS PROGRAM. CONVEX WILL NOT BE LIABLE EVEN IF IT HAS BEEN NOTIFIED OF THE POSSIBILITY OF SUCH DAMAGE BY THE PURCHASER OR ANY THIRD PARTY.

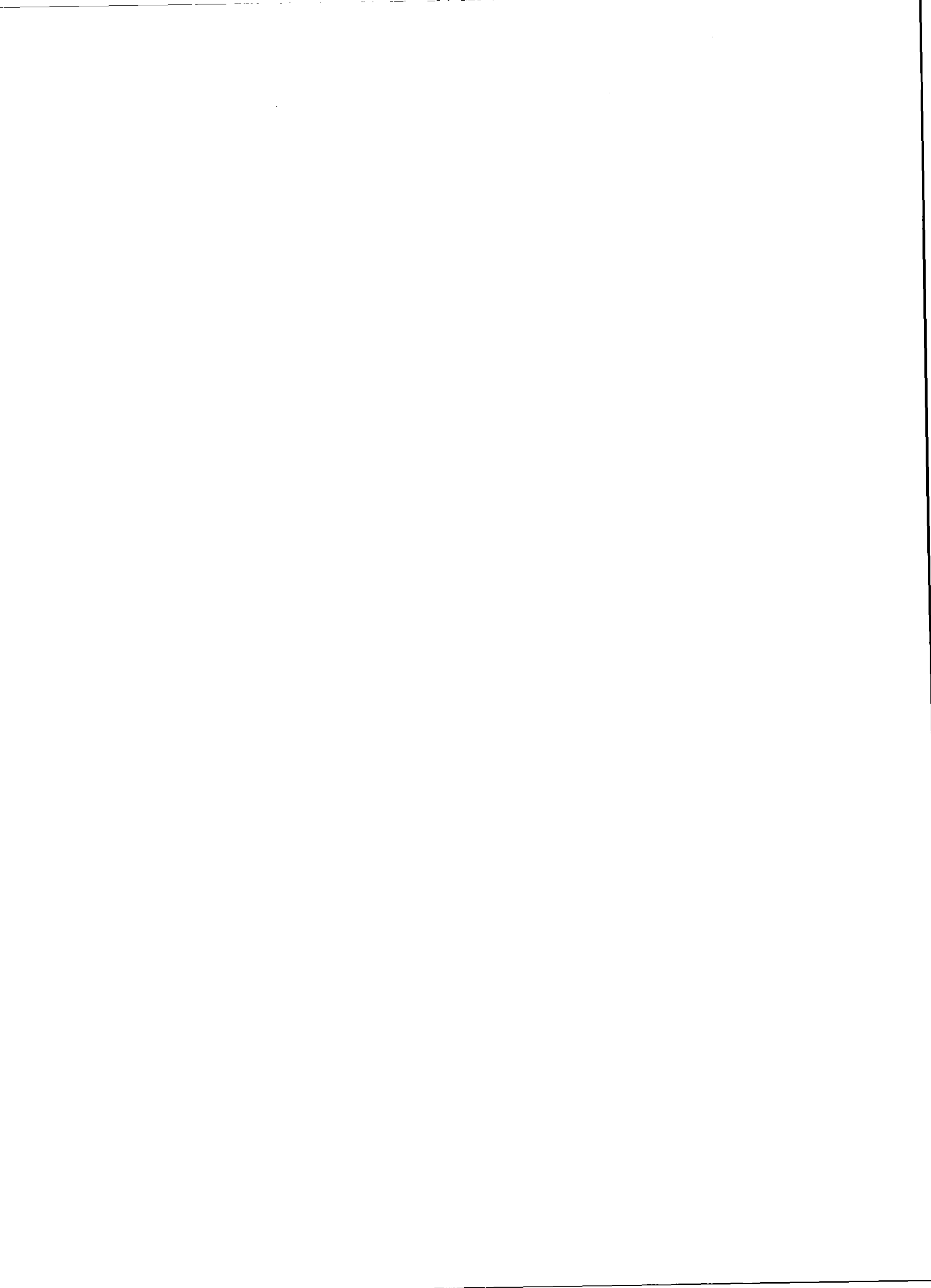
CONVEX and the CONVEX logo ("C") are registered trademarks of CONVEX Computer Corporation.
CXwindows is a trademark of CONVEX Computer Corporation.
PostScript is a trademark of Adobe Systems, Inc.
UNIX is a trademark of UNIX System Laboratories, Inc.
The X Window System is a trademark of the Massachusetts Institute of Technology.

Printed in the United States of America

Revision information for

CONVEX CXpa User's Guide

Edition	Document No.	Description
Third	710-007230-005	Released with CONVEX Performance Analyzer V2.0, March 1993. This edition documents the new graphical user interface for CXpa, the new command set, and offers a new organization for presenting the material.
Second	710-007230-002	Released with CONVEX Performance Analyzer V1.2, May 1990.
First	740-002-930-201	Released with CONVEX Performance Analyzer V1.0, June 1989.



Contents

Using this guide xv

Purpose and audience xv
Organization xv
Notational conventions xvi
  Command syntax xvi
  General conventions xvii
Associated documents xviii
Ordering documentation xviii
Technical assistance xviii
The `contact` utility xix
Acknowledgments xx

1 Introduction 1

Profilers and profiling 1
Why use a profiler? 2
CXpa and profiling 3
Steps for learning CXpa 4

2 Getting started 5

General steps to profiling with CXpa 6
Compiling for profiling with CXpa 7
  CXpa compiler options 7
  Compiling considerations 8
  Compiling to profile routines and loops 8
Invoking CXpa 9
  CRT interface 9
  X interface 12
Enabling monitor points 14
Profiling the program 16
Analyzing performance data 20
Reading performance reports 22
  Routine Report 22
  Loop Report 23
  Parallel Region Report 23
  Basic Block Report 24
Getting information on CXpa 24
  Getting the status of the current CXpa session 24
  Getting version information about CXpa 26

Getting help online 27
Requesting help on a topic 29
Getting help on related topics 31
Quitting CXpa 32

3 Compiling for profiling 33
Compiling a program to run under CXpa 34
Compiling and linking in one step 35
Compiling and linking separately 35
Special compiling considerations 37
Linking with uninstrumented libraries 37
Routines that call uninstrumented routines 38
Routines compiled with both `-pa` and `-pab` . . . 39
Parallel regions not compiled for use with CXpa . . 39
Language-specific information 39
Compiling FORTRAN source code for CXpa . . . 39
Compiling Ada source code for use with CXpa . . 40

4 Using monitor points 41
Understanding monitor points 42
Choosing the monitor points to enable 44
Enabling monitor points from the X interface 45
Enabling all monitor point types in all routines . . 45
Enabling one monitor point type in all routines . . 46
Enabling all monitor point types in select routines . 47
Enabling one monitor point type in select routines . 47
Using the Set Monitor Points dialog box 48
Opening the Set Monitor Points dialog box . . . 48
Selecting the monitor point type 50
Selecting routines from the Disabled or Enabled list 52
Applying changes 54
Closing the Set Monitor Points dialog box . . . 54
Enabling monitor points using the command line . . . 55
Enabling all monitor point types in all routines . . 55
Enabling one monitor point type in all routines . . 57
Enabling all monitor point types in select routines . 59
Enabling one monitor point type in select routines . 61
Enabling all monitor point types at select lines . . . 63
Enabling one monitor point type at select lines . . . 64
Disabling monitor points 66

Listing monitor points 68
Search paths and listing monitor points 68
Listing monitor points from the X interface 69
Listing monitor points from the command line . . . 71
Listing only monitor points 71
Listing all source lines 72
Specifying the search path 74
Setting the search path from the command line . . 75
Setting the search path from the X interface 76
Setting the search path at the shell prompt 81

5 Controlling profiling 83

Specifying an executable 84
Setting the performance data file (PDF) 85
Setting the PDF from the X interface 86
Selecting a PDF 87
Filtering files 88
Setting the current PDF from the command line . . 88
Setting the PDF from the shell prompt 89
Running your program 89
Running the program from the X interface 90
Running the program from the command line . . . 93
Pausing the profiling of a program 94
Pausing profiling from the X interface 94
Pausing profiling at the command line 95
Continuing the profiling of a program 96
Continuing profiling from the X interface 96
Continuing profiling at the command line 96
Stopping the profiling of a program 97
Stopping profiling from the X interface 97
Stopping profiling at the command line 97
Rerunning the program 98
Rerunning the program from the X interface . . . 98
Rerunning the program from the command line . . 99

6 Generating and interpreting reports . . . 101

Generating reports 102
Generating reports from the X interface 102
Generating reports from the command line 103
Printing reports 104
Printing reports from the command line 104
Printing reports from the X interface 105
Printing to a printer 106
Printing to a file 106

Routine Report 107
Routine Performance Analysis table 107
Summary section 107
Details section 109
Dynamic Call Graph table 110
Loop Report 112
Summary section 112
Details section 114
Vectorized loop section 116
Parallel Region Report 117
CONVEX C Series parallel processing 117
Parallel regions 118
Profiling with fixed scheduling 120
Summary section 121
Details section 122
Parallel efficiency theory 123
Concurrency per processor method 123
Serial CPU time versus parallel PVT method . . 124
Basic Block Report 124
Using the Bar Graph window 126
Printing the bar graph 127
Printing to a printer 128
Printing to a file 128
Closing the bar graph 129

A Interpreting Mflops data 131
Vectorized loop section 132
Vector chaining 133
Vector chaining and functional units 133
Functional unit reservation 136
Register bank reservation 138
Chimes 138
Calculating chimes 139
Vector Mflops 140
Calculating estimated Mflops 141
Calculating peak Mflops 142

Index 143

Figures

Figure 1 Compiling a FORTRAN program for CXpa . . . 8
Figure 2 CRT interface 10
Figure 3 Invoking CXpa 13
Figure 4 Using keyboard mnemonics to open a menu 14
Figure 5 Enabling all monitor points from the X interface 15
Figure 6 Opening the Control Center dialog box . . . 17
Figure 7 Running the program to begin profiling . . . 18
Figure 8 Control Center with process running 19
Figure 9 Analyzing all performance data 21
Figure 10 Getting session status 25
Figure 11 Getting CXpa's version information 26
Figure 12 Accessing the CXpa help system from the main window 28
Figure 13 Requesting help on a topic 30
Figure 14 Selecting a related topic 31
Figure 15 Quitting CXpa 32
Figure 16 Compiling and linking in one step 35
Figure 17 Compiling and linking separately 36
Figure 18 Linking with uninstrumented libraries . . . 38
Figure 19 Instrumented routine calling an uninstrumented routine 38
Figure 20 Compiling for CXpa using `a.make` 40
Figure 21 Example of routine and loop monitor points 43
Figure 22 Using the Monitor All option 45
Figure 23 Enabling all monitor points of a particular type 46
Figure 24 Opening the Set Monitor Points dialog box . . 48
Figure 25 Set Monitor Points dialog box 49
Figure 26 Selecting all types of monitor points 51
Figure 27 Selecting multiple routines 52
Figure 28 Moving selected routines between lists . . . 53
Figure 29 Applying the current monitor point setttings 54
Figure 30 Closing the Set Monitor Points dialog box . . 54
Figure 31 Enabling all monitor point types in all routines 56
Figure 32 Enabling all routine monitor points in all routines 58

Figure 33 Enabling all monitor point types in selected routines 60
Figure 34 Enabling all loop monitor points in selected routines 62
Figure 35 Enabling all monitor point types at a line . . 63
Figure 36 Enabling one monitor point type at a line . . 65
Figure 37 Disabling all monitor points in all routines . . 67
Figure 38 Opening the Monitor Point Information dialog box 69
Figure 39 Using the Monitor Point Information dialog box 70
Figure 40 Listing monitor points from the command line 72
Figure 41 Listing source lines from the command line . . 73
Figure 42 Displaying the current search path from the command line 75
Figure 43 Adding to the search path from the command line 75
Figure 44 Replacing the search path from the command line 75
Figure 45 Displaying the search path from the X interface 76
Figure 46 Opening the Change Search Path dialog box . 77
Figure 47 Adding to the search path from the X interface . 78
Figure 48 Removing a directory from the search path . . 79
Figure 49 Applying changes to the search path from the X interface 80
Figure 50 Setting the search path using the shell prompt . 81
Figure 51 Specifying an executable when invoking CXpa 84
Figure 52 Setting the current PDF from the X interface . 86
Figure 53 Set PDF dialog box 87
Figure 54 Setting the current PDF at the command line . 88
Figure 55 Invoking CXpa with a specified PDF 89
Figure 56 Opening the Control Center dialog box . . . 90
Figure 57 Running the program from the X interface . . 92
Figure 58 Starting to profile using the command line . . 93
Figure 59 Pausing profiling from the X interface 94
Figure 60 Pausing the program from the command line . 95
Figure 61 Continuing profiling from the X interface . . 96
Figure 62 Continuing profiling from the command line . 96
Figure 63 Stopping profiling from the X interface . . . 97
Figure 64 Stopping profiling from the command line . . 97
Figure 65 Rerunning the program 98
Figure 66 Rerunning the program from the command line 99
Figure 67 Generating all reports from the X interface . . 102
Figure 68 Generating a report from the command line . 103
Figure 69 Printing a report from the command line . . . 104
Figure 70 Opening the Print dialog box 105
Figure 71 Printing the report from the X interface . . . 106
Figure 72 Summary section of the Routine Performance Analysis table 108
Figure 73 Details section of the Routine Performance Analysis table 110

Figure 74	Dynamic Call Graph table	111
Figure 75	Summary section of the Loop Performance Analysis table	113
Figure 76	Details section of the Loop Performance Analysis table	114
Figure 77	Vectorized loop section of the Loop Performance Analysis table	116
Figure 78	Parallel region	119
Figure 79	Invoking CXpa with fixed scheduling enabled	120
Figure 80	Summary section of the Parallel Region Analysis table	121
Figure 81	Details section of the Parallel Region Analysis table	122
Figure 82	Basic Block Performance Analysis table	125
Figure 83	Opening the Bar Graph window	126
Figure 84	Print graph dialog box	127
Figure 85	Printing the bar graph to a file	128
Figure 86	Closing the bar grapht	129
Figure 87	Vectorized loop section of the Loop Performance Analysis table	132
Figure 88	Effect of vector chaining	135
Figure 89	Chime calculation on C3200 Series	139
Figure 90	Calculating the vector flops of a loop	141

Tables

Table 1 CXpa compiler options 7
Table 2 CRT interface key bindings 11
Table 3 Command Window key-bindings 12
Table 4 CXpa compiler options. 34
Table 5 Monitor point annotations in source listings . . 42
Table 6 Set Monitor Points dialog box buttons 50
Table 7 Commands for enabling monitor points in all routines 57
Table 8 Commands for enabling monitor points in routines 61
Table 9 Commands for enabling monitor points at a line 64
Table 10 Commands to disable monitor points 66
Table 11 Monitor point annotations in source listings . . 68
Table 12 Profiling status annotations 109
Table 13 CXpa annotations for compiler optimizations. . 115
Table 14 C Series vector functional units. 137

Using this guide

Purpose and audience

The *CONVEX CXpa User's Guide* explains how to use the CONVEX profiler, CXpa V2.0. CXpa provides performance analysis for CONVEX C, FORTRAN, and Ada applications.

This book provides a quick introduction to profiling with CXpa that covers typical profiling needs. Each major profiling step is introduced and explained. Each of these steps, in turn, is discussed in detail in its own chapter, providing complete coverage of CXpa's functionality.

Particular emphasis is given to the X Window System interface to CXpa. CXpa commands are also discussed; however, this book does not cover every detail of the CXpa command set. For a complete description of every CXpa command, refer to the *CONVEX CXpa Reference* or the online help system.

This book is intended for analysts, programmers and scientists interested in analyzing and improving program performance of their CONVEX applications. It does not assume you have any prior knowledge of profiling. A general overview of profiling is given in Chapter 1, "Introduction."

Organization

This book includes a quick introduction to CXpa and profiling, and in-depth coverage of all CXpa's features.

The first two chapters are intended for those who have little or no prior experience with profilers or CXpa. The remaining chapters provide detailed explanations of each major function in CXpa. These chapters serve as a guide after the basic features of CXpa are mastered and more detailed profiling functionality is needed.

If you are new to profiling, read Chapter 1, "Introduction." This introduces profiling and the particular features of CXpa that will enable you to improve the performance of your programs.

If you are new to CXpa, profile a small program you are very familiar with while reading Chapter 2, "Getting started." This will introduce each major profiling step as you learn CXpa.

For an explanation of the reports generated by CXpa, refer to Chapter 6, "Generating and interpreting reports."

The remaining chapters each detail a profiling task:

- Chapter 3 describes how to compile your program for CXpa. It explains CXpa compiler options and discusses special considerations when compiling your program for profiling.
- Chapter 4 explains how to specify the areas of your program you want to monitor.
- Chapter 5 describes how to run, pause, continue, and stop your program under CXpa.
- Chapter 6 explains how to generate and interpret CXpa's performance reports. This chapter also covers the bar graph available in the X Window System interface.
- Appendix A details the special report generated for vectorized loops. This chapter explains how CXpa calculates and reports chime analysis and Mflops (millions of vector floating point operations per second) measurements.

Notational conventions

This section discusses notational conventions used in this book.

Command syntax

Consider this example:

```
(CXpa) command <param1> [, ...] {a | b} [<param2>]
  ①       ②         ③       ④      ⑤        ⑥
```

1. `(CXpa)` is the CXpa command prompt.
2. **`command`** must be typed as it appears.
3. <*param1*> indicates a parameter that must be supplied.
4. The horizontal ellipsis in brackets [`,` ...] indicates that additional parameters may be specified.
5. Either **`a`** or **`b`** must be specified.
6. [<*param2*>] indicates an optional parameter.

General conventions

In general, the following conventions are used in this guide:

- *Italics:*
  - Designate user-supplied variables in a command-line example
  - Designate variable fields in data descriptors
  - Introduce new and important terms
  - Indicate document titles
- **`Bold, constant-width font`** designates user input in screens and examples.
- `Constant-width font` designates:
  - Command names and options
  - System calls and function names
  - Data structures and types
  - Directives, program statements, display examples, printout examples, and error messages returned
  - System output in screens and examples
- Horizontal ellipsis (...) shows repetition of the preceding item(s).
- Vertical ellipsis shows that lines of input or output have been left out of an example.
- Words and abbreviations that indicate keyboard keys you press are identified in a distinctive bold type. For example, **RETURN** refers to the carriage return key. Words separated by a hyphen indicate two keys that you must press simultaneously. For example, **CTRL-X** indicates that you must press and hold down the **CTRL** key and then press the **X** key.
- The word "enter" in a phrase such as "enter **`ls`**" means that you type the command and then press **RETURN**.
- The ConvexOS prompt is printed as a percent sign (`%`).

Note

A note highlights important information.

Associated documents

Using this software may require information not specific to the tasks described in this document.

For more information on CONVEX CXpa, you can order these books from CONVEX Computer Corporation:

- *CONVEX CXpa Reference* (DSW-253) provides a reference source for CXpa commands, windows, and messages.
- *CONVEX CXpa Quick Reference* (DSW-252) is a quick reference to CXpa command syntax, with brief descriptions of command functionality.
- *CONVEX C Optimization Guide* (DSW-089) describes scalar, vector, and parallel optimizations performed by the CONVEX C compiler.
- *CONVEX FORTRAN Optimization Guide* (DSW-034) describes scalar, vector, and parallel optimizations performed by the CONVEX FORTRAN compiler.
- *CONVEX Ada Optimization Guide* (DSW-144) describes scalar, vector, and parallel optimizations performed by the CONVEX Ada compiler.

Ordering documentation

To order the current edition of this or any other CONVEX document, send requests to:

CONVEX Computer Corporation
Customer Service
P.O. Box 833851
Richardson TX 75083-3851 USA

Include the order number or the exact title, as listed on the front cover.

Technical assistance

If you have questions that are not answered in this book, contact the CONVEX Technical Assistance Center (TAC).

- Within the continental U.S., call 1(800)952-0379.
- From Canada, call 1(800)345-2384.
- Outside continental U.S., contact your local CONVEX office.

The `contact` utility

The TAC recommends using the `contact` utility to report a hardware, software, or documentation problem. The `contact` utility is an interactive program that helps the TAC track reports and route them to the CONVEX personnel most qualified to fix a problem.

After you invoke `contact`, it prompts you for information about the problem. When you finish your report, `contact` mails it to the TAC electronically. The TAC notifies you within 48 hours that your report has been received.

Using `contact` requires:

- UNIX-to-UNIX Communication Protocol (UUCP) connection to the TAC
- Full path name of the program or utility in question
- Version number of the program or utility in question

Refer to the contact(1) man page for complete details.

Acknowledgments

The author wishes to thank everyone at CONVEX who contributed their ideas and time in the development of this book.

In particular, the author would like to thank:

- Chuck Summers, Tracy Webb, and Jeff Woods for the development of CXpa V2.0
- Brent Henderson and Marianne Becker, for ensuring product quality through extensive testing
- Keith Knox and Ray Cetrone, for many suggestions that improved the book's organization and flow
- Gary Brooks, for insightful management and a productive work environment
- Steve Simmons, for his contributions to CXpa's Mflops reporting
- Randy Torres and Mike Gafka, for representing CXpa in the Technical Assistance Center
- Ron Gray, for providing a basis for CXpa's Mflops reporting
- Mary Clare Bernier, for editing the documentation and improving its overall quality

—Ken Harward

Introduction 1

This chapter introduces general profiling concepts and the CONVEX performance analyzer, CXpa. If you are new to profiling, read this chapter before using CXpa.

The topics discussed in this chapter include:

- Overview of profiling and profilers
- Overview of the CONVEX performance analyzer, CXpa
- Typical approach to profiling with CXpa

After reading this chapter, try using CXpa on one of your own programs while reading Chapter 2, "Getting started."

Profilers and profiling

A profiler is a software tool that measures the run-time performance of a program. Profiling typically measures the total time it takes each portion of your program to execute. Additional information may be measured as well, such as the number of times routines or loops are executed.

Running your program under a profiler's control enables the profiler to collect performance measurements. After the measurements are analyzed, the profiler reports the performance data (often as a table to make it easier to compare routine times).

The method of collecting performance data differs between profilers. Some profilers sample a program's performance at measured intervals, obtaining an average of a routine's execution time. This is known as statistical sampling. The `prof` utility uses statistical sampling.

Other profilers, including CXpa, measure a program's entire execution time, reporting the total time spent in individual routines, loops, parallel regions, and basic blocks (a section of code with a single entry point and single exit point). This produces more accurate profiling results than statistical sampling.

With most profilers, including CXpa, the program's executable must be modified for use with the profiler. When a program is compiled with a special compiler option, the resulting executable contains extra information that the profiler uses to collect performance data.

Once your program is compiled, you can run it under the profiler's control. The profiler gathers performance data during execution and reports the results.

Why use a profiler?

Profiling a program is different from most other development steps because it typically happens after a program is working correctly. It may occur at the end of a development cycle and, because it is not a required step, may be overlooked completely.

Yet profiling is an important step. A profiler cannot pinpoint lines of troublesome source code and suggest changes. But a good profiler can help you learn how your program executes and where that execution time is spent.

Using the information CXpa provides, you can learn which routines and loops are slowest, and which are fastest. Profiling your program compiled at different optimization levels can provide insight into the types of optimizations that work best for given situations.

In some cases, simple modifications to the source code can result in significant performance improvements. Without profiling, you might not be able to find performance bottlenecks.

Profiling is an iterative process. By profiling, making changes to your program, and profiling again, you can improve the performance of your program and develop a better understanding of code performance.

CXpa and profiling

CXpa is an interactive profiler, capable of profiling programs compiled by the CONVEX Ada, C, and FORTRAN compilers. Using CXpa, you can profile selected parts of your program, control your program's execution, and view performance reports.

CXpa enables you to profile your program in great detail. You can profile at the routine level, or you can profile the loops within the routines. CXpa also profiles optimized loops, including parallel regions created by the compiler. Finally, CXpa can profile at the block level, including the number of times each basic block executed.

CXpa provides a variety of performance statistics. For each routine, CXpa reports:

- Total CPU time, not including routines called
- Total CPU time, including routines called
- Total number of times routine was called
- Minimum, maximum, and average time spent in a single call to the routine

For each loop, CXpa reports:

- Total CPU time, not including nested loops
- Total CPU time, including nested loops
- Minimum, maximum, and average number of iterations for the loop
- Optimizations performed on the loop
- The loop's nesting level
- Vector floating point performance metrics for vectorized loops

For each parallel region, CXpa reports:

- Total CPU time
- Total wall-clock time
- Total Process Virtual Time (PVT). Process virtual time is time that at least one thread of a process is executing.
- Concurrency factor (parallel efficiency)
- Information about each thread created
- Chore counts for each thread (representing the distribution of the work load among threads)

For each block, CXpa reports:

- Total number of times executed
- Percentage of routine's total CPU time
- Program counter address for the start of the basic block

Steps for learning CXpa

Profiling a program is an iterative process. You profile the program, make changes based on the results, and profile again. This profiling experience and an understanding of the CONVEX compiler's optimzations, are crucial to improving program performance.

With that in mind, here are some suggested guidelines for gaining practical experience using CXpa:

Step 1 Select a small program you have written (or are very familiar with) that takes several seconds to execute and contains some loops.

Step 2 Read Chapter 2, "Getting started." As you read the chapter, perform the profiling steps on your program.

Step 3 Recompile the program at a higher optimization level and profile it again. Continue doing this until you have found the optimization level at which your program performs best.

Step 4 Identify the routine that takes the longest to execute. If the routine has loops, identify the loop that took the longest to execute.

At this point you have identified the location in your program that is having the greatest effect on your program's performance.

Step 5 Try rewriting the loop (or routine) in another way. Use the CONVEX optimization guides for C and FORTRAN for assistance.

Step 6 Finally, recompile and profile the program again. Compare the execution time of the loop to the previous time.

You can repeat steps 4 through 6 to continue to identify areas of your program that are taking the most CPU time and try to improve their performance.

Getting started

2

This chapter introduces the most common steps performed when profiling with CXpa. These steps provide a great deal of performance information, are the quickest approach to profiling, and typically suffice for general profiling.

As you read this chapter, you may find it helpful to perform the steps on a simple program you are familiar with. In doing so, you will quickly become familiar with CXpa.

This chapter explains how to:

- Compile a program for CXpa
- Invoke CXpa
- Select the regions of your program you want to profile
- Profile your program to collect performance data
- Generate performance reports from the data
- Get information on the current CXpa session
- Use the online help system
- Quit CXpa

General steps to profiling with CXpa

Tuning application performance is an iterative process. Each iteration identifies another area of the program whose performance you may want to improve. After making a change to the program and profiling again you can determine if the change improved performance, and perhaps identify the next area of the program to improve.

In general, profiling consists of the following steps:

Step 1 Compile your source code for use with CXpa.

Step 2 Invoke CXpa with the name of the executable.

Step 3 Select the areas of your program to profile.

Step 4 Run the program to collect performance data.

Step 5 Generate a performance report.

Step 6 Interpret the profile report.

After viewing the profile report, you may choose to profile different areas or make changes to the source file. Or, you may want to recompile with a different optimization and profile the program again.

These steps are further described throughout the rest of this chapter.

Compiling for profiling with CXpa

To run your program under CXpa, you must compile it with a CXpa compiler option. The compiler adds information to the executable that enables CXpa to collect performance data while the executable is running. A program that has this additional information is said to be *instrumented.*

The CXpa compiler options are briefly described below, followed by an example of compiling for CXpa using the most common compiler option.

For more details on compiling for use with CXpa, refer to Chapter 3, "Compiling for profiling."

CXpa compiler options

There are three compiler options you can use to prepare your program for profiling. Each option enables you to profile your program in different ways. These options are summarized in Table 1.

Table 1 CXpa compiler options

Compiler option	Enables CXpa to profile
`-pa`	Routines, loops, and parallel regions
`-par`	Routines only
`-pab`	Blocks only

In general, the `-pa` compiler option provides the most complete coverage for profiling. This enables you to profile routines, loops, and parallel regions.

Note

You cannot profile loops unless you compile at optimization level `-O1` or higher. This is because the compiler does not track the needed information at lower optimization levels.

If you want to profile basic blocks to track which basic blocks are executed and how many times, you could compile your program with the `-pab` option.

Compiling considerations

Each source file that you compile for use with CXpa should be compiled with only one of the above options. If you compile a source file with both the `-pa` and `-pab` option, the resulting profile for the routines within that source file will be inflated. For further details on compiling limitations, refer to Chapter 3, "Compiling for profiling."

You only need to use a CXpa compiler option on the source files you want to profile with CXpa.

Compiling to profile routines and loops

Typically, you will want to profile your routines and the loops within those routines. To do this, compile all of your source files with the `-pa` option, as shown Figure 1.

Figure 1
Compiling a FORTRAN program for CXpa

```
% fc -pa prog.f -c
% fc -pa subs1.f -c
% fc -pa subs2.f -c
% fc -pa prog.o sub1.o subs2.o
```

In the above example, the compiler generates profiling information for all routines and loops in the source files prog.f, subs1.f, and subs2.f. The `-c` compiler option generates object files from the source files. These object files are linked together with CXpa's support routines into an executable in the last step.

If you do not compile any file of your program with a CXpa compiler option, you will be unable to use CXpa to profile the routines in your program.

Invoking CXpa

Once you have compiled your program, you can invoke CXpa using the `cxpa` command followed by the name of your executable. The executable must have been compiled for use with CXpa.

If your executable is named `a.out` and is located in the current directory, you do not need to specify it when invoking CXpa. When you do not specify an executable, CXpa looks for `a.out` in the current directory. If `a.out` does not exist, you can still use CXpa to look at reports generated in previous session, but you will be unable to profile a program and generate new reports.

After CXpa processes any additional options used to invoke it, CXpa executes any commands found in two initialization files:

- `$HOME`/.cxpainit—Your default initialization file
- ./.cxpainit—Your local initialization file

These two initialization files can be used to automatically set up the CXpa environment, such as adding directories to the search path. Initialization files, or command files, can also be executed during a CXpa session by using the `source` command. For more information on the `source` command, refer to the *CONVEX CXpa Reference.*

After CXpa executes the commands in the initialization files (if they exist), the CXpa window appears. There are two interfaces to CXpa:

- CRT interface that can run on any ASCII terminal
- X interface that runs under the X Window System

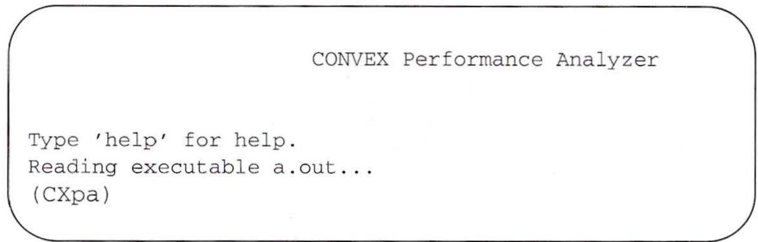
Each interface is introduced in the following two sections.

CRT interface

The CRT interface runs on most ASCII terminals, such as the VT100. The CRT interface can be accessed using the `-nw` option when invoking CXpa. CXpa displays a command line (the CXpa prompt) for entering commands.

Figure 2 is an example of the CRT interface of CXpa.

Figure 2
CRT interface



Note

CXpa cannot run on certain terminal types, such as the VT200, VT220, and VT320. If CXpa cannot use the terminal, or if your `$TERMCAP` setting has too limited a set of capabilities, CXpa prints an error message and exits.

The command line provides the same base functionality as the X interface. You can set monitor points, profile the program, and generate performance reports. You can also access the CXpa online help system.

CXpa uses the pager specified in your `$PAGER` environment variable for paging long output (such as reports). If the `$PAGER` variable is not set, CXpa uses the `more` command.

The command line provides the following additional functionality that is not available through the X interface:

- Profile individual regions within a routine, in addition to profiling all such regions in a routine
- List any source file in your program
- Execute command files using the `source` command.

When using the CRT interface, you can use key bindings to edit the command line. The key bindings available in the CRT interface are given in Table 2.

Table 2 CRT interface key bindings

Function	Key binding
beginning-of-line	CTRL-a
end-of-line	CTRL-e
forward-character	CTRL-f
backward-character	CTRL-b
forward-word	ESC-f
backward-word	ESC-b
delete-character	CTRL-d
backward-kill-character	DEL
backward-kill-character	CTRL-h
erase-line	CTRL-g
kill-line	CTRL-k
kill-word	ESC-d
backward-kill-word	ESC-DEL
backward-kill-word	ESC-h
transpose-character	CTRL-t
transpose-words	ESC-t
capitalize-word	ESC-c
upcase-word	ESC-u
previous-command	ESC-p
next-command	ESC-s
help-keys	ESC-B
redraw-screen	CTRL-l

X interface

The X Window System interface to CXpa provides a window with menus for access to CXpa's functionality. To use the X interface, you must be using an X display.

As with the command line, you can specify the regions to profile, profile the program, generate performance reports, and access the CXpa help system. Using the X interface you can also:

- Graphically compare routine CPU times using a bar chart
- Print performance reports directly to the printer
- Open the Command Window that can accept all CXpa commands. The Command Window enables you to access the full functionality of the command line.

  In the Command Window, you can edit the command line using key bindings. The available key bindings are listed in Table 3.

Table 3 Command Window key-bindings

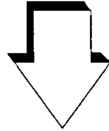
Function	Key binding
`beginning of line`	CTRL-a
`end-of-line`	CTRL-e
`forward-character`	CTRL-f
`backward-character`	CTRL-b
`delete-character`	CTRL-d
`backward-kill-character`	DEL
`backward-kill-character`	CTRL-h
`delete-to-right`	CTRL-k
`delete-to-left`	CTRL-u
`previous-command`	CRTL-p
`next-command`	CRTL-n

CXpa tries to automatically use the X interface if your `$DISPLAY` environment variable is set. When CXpa is invoked, the CXpa window appears, as shown in Figure 3.

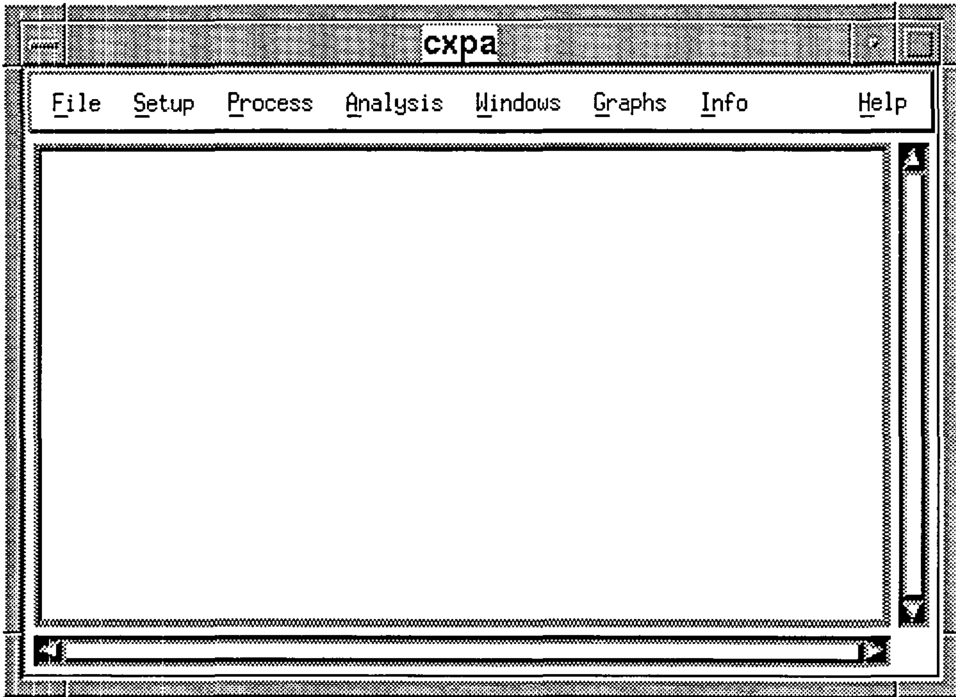
Figure 3
Invoking CXpa

Shell window

% **cxpa a.out**



CXpa window

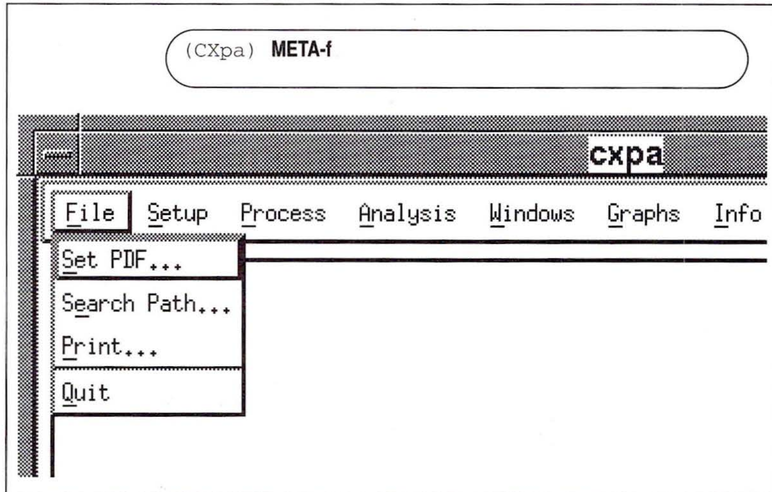


The `cxpa` command in Figure 3 invokes CXpa with the executable `a.out`. The CXpa window appears. The menu bar at the top of the window provides access to CXpa's functionality.

The display area and scroll bars are used for viewing reports.

Every menu item on the menu bar can be accessed by a keyboard shortcut , called a mnemonic. Open a menu by holding down the **META** key and typing the underlined letter on your keyboard. For example, to open the File menu, type **META-f**, as shown in Figure 4.

Figure 4
Using keyboard mnemonics to open a menu



The options under a menu also have keyboard mnemonics, identified by the underlined letter. To access an option, type its letter while the menu is open.

The examples throughout the rest of the chapter use the X interface of CXpa. Where appropriate, the command equivalent is also given. For more information on using CXpa commands, refer to the *CONVEX CXpa Reference*.

Enabling monitor points

A monitor point is a location in your executable where performance data can be collected. Before profiling a program with CXpa, you must first specify the areas of your program that you want to profile. You do this by enabling the monitor points at these regions.

A monitor point's type determines the kind of data it collects. There are four types of monitor points, corresponding to the four types of areas that you can profile:

- **Routine**—Collects data for the entire routine, including CPU time and number of times called.
- **Loop**—Collects data for the loop, including CPU time and iteration count.
- **Parallel region**—Collects data for the parallel region, including CPU time, process virtual time (time to solution), and information for each thread executing in the region.
- **Block**—Counts the number of times the block was executed.

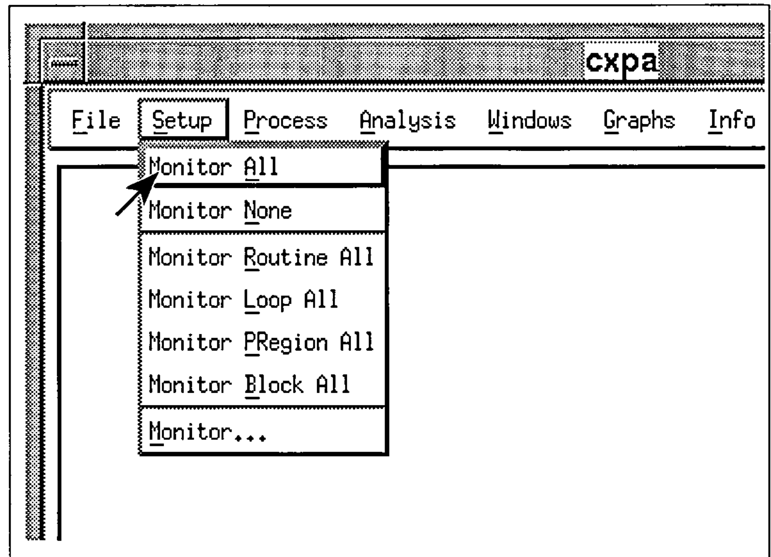
The type of monitor points that exist in your executable depends on the CXpa compiler option you used when compiling the program. If you compiled the program with the `-pa` compiler option, you will have routine, loop, and parallel region monitor points (but not block monitor points).

Initially, all monitor points are disabled. A disabled monitor point is ignored by CXpa during execution, and incurs virtually no overhead.

Before profiling the program, enable the monitor points at the regions you want to profile. You can enable all monitor points if you want to profile the entire program, or a subset of monitor points. As the program runs, CXpa collects performance data at all enabled monitor points in your program.

To enable all monitor points, select the Monitor All option from the Setup Menu, as illustrated in Figure 5.

Figure 5
Enabling all monitor points from the X interface



You can also monitor all regions using the `monitor all` command from the command line.

For more information on monitoring, refer to Chapter 4, "Using monitor points."

Profiling the program

When you run your program under CXpa, CXpa collects profiling data at all enabled monitor points.

This data is collected by CXpa and stored in a binary performance data file (usually referred to as a PDF). By default the PDF is named cxpa.pdf and is located in the directory from which you invoked CXpa.

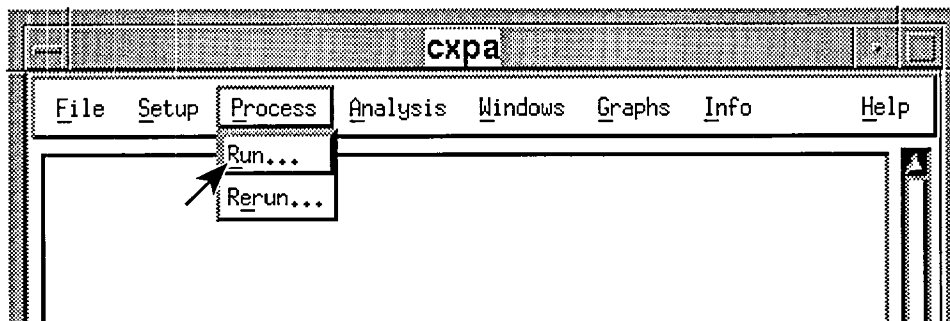
Using the X interface, you control the profiling of your program by using the Control Center dialog box. You can start, pause, continue, and stop profiling using the Control Center.

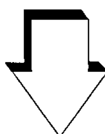
Using the CRT interface, profiling is controlled through the command line. There are commands to start, pause, continue, and stop your program.

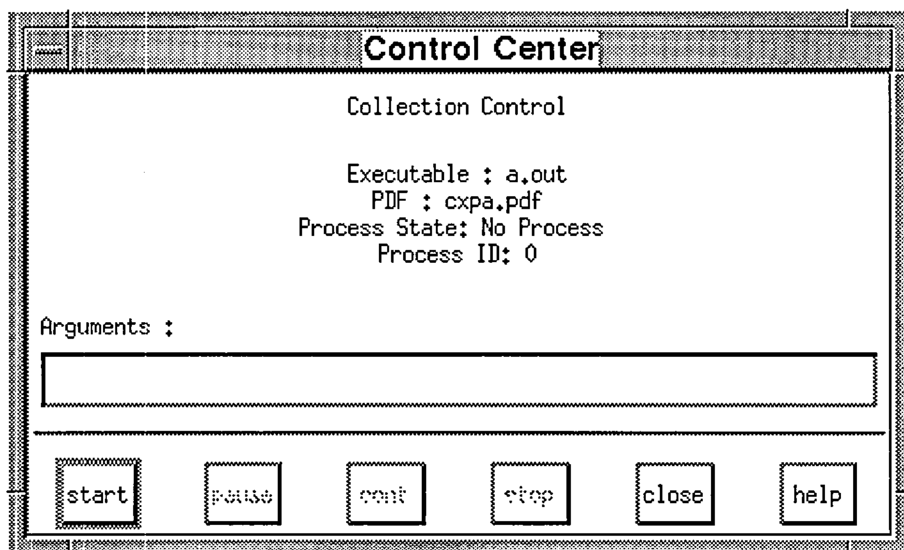
The examples that follow demonstrate how to control profiling using the X interface. For more information on controlling profiling through the command line, refer to Chapter 5, "Controlling profiling."

To open the Control Center dialog box, select the Run option from the Process menu, as shown in Figure 6.

Figure 6
Opening the Control Center dialog box



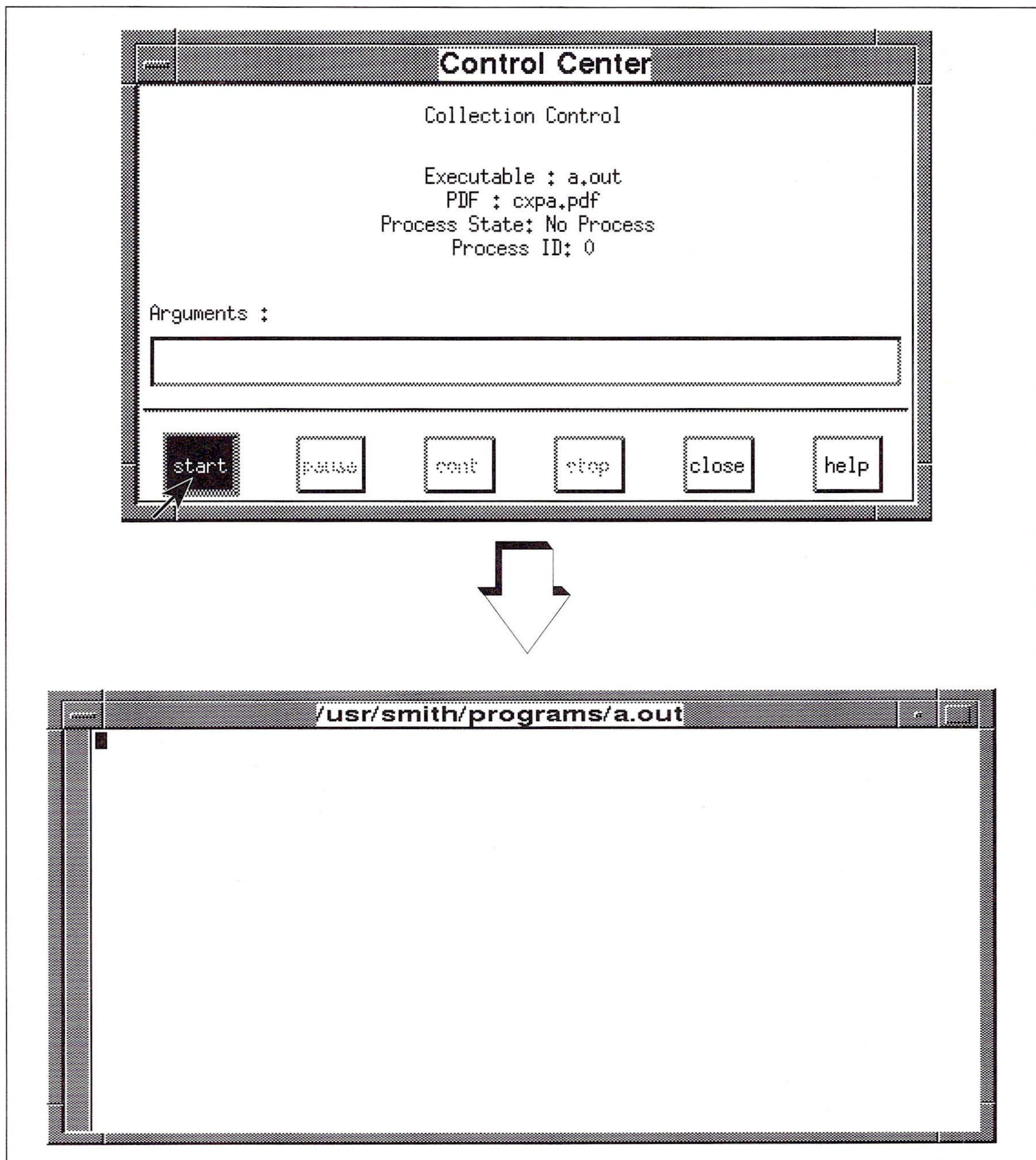




Getting started

To begin program execution and performance data collection, click the start button or use the `run` command from the command line. Figure 7 demonstrates how to begin profiling using the Control Center.

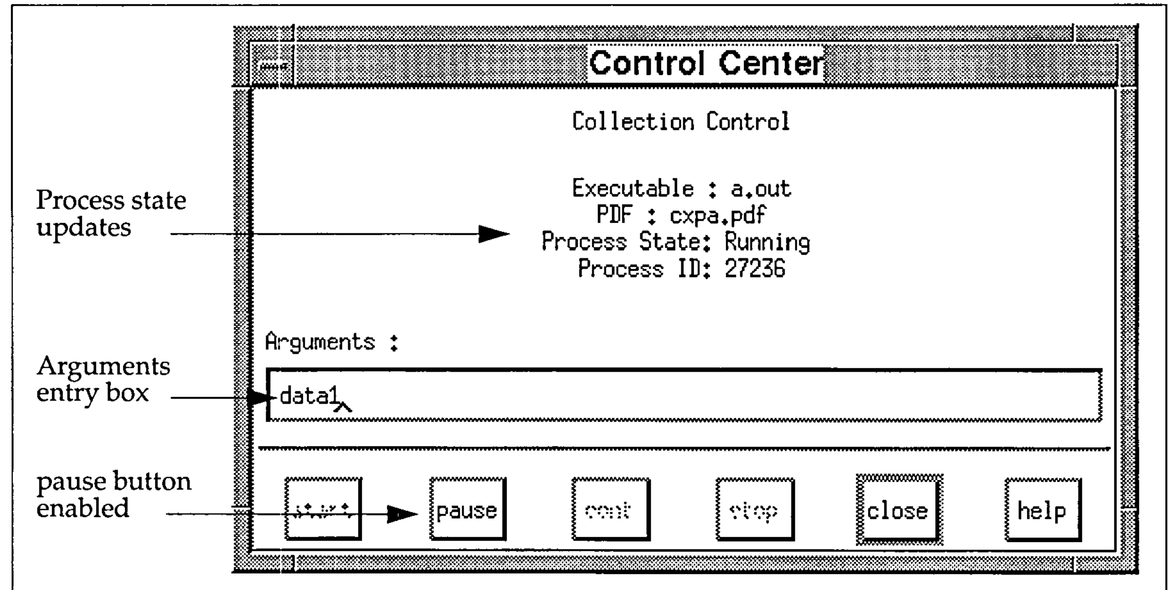
Figure 7
Running the program to begin profiling



After you press the start button, CXpa opens an xterm window for the program's input and output. You can iconify this window if you do not want to view your program's input or output (or you can redirect input and output using shell redirection in the Arguments entry field).

Once the process is running, the Control Center updates to reflect the new process state, as shown in Figure 8.

Figure 8
Control Center with process running



Process execution continues until the program exits, terminates abnormally from a signal, or you click the pause button. If you are using the CRT interface, you can pause execution using CTRL-c. If you are entering commands in the Command Window of the X interface, you can pause execution using the `pause` command.

Pausing profiling enables you to view performance reports based on the collected data, without having to wait for process completion. For more information on pausing profiling, refer to Chapter 5, "Controlling profiling."

When the process is finished executing, CXpa stops data collection and terminates the process. The Control Center dialog box displays the process state as Finished or Terminated. You do not have to close the Control Center dialog box to continue using CXpa.

For more information on profiling your program and controlling process execution, refer to Chapter 5, "Controlling profiling."

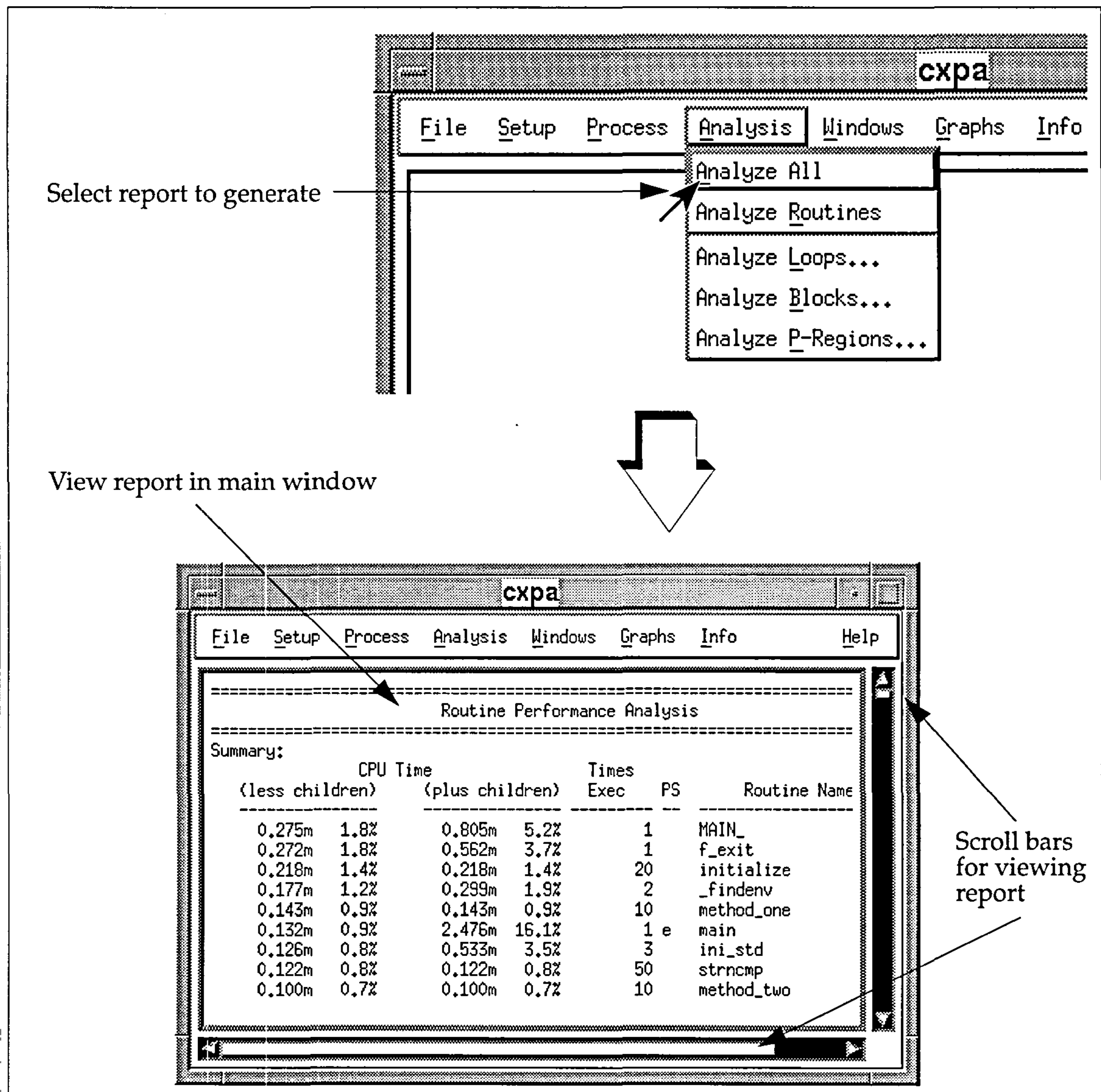
Analyzing performance data

Once profiling has been paused or completed, you can create performance reports. You create these reports by asking CXpa to analyze the performance data collected in the PDF.

You can choose to generate all possible reports from the data, or individual reports. Generating all reports ensures that you get all reports possible from the PDF. Generating a particular report is useful if you are only interested in a particular region (for example, if you only want to look at loop performance).

The example in Figure 9 shows how you can analyze all performance data by selecting the Analyze All option from the Analysis menu. You can also analyze the data in the PDF using the `analyze` command.

Figure 9
Analyzing all performance data



After CXpa analyzes the data, it generates performance reports. The reports appear in the CXpa window of the X interface or are displayed using your pager at the command line.

When viewing the reports through the X interface, use the scroll bars on the side and bottom of the window to browse the report.

Reading performance reports

The performance reports display a summary of the performance data gathered during the execution of your program. Using these reports, you can quickly determine which areas of your program take the most time to execute.

CXpa can generate several reports. CXpa only generates a report if profiling data has been collected for that type of report. The reports are:

- Routine Report
- Loop Report
- Parallel Region Report
- Basic Block Report

Note

CPU times for all reports are expressed in seconds unless annotated with the letter "m" for milliseconds.

Each report is briefly described in the following sections. For full details on interpreting the information contained in each section of the profile report, refer to Chapter 6, "Generating and interpreting reports."

Routine Report

The Routine Report consists of the following tables:

- **Routine Performance Analysis table**—Consists of the following two sections:
  - **Summary**—Provides an overview of the performance of each routine including the number of times the routine was called and the total time spent in each routine, including called routines.
  - **Details**—Lists the minimum, maximum, and average time spent for a single call to the routine.
- **Dynamic Call Graph**—Displays an outline of the routine calls in your program. It can be used to trace the program's execution.

Loop Report

The Loop Report consists of a Loop Performance Analysis table for each executed routine that has monitored loops. Each Loop Performance Analysis table can contain three sections:

- **Summary**—Lists summary information such as the number of times each loop was executed, the iteration counts for each loop, and the CPU time spent in each loop, including time spent in inner loops.
- **Details**—Lists detailed information such as the resulting nesting level of each loop after optimization, the compiler optimizations performed on each loop, and the CPU time spent in each loop.
- **Vectorized loop**—Lists information such as the number of vector spills, number of vector floating point arithmetic instructions, chime count, and estimated number of millions of floating point operations per second (Mflops).

  These measurements can be to evaluate the efficiency of a vectorized loop on that machine. The methods used by CXpa in calculating the data in this table are described in Appendix A, "Interpreting Mflops data."

Getting started

Note

The compiler will not generate the information necessary for profiling loops unless you compile your program at optimization level `-O1` or higher.

Parallel Region Report

The Parallel Region Report provides performance data for monitored parallel regions. There is one Parallel Region Performance Analysis table for each executed routine with a monitored parallel region. The table is divided into two sections:

- **Summary**—Lists the total time spent in each parallel region. The total CPU time, wall-clock time, and process virtual time (PVT) is given. The process virtual time is defined as the total CPU time during which at least one thread is executing in a parallel region.

  CXpa also lists the concurrency factor (CPU time divided by PVT time) for the parallel region. The concurrency factor measures the efficiency of code executing in parallel.
- **Details**—Lists the CPU time, wall-clock time, and chore count of each thread in each parallel region.

A chore is one unit of work executed by a thread in a parallel region, typically a single iteration of a parallelized loop. The chore count is the number of chores a particular thread executed in the region.

Basic Block Report

The Basic Block Report provides performance data for the basic blocks of each monitored routine. A basic block is a section of code that has a single entry point and a single exit point. This report consists of one Basic Block Performance Analysis table for each executed routine with monitored basic blocks.

The Basic Block Performance Analysis table provides the percentage of CPU time spent executing each basic block. This percentage is calculated by dividing total basic blocks executed by a basic block's times executed. The total CPU time of the routine is multiplied by this percentage to give the % CPU time for each basic block.

Note

CXpa can only generate this report if you use the `-pab` option when compiling all routines with basic blocks you wish to monitor.

Getting information on CXpa

During a CXpa session, you may want to get information about the executable being profiled, the PDF being used, or CXpa itself.

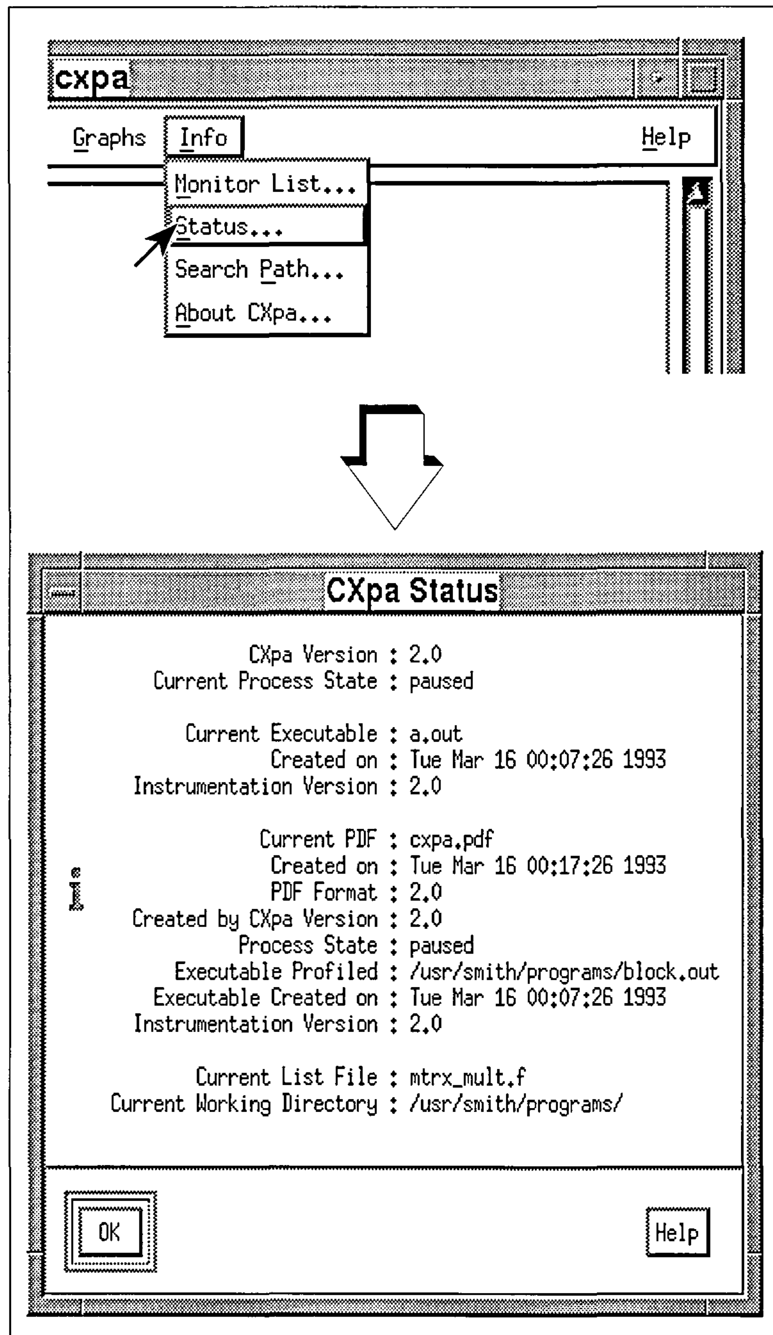
Getting the status of the current CXpa session

CXpa can display the following information about the current session:

- CXpa version number
- Current process state
- Name and creation date of the current executable. The CXpa version for which the executable is instrumented is also given.
- Name and creation date of the PDF. The format and CXpa version that created the PDF is also given.
- Current source file used by the `list` command
- Current working directory

To access this information, use the `info cxpa` command or select the Status option from the Info menu on the CXpa window, as demonstrated in Figure 10.

Figure 10
Getting session status

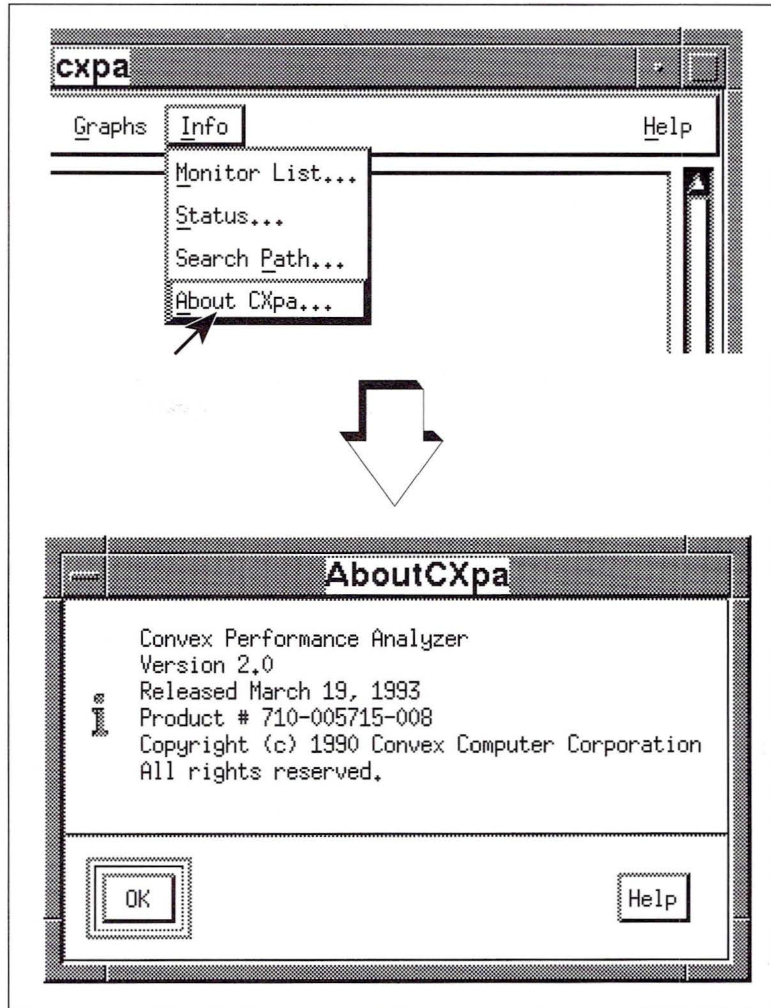


When you select the Status option under the Info menu, the CXpa Status dialog box appears. Click the OK button to close the dialog box.

Getting version information about CXpa

Using the `about cxpa` command or by selecting the About CXpa option under the Info menu, you can display the current version and product number of CXpa, as shown in Figure 11.

Figure 11
Getting CXpa's version information



When you select the About CXpa option from the Info menu, the About CXpa dialog box appears. Click the OK button to close the dialog box.

Getting help online

The CXpa Online Help System contains help pages for a number of CXpa topics, including all CXpa commands, windows, and dialog boxes.

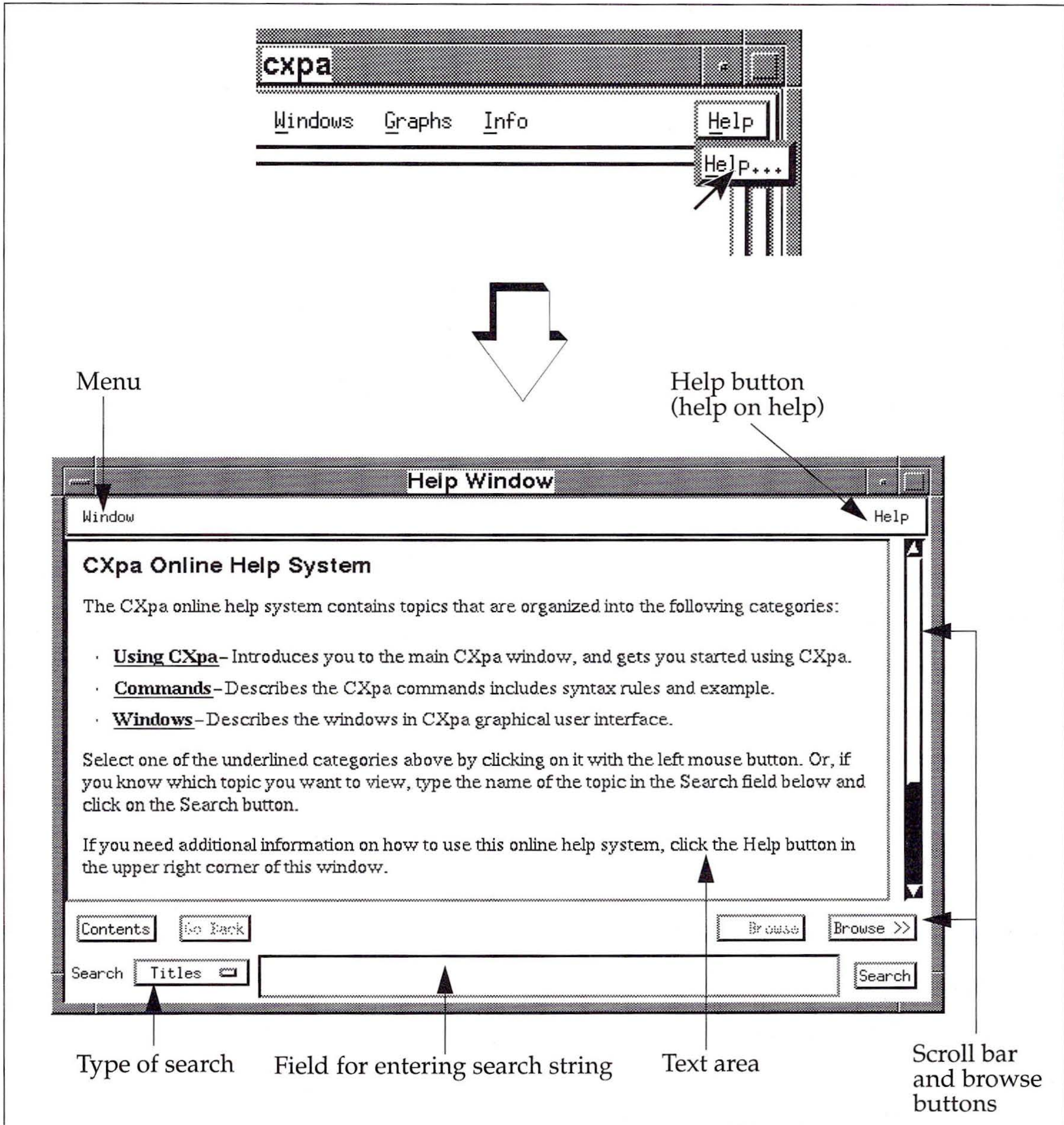
The Help System is available in both the CRT interface and X interface to CXpa. If you access the Help System through a button or menu, the help pages are displayed in the Help Window. If you use the `help` command, help pages are displayed at the command line using your pager.

There are three methods for accessing help:

- Select the Help option from the Help menu in the CXpa main window. This invokes the top level of the CXpa Help System.
- Click the Help button in the bottom right corner of a dialog box to get help on that particular dialog box.
- Use the `help` command at the command line. To get help on a particular topic, follow the `help` command with the topic, such as the name of a CXpa command. For example, to get help on the `analyze` command, use `help analyze`.

To enter the Help System from the X interface, click on the Help button on the CXpa window, as shown in Figure 12.

Figure 12
Accessing the CXpa help system from the main window



The Help Window includes:

- **Window menu**—This pull-down menu allows you to close the Help Window.

- **Help button**—This button displays an explanation of how to use the features of the Help Window.
- **Text area**—This part of the window displays either the text of a help page or a list of topics found by the search.
- **Scroll bar**—This bar enables you to scroll vertically through the current help page using the mouse.
- **Browse buttons**—These buttons let you scroll forward (>>) or backward (<<) through the current help topic. Each click of these buttons moves the text by one screen length. The buttons are deactivated when you reach the end of a topic.
- **Go Back button**—This button displays the previous topic stored in the help history. The help history contains all the topics you viewed during the current session.
- **Contents button**—This button displays the table of contents for the Online Help System.
- **Search field**—This is a text field where you enter a character string to search for. Click on the field with the left mouse button, then type the character string.
- **Type of search**—This pull-down menu lets you select the type of search you want to perform. You can search just the help topic titles or the full text of all help topics.
- **Search button**—This button starts the search for the character string you entered. Activate this button by clicking on it with the mouse or by pressing RETURN when the Help Window is active.

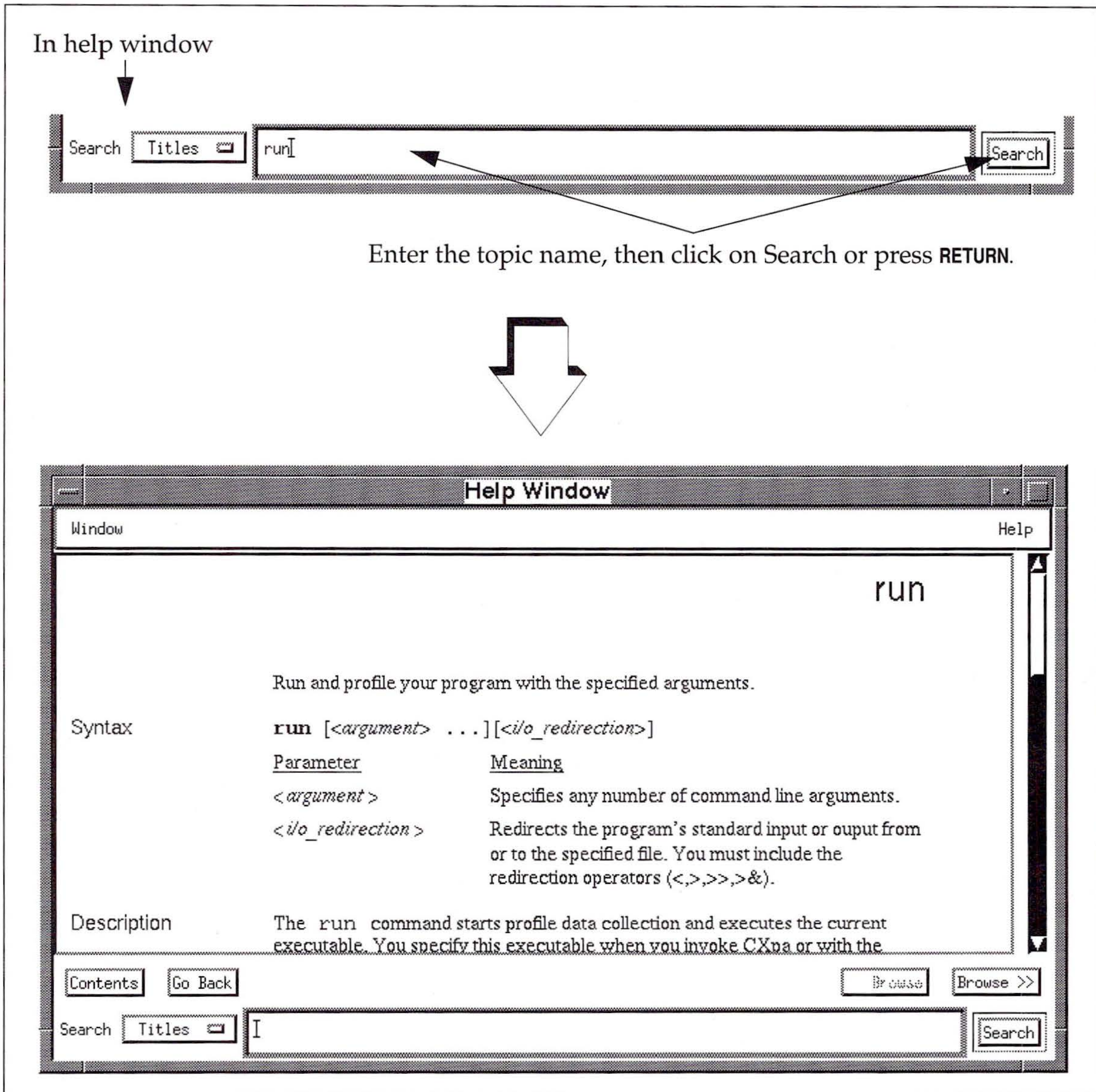
Requesting help on a topic

You can get help on a specific topic by requesting it in one of two ways:

- At the command line, specify the name of the topic with the `help` command.
- In the help window, enter the name of the topic in the search field.

Figure 13 illustrates searching for a help topic using the Help Window.

Figure 13
Requesting help on a topic

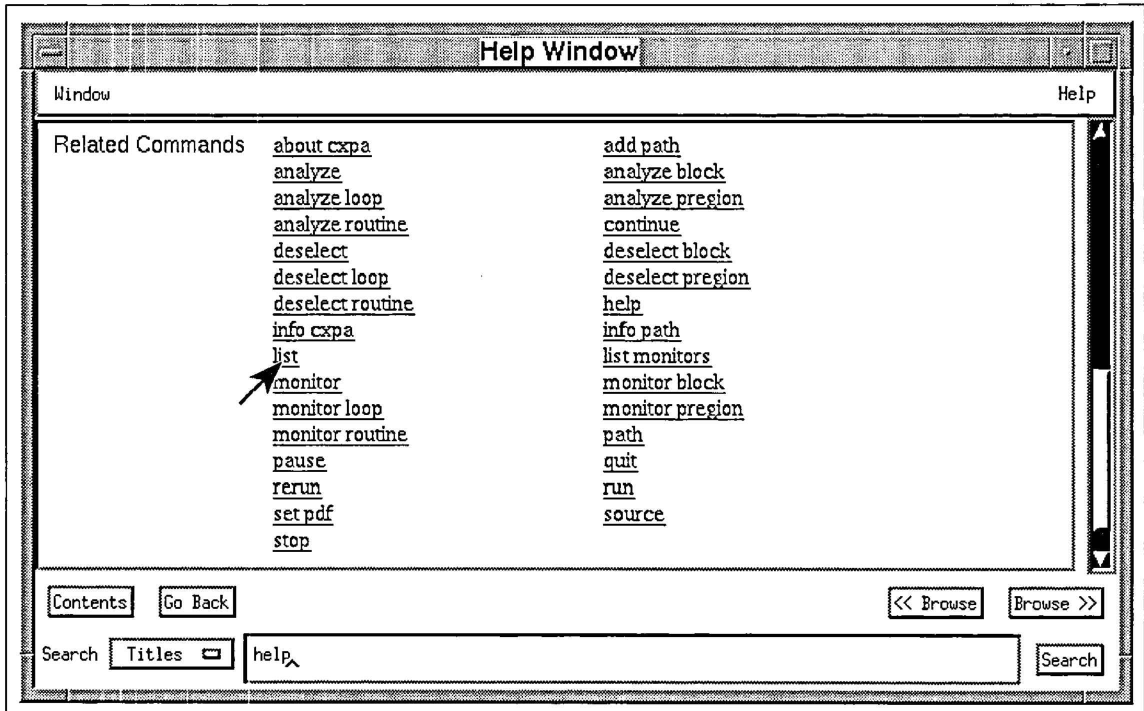


When you click on the Search button, CXpa searches for a help topic whose title contains the specified string. CXpa displays the help page for the specified topic.

Getting help on related topics

At the bottom of each command help page, there is a list of additional commands that are related to the current help topic. To see the list, scroll to the bottom of the help page. To view one of the related topics, click on it with the left mouse button, as shown in Figure 14.

Figure 14
Selecting a related topic



Note

In general, underlined text in the Help Window represents the name of a help topic. To view that topic, simply click on the underlined text with the left mouse button.

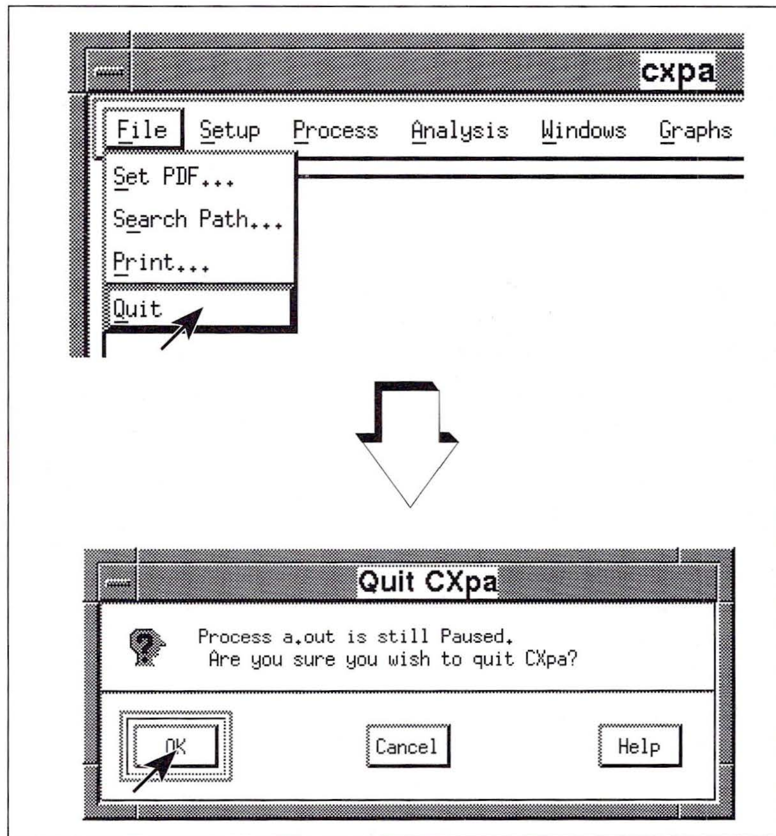
Quitting CXpa

You can quit CXpa using the `quit` command or by selecting the Quit option from the File menu.

If a process is running, CXpa opens the Quit CXpa dialog box asking you if you still wish to quit. If you click the OK button, CXpa stops data collection, closes the open PDF, terminates the process, and exits. If you click the Cancel button, the dialog box closes, and CXpa does not exit.

Figure 15 illustrates how to quit CXpa.

Figure 15
Quitting CXpa



Compiling for profiling

3

This chapter explains how to compile your source files for use with CXpa. You must compile your program with a CXpa compiler option before you can profile it.

You need only compile the source files of your program that you wish to profile. You can also choose what kinds of regions in your program you want to profile, whether routines, loops, parallel regions, or basic blocks.

The topics covered include:

- Compiling a program to run under CXpa
- Compiling limitations
- Language-specific information

Compiling a program to run under CXpa

Before you can profile your program with CXpa, it must be prepared by the compiler for profiling. The compiler adds instructions to the executable file that enable CXpa to gather performance data during execution of your program. Once the compiler has added these instructions, the executable is said to be *instrumented*.

When you compile your program, you specify which types of regions in your program you are interested in profiling. Four different types of regions can be profiled:

- **Routine**—A main routine, functions, or procedures.
- **Loops**—A loop construct (such as the FORTRAN `DO` statement). To profile loops you must compile at optimization level `-O1` or higher. At lower optimization levels, the compiler does not instrument loops.
- **Blocks**—A basic block construct (such as the statements within a loop). A basic block is determined by the compiler, but is always a single-entry, single-exit section of code.
- **Parallel regions**—A section of code (typically a loop) that has been parallelized by the compiler. A parallel region is often called a *p-region*.

The compiler adds instructions around these areas in the executable depending upon the compiler option you choose.

There are three compiler options that instrument an executable for profiling. Each compiler option instruments a different type of program region. The options are described in Table 4.

Table 4 CXpa compiler options.

Compiler option	Enables CXpa to profile
`-pa`	Routines, loops, and parallel regions
`-par`	Routines only
`-pab`	Blocks only

These options may precede or follow any other compiler option except `-p`, `-pb`, and `-pg`, which are designed for use with other profilers.

Only one of these options should be used on a source file when compiling your source code. You can, however, compile different source files with different options. If you do, include a CXpa compiler option when linking them together.

With CXpa, you can select the particular regions to profile. For example, you can profile some or all of the loops that were instrumented by the compiler. Instrumented regions that are not profiled are ignored by CXpa and incur virtually no overhead.

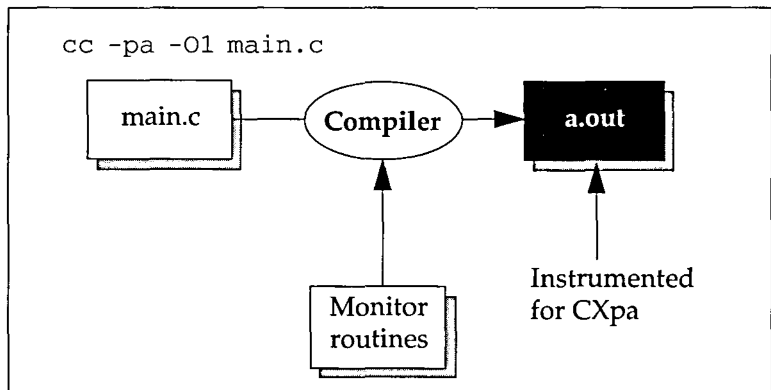
An executable that has been instrumented for use with CXpa can still be executed outside of CXpa. CXpa instrumentation is designed to have a minimal effect on the size and performance of the executable.

Compiling and linking in one step

If you are compiling your source files into an executable with a single call to the compiler, you are compiling and linking in the same step. In this case, object files are not saved, and the executable file is ready to be used by CXpa.

An example of compiling a single source file is shown in Figure 16.

Figure 16
Compiling and linking in one step



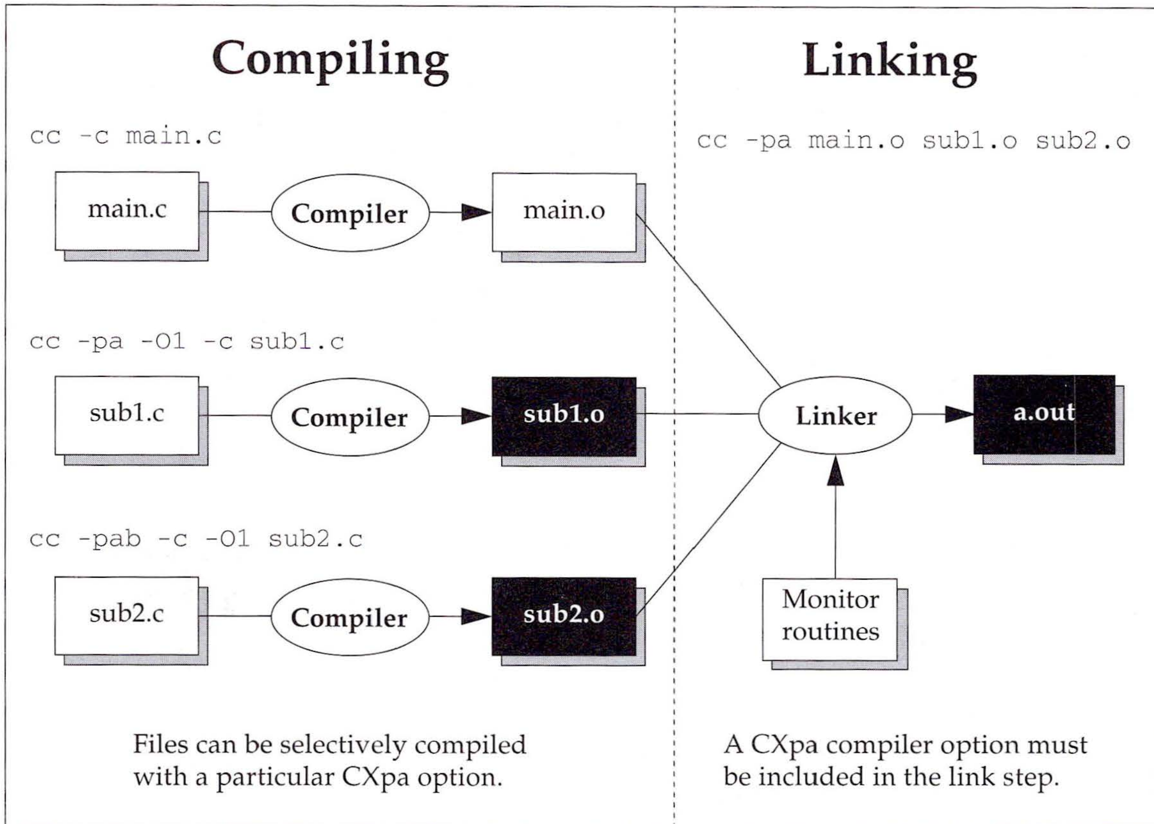
In Figure 16, the source file main.c is compiled at optimization level `-O1` with the compiler option `-pa` to produce the executable `a.out`. Because the `-pa` option was used, the executable has been instrumented for routine and loop profiling.

Compiling and linking separately

Typically, the numerous source files of large programs are compiled separately. Each source file is compiled into an object file using the `-c` compiler option, and then the object files are linked together into an executable.

When compiling for CXpa, you can compile each source file with the same CXpa option, or different profiling options. You must, however, include a single CXpa option when linking. This is demonstrated in Figure 17.

Figure 17
Compiling and linking separately



The program shown in Figure 17 is made up of three source files. The source file main.c is compiled into an object file (because of the `-c` option) without adding any instrumentation for CXpa.

The source file sub1.c is compiled for routine and loop profiling with the `-pa` option, while sub2.c is compiled for block-level profiling with the `-pab` option. Both sub1.c and sub2.c are compiled at optimization level `-O1` to be able to profile loops.

Another call to the compiler invokes the linker, which combines the object files into an executable. The linker also links CXpa's timing routines into the executable. You cannot profile using CXpa unless these routines are linked into your executable.

Special compiling considerations

Consider the following when compiling your source code for use with CXpa:

- If compiling at an optimization level other than -O3, you can link in uninstrumented library routines, rather than having the linker automatically including instrumented the versions of these routines. Doing so removes information on the performance of the routines in that library.
- Profiling instrumented routines that call uninstrumented routines will result in inflated values for the instrumented routine.
- Routines compiled simultaneously with `-pa` and `-pab` can have inflated routine timings.
- Not compiling parallel regions for use with CXpa can result in incorrect timings for the parallel regions themselves, as well as the routines and loops in which they occur.

Each of these considerations is explained in more detail in the sections that follow.

Linking with uninstrumented libraries

When you compile your program with a CXpa compiler option, the linker automatically links in the instrumented versions of the libraries needed by your program. When you profile your program under CXpa, you can track the performance of these routines and separate time spent in library routines from time spent in the calling routine.

However, in some cases, you may not be interested in the time spent in some library routines, typically because you are not interested in modifying the library routine itself. To eliminate these routines from CXpa's reports, you can force the linker to link the standard uninstrumented version of the library rather than the uninstrumented version.

When you do so, the time spent in a routine of that library is included as time spent in the calling routine. This can simplify the reports generated by CXpa (performance data for each called library routine is no longer reported).

Note

CXpa cannot properly profile a program that has uninstrumented libraries and was compiled at optimization level `-O3`. If you are compiling for parallel optimizations, do not link with uninstrumented libraries.

To link the uninstrumented libraries, include the `-L` compiler option when compiling your program. The `-L` option is followed by the name of the library you wish to link, rather than the instrumented library, as shown in Figure 18.

Figure 18
Linking with uninstrumented libraries

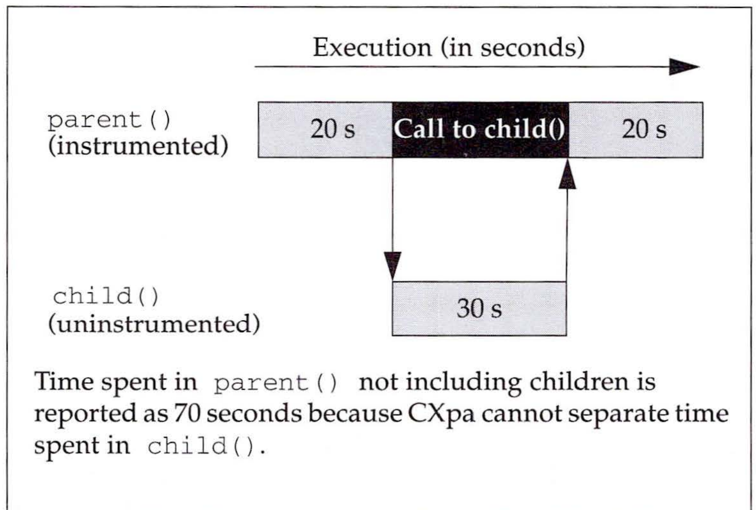
```
% cc -pa -O2 mtrx_mult.c -L/usr/lib/libC.a
```

In Figure 18, the `-L` compiler option links the standard version of the libC.a library with the executable. No performance information will be generated for the routines in this library.

Routines that call uninstrumented routines

If an instrumented routine calls an uninstrumented routine, CXpa will not be able to separate the time spent in the uninstrumented routine from the time spent in the instrumented parent. This condition is illustrated in Figure 19.

Figure 19
Instrumented routine calling an uninstrumented routine



In Figure 19, routine `parent()` has been instrumented for CXpa while routine `child()` has not. If routine `child()` had been compiled with the `-pa` flag, CXpa would have correctly reported `parent()` as having executed for 40 seconds not including children.

Routines compiled with both `-pa` and `-pab`

You should not compile a source file with both the `-pa` and `-pab` options. If you do so, the compiler generates instrumentation for routine measurement and basic block measurement for all routines in the source file.

If you profile a routine that has both sets of instrumentation, the routine times may be longer than expected due to the instrumentation surrounding the blocks. The block timings will be correct.

Parallel regions not compiled for use with CXpa

You should compile all parallel regions of your program for use with CXpa. Because CXpa cannot detect if a parallel region has not been instrumented, CXpa may report incorrect timings.

To protect against parallel regions that are not instrumented, use the `-pa` compiler option when compiling any routines at optimization level `-O3`.

Language-specific information

CXpa can be used with Ada, C, and FORTRAN programs compiled with the CXpa compiler options of the CONVEX compilers. There are some differences between these languages when compiling for CXpa. These differences are detailed below.

Compiling FORTRAN source code for CXpa

The `-pa` and `-pab` options cannot be used with source files containing a FORTRAN `OPTIONS` statement. If you need to use the `OPTIONS` statement for a particular section of code, do not compile that source file for use with CXpa.

You can still profile the program, but no data will be collected for the routines in that source file.

Compiling Ada source code for use with CXpa

There are several considerations when compiling an Ada application or library for use with CXpa:

- Ada libraries can be instrumented for use with CXpa by including the `-pa` or `-pab` option on the `a.mklib` command line:

a.mklib {-pa | -pab} [*options*] [*new_library* [*parent_library*]]

- Ada executables generated by `a.make` can be instrumented for use with CXpa by including a CXpa compiler option on the `a.make` command line. This is demonstrated in Figure 20.

Figure 20
Compiling for CXpa using `a.make`

```
% a.make -pa -O1 mtrx_mult
```

- Ada applications that call `_exit()`, should instead call `_ADA_EXIT()`. Calling `_exit()` exits the Ada run-time environment prematurely, leaving CXpa in an unpredictable state.

Using monitor points

4

This chapter explains what monitor points are and how to enable and disable them. Using monitor points, you can control where in the program CXpa collects performance data, and the type of performance data collected.

CXpa collects performance data at the enabled monitor points in your program. The more monitor points you have enabled, the more performance data you collect, and the longer it takes for profiling to complete.

The chapter also explains how to list the monitor points in your program. This enables you to determine which areas of your program can be profiled as well as the current status of each monitor point.

The topics covered in this chapter include:

- Understanding monitor points
- Choosing the monitor points to enable
- Enabling monitor points using the X interface
- Enabling monitor points using the command line
- Disabling monitor points
- Listing monitor points

Understanding monitor points

Before you run your program under CXpa, you must specify the regions of your program that you want to profile. You do this by enabling the monitor points at these regions.

When you compile your program for use with CXpa, the compiler inserts special instructions in the executable around the routines, loops, blocks, and parallel regions of your program. A region with these instructions is said to be instrumented.

A monitor point is the set of instructions used to monitor a particular region. The type of region that monitor point monitors determines its type. There are four types of monitor points:

- **Routine**—Monitors CPU time spent inside of the routine. Also counts the number of times a routine is called.
- **Loop**—Monitors CPU time spent inside of the loop, and the minimum, maximum, and average iteration count. Loop monitor points will not exist in your program unless it was compiled at optimization level `-O1` or above.
- **Parallel region**—Monitors CPU time spent inside of the parallel region, including measurements for each thread executing inside the region.
- **Block**—Monitors the number of times the basic block was executed.

Note

Enabled routine monitor points collect only routine measurements. To profile loops, parallel regions, or basic blocks inside of a routine, you must enable the corresponding loop, parallel region, or block monitor points.

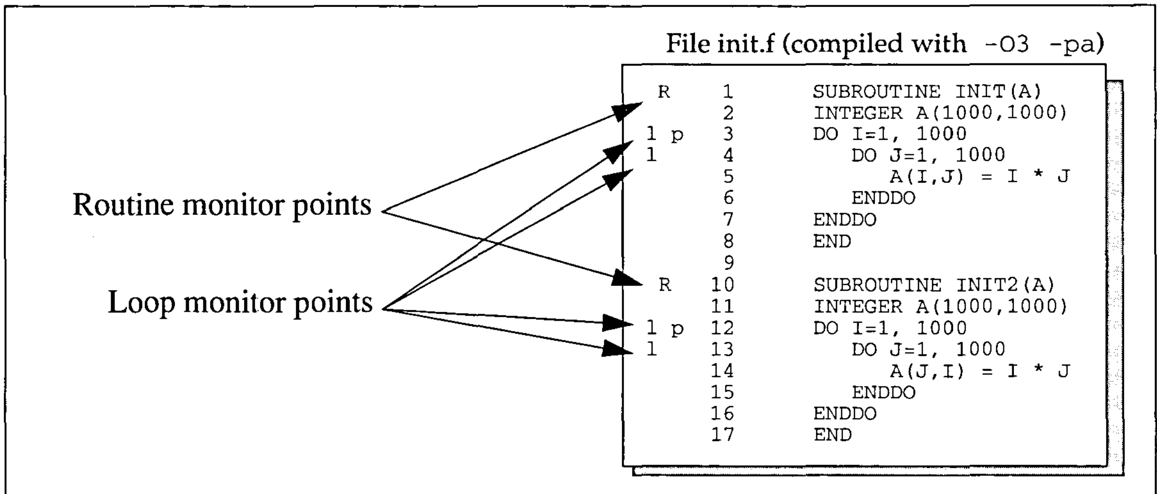
CXpa relates monitor points to source code, making it easier to identify the area being monitored. When listing monitor points with CXpa, your source code is displayed with special annotations, as shown in Table 5.

Table 5 Monitor point annotations in source listings

Type	Enabled	Disabled
Routine	`R`	`r`
Loop	`L`	`l`
Parallel region	`P`	`p`
Block	`B`	`b`

Figure 21 shows a sample routine with several types of monitor points.

Figure 21
Example of routine and loop monitor points



Each routine in Figure 21 has a routine monitor point. Each loop has a loop monitor point, and each parallel region has a parallel region monitor point.

Initially all monitor points in your executable are disabled, and the instructions of the monitor points do *not* call the timing routines. If you run the program without any enabled monitor points, no performance data is collected.

When you enable a monitor point, CXpa changes the instructions to call the timing routines. This enables CXpa to collect performance data for that region.

When you run your program, CXpa collects performance data at every enabled monitor point whose region is executed. You can control the data that is collected by controlling which monitor points are enabled.

Choosing the monitor points to enable

You can enable all of the monitor points in your program, or a subset. You do not have to recompile your program to enable or disable monitor points.

Typically, you will want to enable all monitor points the first time you profile your program. This provides a complete picture of your program's performance. Using this information, you can then identify the regions that take the longest to execute.

After you have identified the regions of your program whose performance you want improve, you may want to enable the monitor points at just these regions. There are two benefits to enabling a subset of monitor points:

- **Profiling is quicker**—With fewer enabled monitor points, less time is spent in the timing routines CXpa uses to collect performance data. Enabled monitor points do not affect CPU time, but they do slow down wall-clock time. The more enabled monitor points in your program, the longer it takes to profile.
- **Generated reports are more focused**—By removing extra performance data from the reports, you can quickly compare the performance of the regions you are interested in.

In addition, if you want to profile only parallel regions, enable only parallel region monitor points. This provides a more accurate representation of the performance of parallel regions containing nested loops. If CXpa profiles a parallel region *and* a nested loop within that region, the time spent monitoring the nested loop affects the parallel virtual time of the parallel region.

CXpa provides functionality to enable a wide range of monitor points. You can enable and disable:

- Monitor points in all routines
- Monitor points in specific routines
- Monitor points at specific lines (from command line only)

You can also choose to enable or disable all types of monitor points (routine, loop, parallel region, and block) or only specific types (for example, only parallel regions).

Note

When you enable monitor points in all routines, you are also enabling monitor points in any instrumented library routines that are included in your executable. You can prevent this by linking in uninstrumented libraries with your executable. For more information, refer to "Linking with uninstrumented libraries," on page 37.

Enabling monitor points from the X interface

Using the X interface, you can:

- Enable all monitor point types in all routines
- Enable one monitor point type in all routines
- Enable all monitor point types in select routines
- Enable one monitor point type in select routines

Each of these methods is described in the sections that follow.

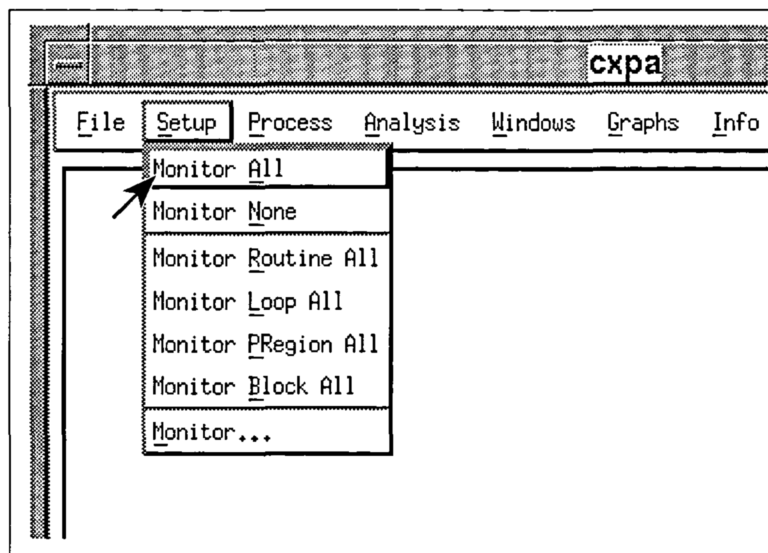
Enabling all monitor point types in all routines

Enabling all the monitor points in your program is both the simplest and most comprehensive method of enabling monitor points.

It is a good idea to enable all monitor points the first time you profile an application. This ensures that you will collect performance data at all available regions of your program.

To enable every monitor point in your program, use the Monitor All option from the Setup menu, as demonstrated in Figure 22.

Figure 22
Using the Monitor All option

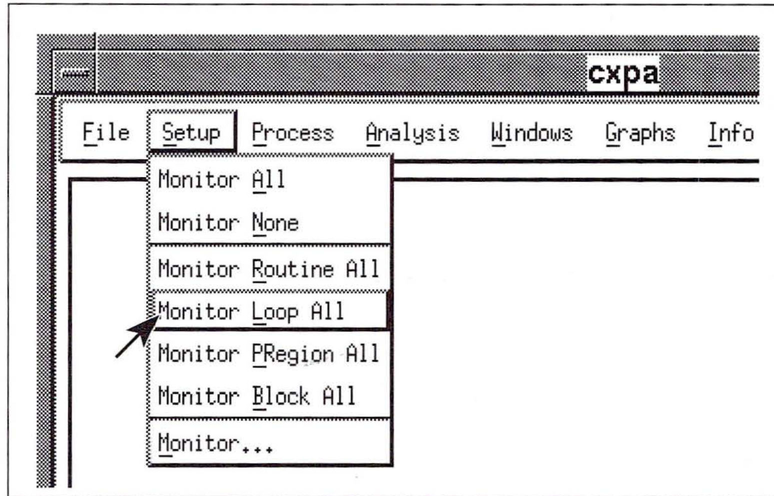


Enabling one monitor point type in all routines

You can also enable all monitor points of a particular type. For example, if you want to only profile the loops in your program, you could enable all loop monitor points, without enabling routines or parallel regions.

To enable all monitor points of a particular type, use the corresponding Monitor option from the Setup menu. This is shown in Figure 23.

Figure 23
Enabling all monitor points of a particular type



Enabling all monitor point types in select routines

You may want to enable monitor points in only selected routines, rather than in all routines. This shortens the time spent profiling because less time is spent in the timing routines.

To enable monitor points in specific routines, you use the Set Monitor Points dialog box. The steps to use this book are briefly described below. For a detailed description of this dialog box, refer to "Using the Set Monitor Points dialog box," on page 48.

Step 1 Open the Set Monitor Points dialog box using the Monitor... option from the Setup menu.

Step 2 Click the all radio button in the Monitor Point Type column.

Step 3 Select the routines whose monitor points you want to enable by clicking on them in the Disabled List.

Step 4 Move the routines to the Enabled List by clicking the Move arrow in between the two lists. The routines now appear in the Enabled List.

Step 5 Apply these settings by clicking the OK button. The monitor points are enabled, and the dialog box closes.

Enabling one monitor point type in select routines

Enabling a particular monitor point type in selected routine follows the same procedure as enabling all monitor points for a routine. Using the Set Monitor Points dialog box, select the type of monitor point to enable and move the routines from the Disabled List to the Enabled List.

The steps for using the Set Monitor Points dialog box are briefly described below. For a detailed description of this dialog box, refer to "Using the Set Monitor Points dialog box," on page 48.

Step 1 Open the Set Monitor Points dialog box using the Monitor... option from the Setup menu.

Step 2 Click the radio button for the correct monitor point type in the Monitor Point Type column.

Step 3 Select the routines whose monitor points you want to enable by clicking on them in the Disabled List.

Step 4 Move the routines to the Enabled List by clicking the Move arrow in between the two lists. The routines now appear in the Enabled List.

Step 5 Apply these settings by clicking the OK button. The monitor points are enabled, and the dialog box closes.

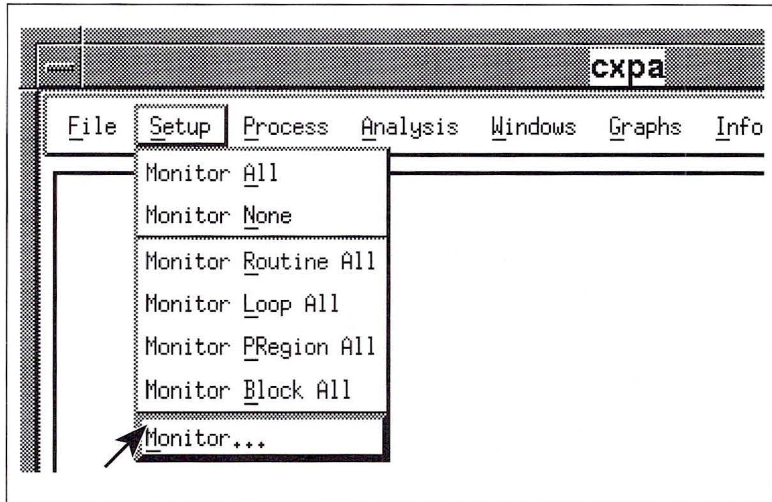
Using the Set Monitor Points dialog box

When using the X interface, you enable monitor points in specific routines using the Set Monitor Points dialog box. The sections below explain how to use this dialog box.

Opening the Set Monitor Points dialog box

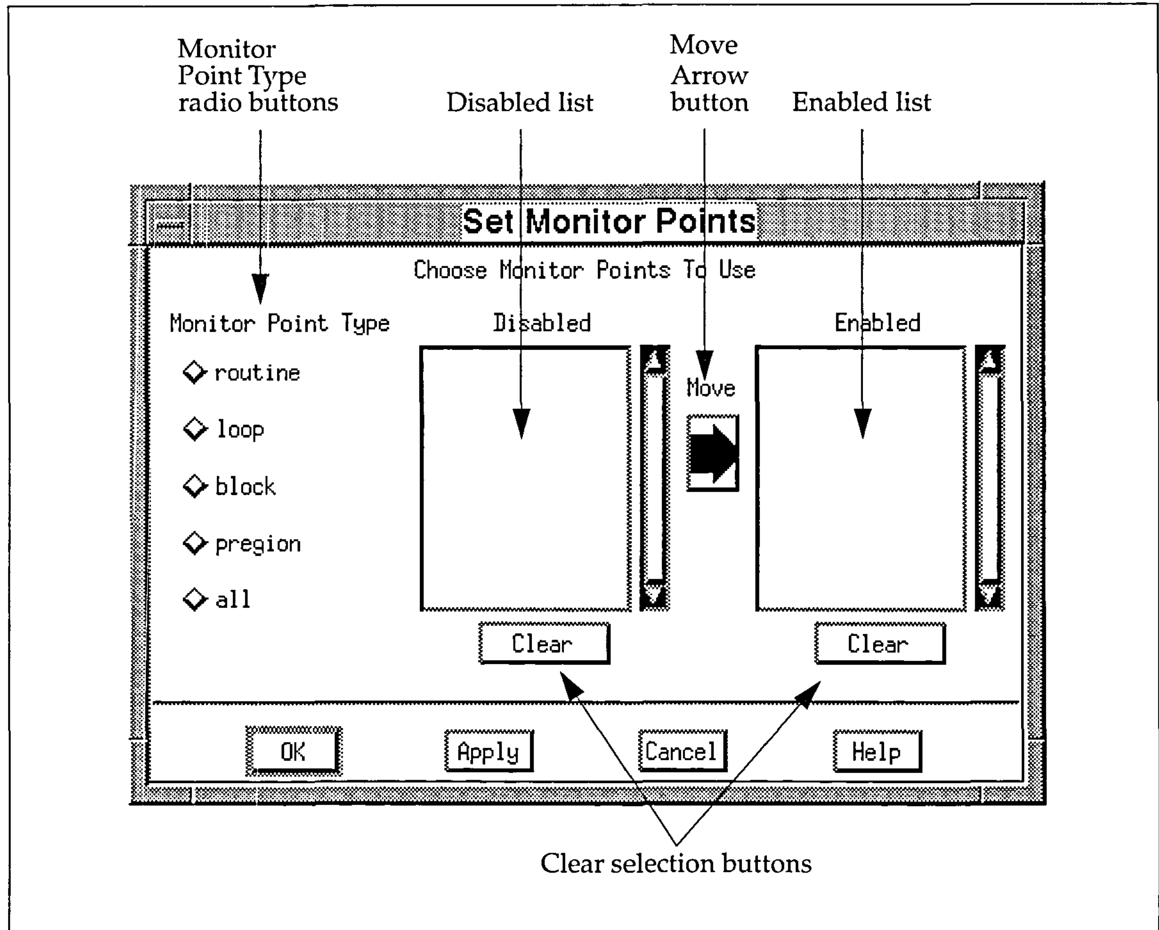
Open the Set Monitor Points dialog box using the Monitor... option in the Setup menu, as shown in Figure 24.

Figure 24
Opening the Set Monitor Points dialog box



When you do so, the Set Monitor Points dialog box appears, as shown in Figure 25.

Figure 25
Set Monitor Points dialog box



The parts of the Set Monitor Points dialog box are described below:

- **Monitor Point Type radio buttons**—Select the type of monitor points to enable or disable. There is a radio button for each of the four monitor point types. The all radio button represents all monitor point types. Only one monitor point type may be selected at a time.
- **Disabled list**—Lists the names of all routines that have disabled monitor points of the type selected in the Monitor Point Type radio box. Initially, all monitor points are disabled.

- **Enabled list**—Lists the names of all routines that have enabled monitor points of the type selected in the Monitor Point Type radio box.
- **Clear selection button**—Deselects all routines from the list above it. This makes it easy to deselect all the routines in the list if you select them all and then change your mind.
- **Move Arrow button**—Moves selected routines from one list to the other. The arrow changes directions according to which list has selected items.

The buttons at the bottom of the Set Monitor Points dialog box are described in Table 6.

Table 6 Set Monitor Points dialog box buttons

Button name	Description
OK	Applies the current set of enabled monitor point, then closes the dialog box.
Apply	Applies the current set of enabled monitor points, but does not close the dialog box.
Cancel	Closes the dialog box (without applying the current set of enabled monitor points).
Help	Displays the help page for this dialog box.

Initially, a Monitor Point Type radio button is not selected, so both lists are empty.

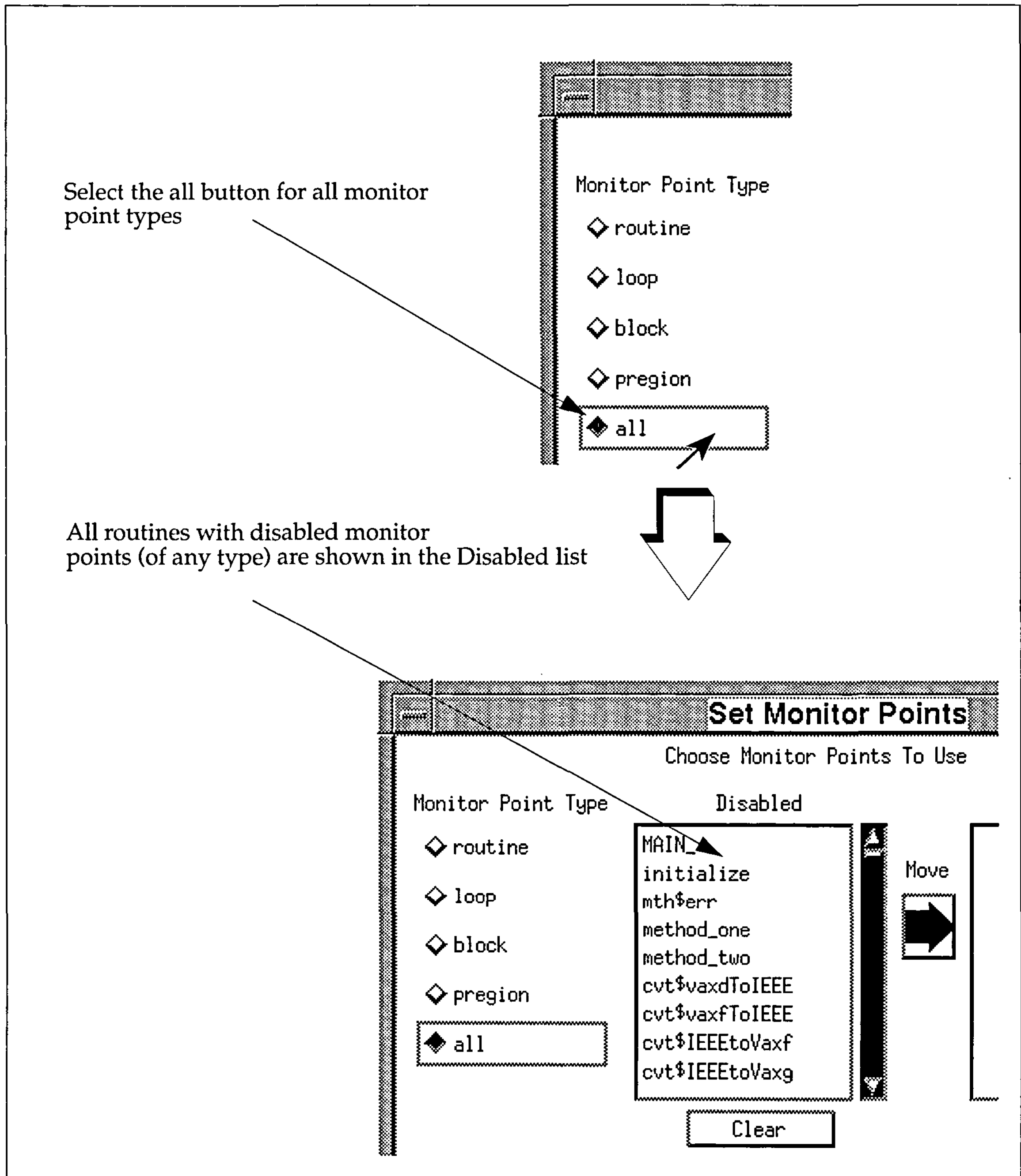
Selecting the monitor point type

Before enabling the monitor points of particular routines, you must select the type of monitor point you want to enable.

If there are no monitor points of the specified type in your program, both lists appear empty. The type of monitor points in your program depends on the CXpa compiler option used when compiling.

To enable all monitor points in a routine, click the left mouse button on the all radio button at the Monitor Point Type column, as shown in Figure 26.

Figure 26
Selecting all types of monitor points



Using monitor points

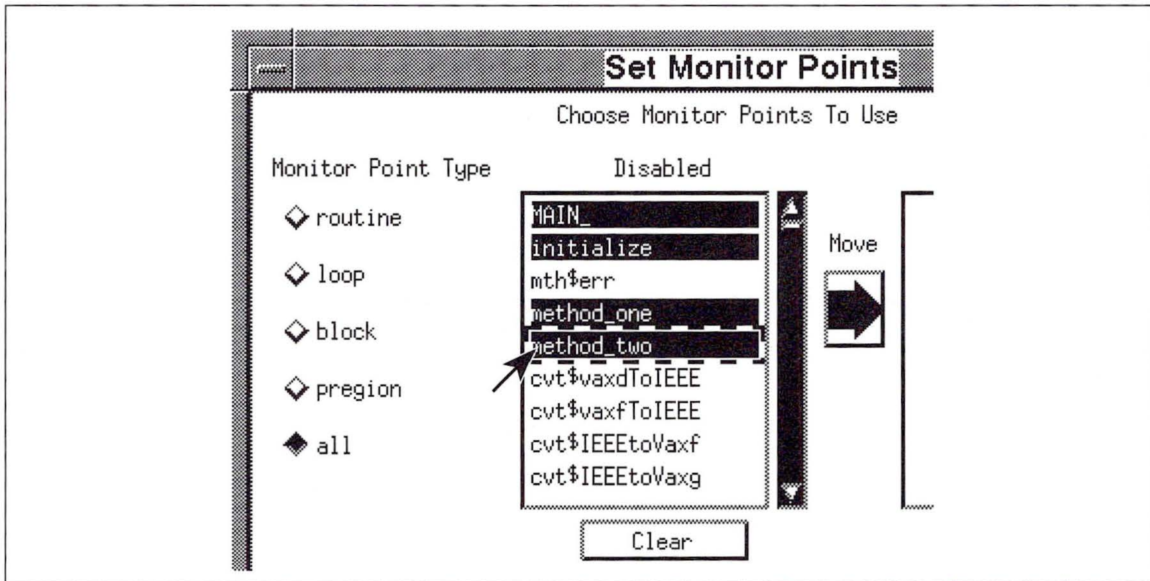
Selecting routines from the Disabled or Enabled list

Once you have chosen the type of monitor point, you can begin selecting the routines whose monitor points you want to enable.

Each list displays the names of all the routines in your program with monitor points of the particular type. Because this includes any instrumented library routines, the list may include more routines than in the program itself. Scroll the list using the scroll bar.

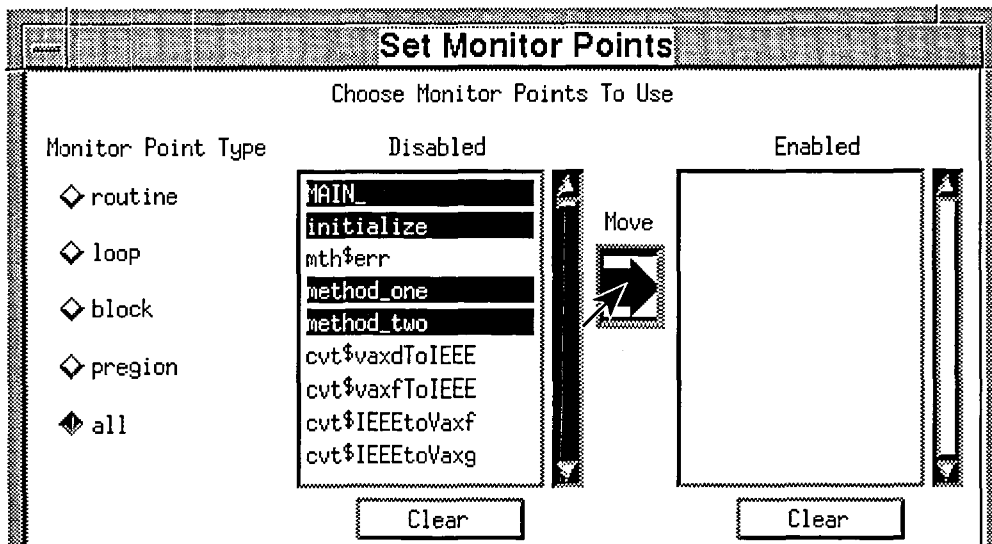
To select a routine, click the left mouse button on the routine name. You can select multiple routines at a time. Figure 27 shows an example of selecting multiple routines.

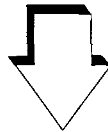
Figure 27
Selecting multiple routines

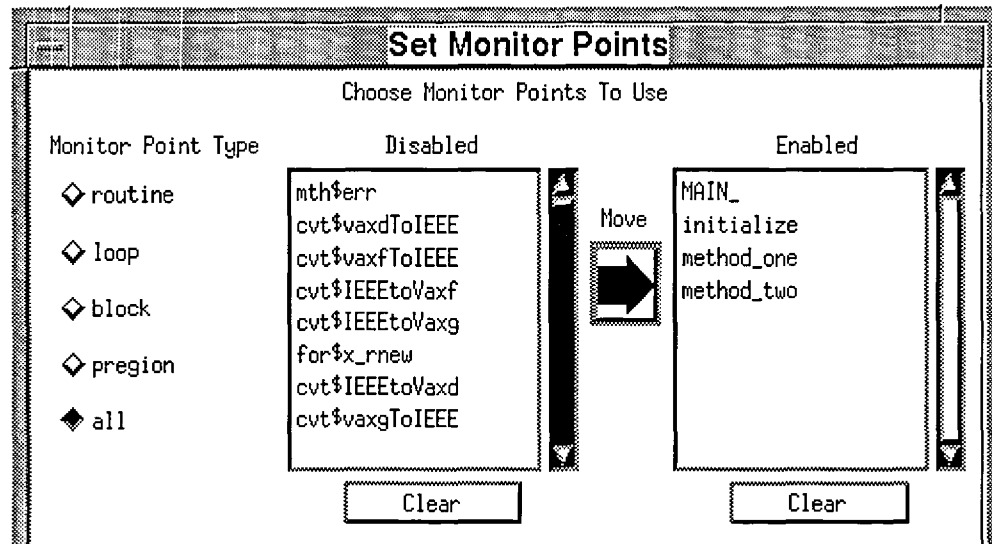


Click the Move Arrow button to move the selected routines from one list to the other, as demonstrated in Figure 28.

Figure 28
Moving selected routines between lists







Using monitor points

After clicking the Move Arrow button, the highlighted routines are moved from the Disabled list to the Enabled list.

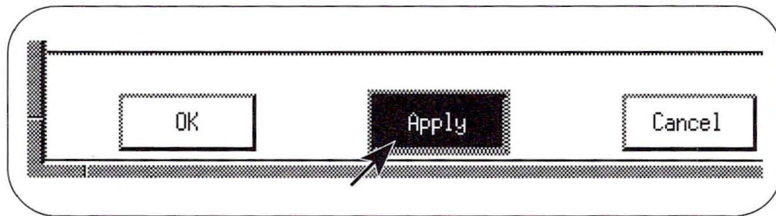
Applying changes

Once you have selected the monitor points you want enabled, you must apply the changes. Applying the changes causes CXpa to update the current set of enabled monitor points.

There are two ways to apply the changes you have made. If you want to continue enabling and disabling monitor points, click the Apply button at the bottom of the dialog box. If you are done making changes, you can apply the changes and close the dialog box by selecting the OK button.

Figure 29 illustrates how to apply the current changes to the set of enabled monitor points without closing the dialog box.

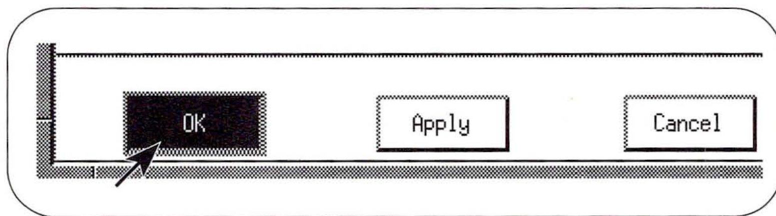
Figure 29
Applying the current monitor point setttings



Closing the Set Monitor Points dialog box

To close the dialog box, click the OK button at the bottom of the window, as shown in Figure 30.

Figure 30
Closing the Set Monitor Points dialog box



The OK button applies the current list of enabled monitor points, then closes the window.

Note

If you do not want to apply the current list of enabled monitor points, you can close the dialog box using the Cancel button.

Enabling monitor points using the command line

From the CXpa command line, you can enable any set of monitor points in your program. You can:

- Enable all monitor point types in all routines
- Enable one monitor point type in all routines
- Enable all monitor point types in select routines
- Enable one monitor point type in select routines
- Enable all monitor point types at select lines
- Enable one monitor point type at select lines

Selecting specific routines to monitor, rather than all routines, provides greater control over profiling and shortens profiling time because less data is being collected.

Each of these methods is described in detail in the sections that follow. The same source code is used in all figures to contrast different command line options.

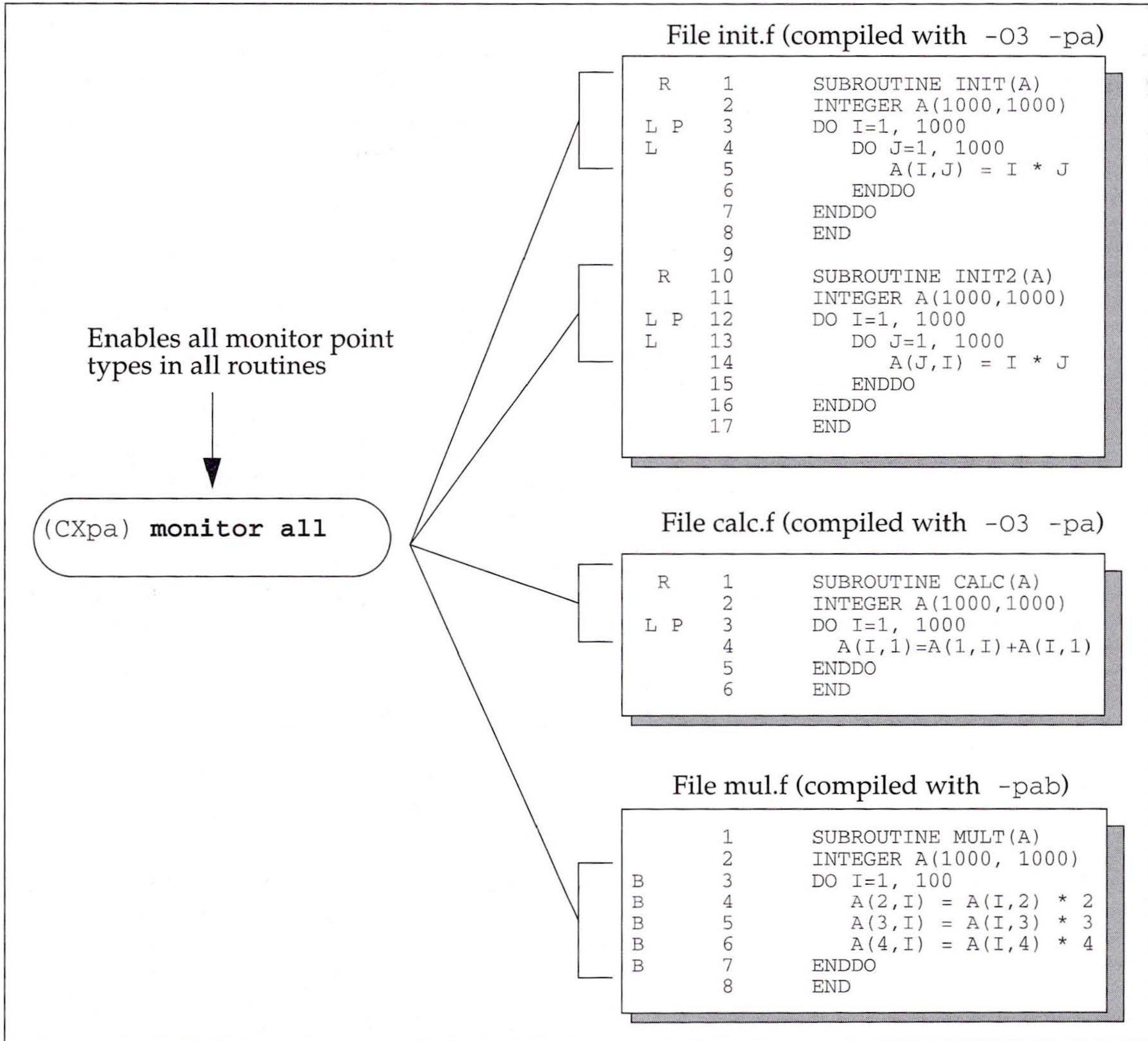
Enabling all monitor point types in all routines

Enabling all the monitor points in your program is both the simplest and most comprehensive method of specifying monitor points.

It is a good idea to enable all monitor points the first time you profile an application. This ensures that you will collect performance data at all available regions of your program.

Figure 31 shows how to enable all monitor points of all types throughout the program.

Figure 31
Enabling all monitor point types in all routines



As shown in Figure 31, every monitor point in the program is enabled, regardless of type.

Enabling one monitor point type in all routines

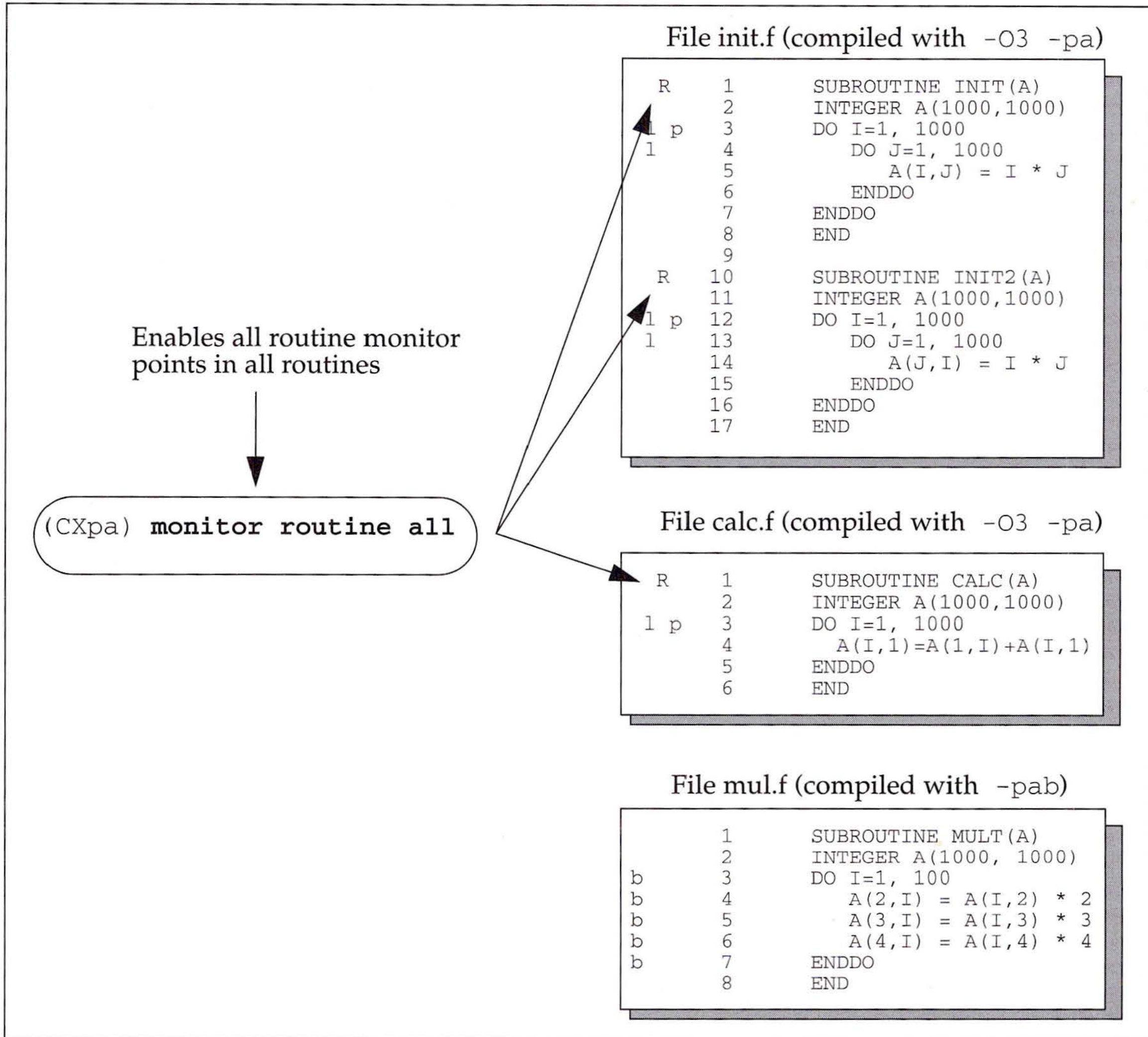
To enable the monitor points of a given type, use one of the commands in Table 7.

Table 7 Commands for enabling monitor points in all routines

Command	Monitor points enabled
`monitor routine all`	All routine monitor points
`monitor loop all`	All loop monitor points
`monitor pregion all`	All p-region monitor points
`monitor block all`	All block monitor points

In Figure 32, the `monitor routine all` command enables all routine monitor points throughout the program. Loop, parallel region, and block monitor points are not enabled.

Figure 32
Enabling all routine monitor points in all routines



Enabling all monitor point types in select routines

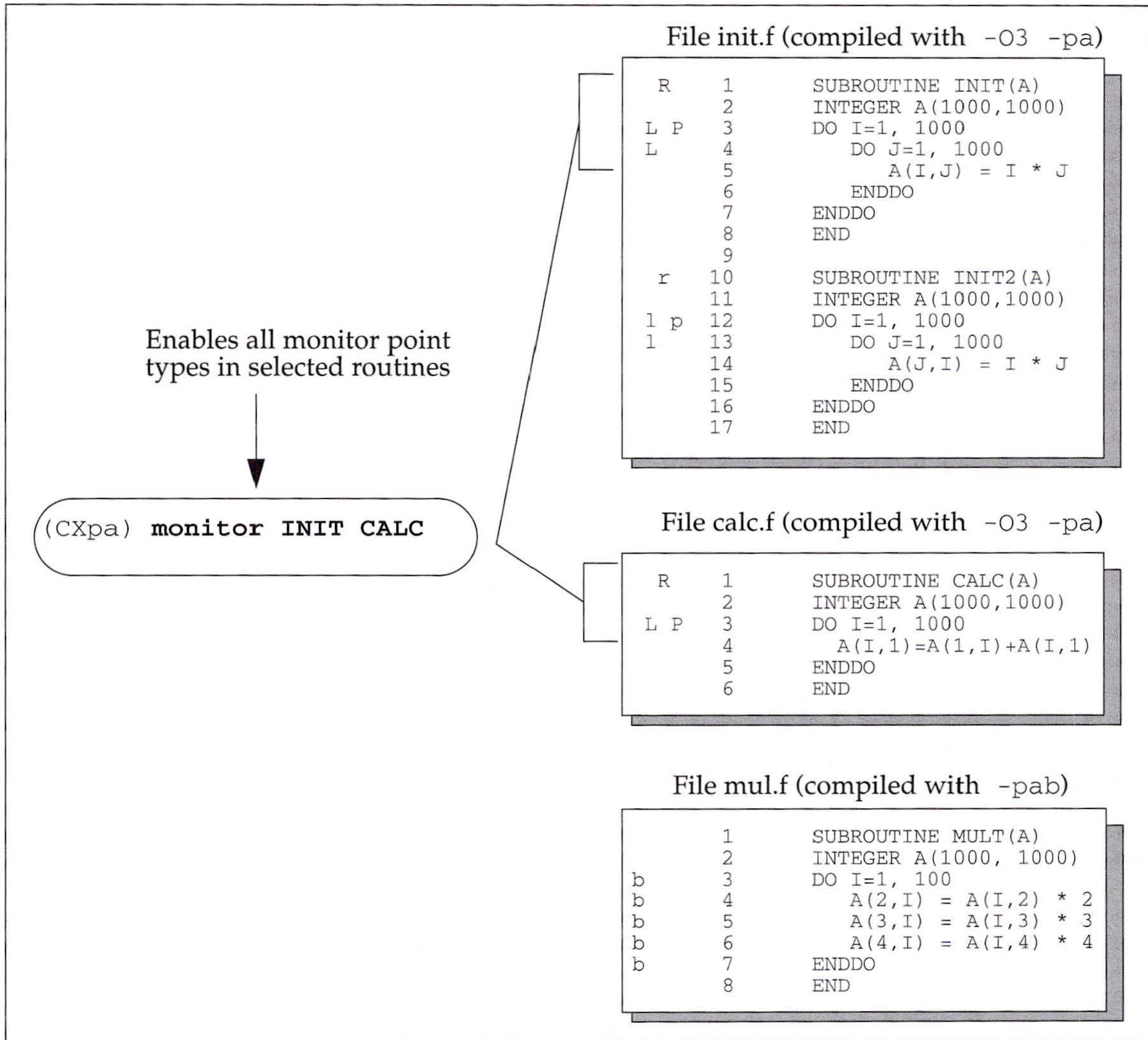
After profiling all routines in your program, you may want to profile only particular routines. Typically, these are the routines whose performance you want to improve.

Use the `monitor` command to enable all monitor points in a routine. Follow the `monitor` command with the name of each routine whose monitor points you want to enable. Multiple routine names are separated by spaces.

Using monitor points

Figure 33 illustrates enabling all monitor points in selected routines.

Figure 33
Enabling all monitor point types in selected routines



Enabling one monitor point type in select routines

You can enable one monitor point type in select routines using one of the commands in Table 8.

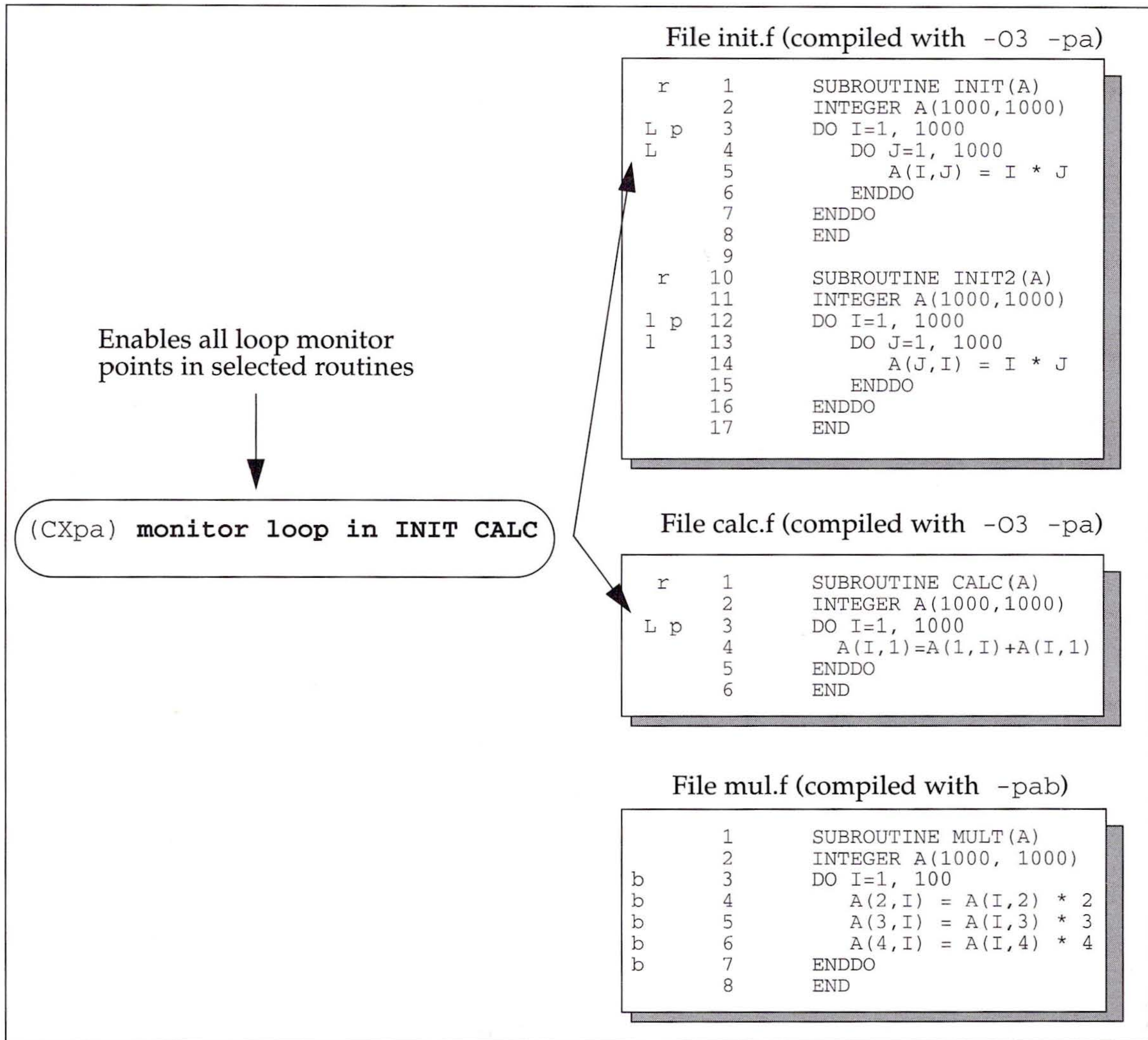
Table 8 Commands for enabling monitor points in routines

Command	Monitor points enabled
`monitor loop in`	All loop monitor points in a routine
`monitor pregion in`	All parallel region monitor points in a routine
`monitor block in`	All block monitor points in a routine

Each of these commands enables every monitor point of that type in the specified routine (or routines). Multiple routines are separated by spaces on the command line.

In Figure 34, the `monitor loop in` command enables all loop monitor points in the `INIT` and `CALC` routines. No other monitor points are enabled.

Figure 34
Enabling all loop monitor points in selected routines

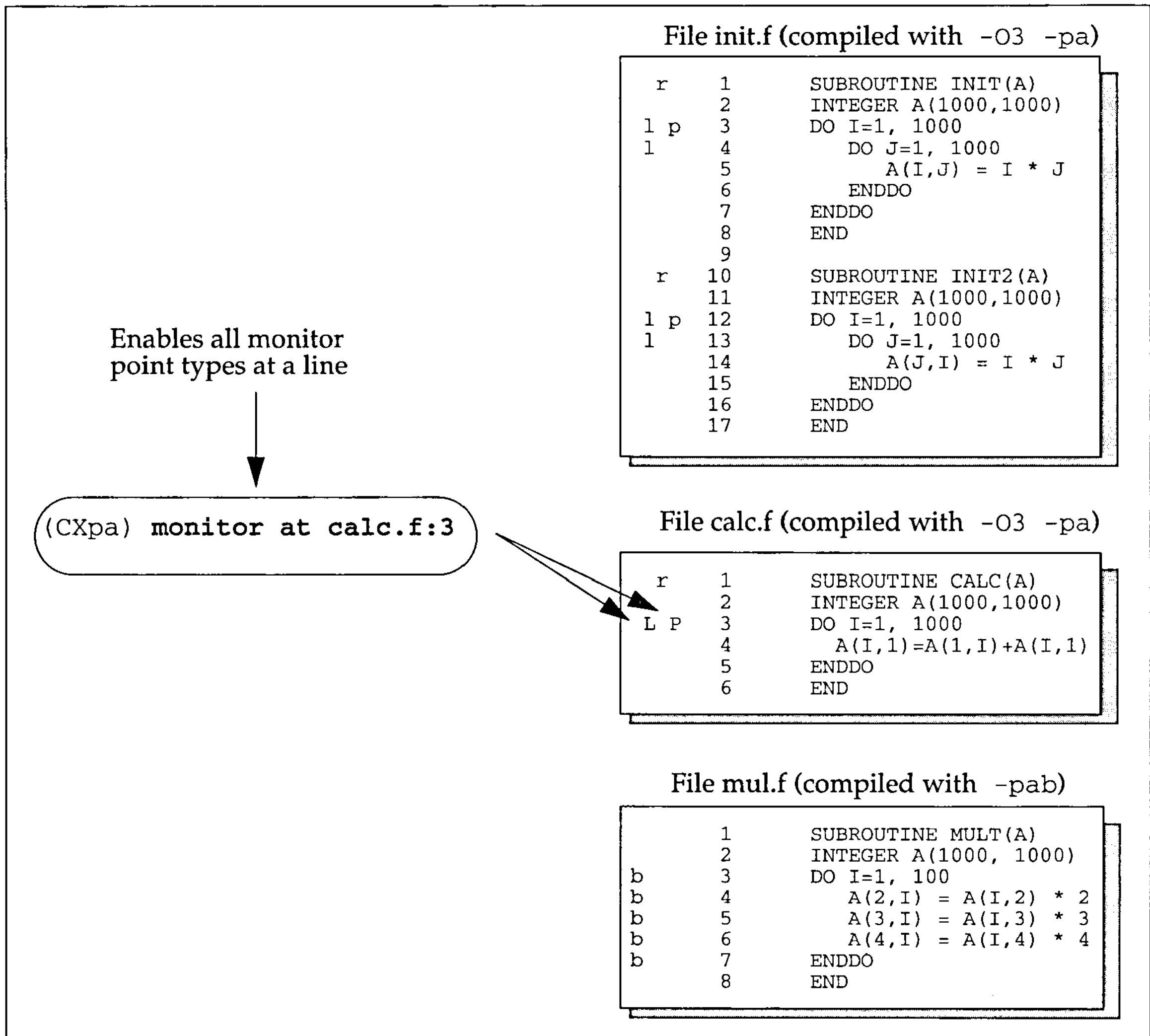


Enabling all monitor point types at select lines

You can enable all types of monitor points at specific source lines. This makes it possible to profile a single loop in a routine without having to profile all of the loops within that routine.

To enable all types of monitor points at a line in a specific routine, use the `monitor at` command, followed by the file name and line number you want to enable. The file name and line number are separated by a colon. The example below shows how to enable all types of monitor points at a particular line.

Figure 35
Enabling all monitor point types at a line



Enabling one monitor point type at select lines

If you do not want to enable all of the monitor points at a particular line, you can enable a particular monitor point type at a line. This makes it possible to profile a parallel region without profiling the loop around it.

To enable a type of monitor point at a line, use one of the commands in Table 9.

Table 9 Commands for enabling monitor points at a line

Command	Monitor points enabled
`monitor loop at`	All loop monitor points at a line
`monitor pregion at`	All parallel region monitor points at a line
`monitor block at`	All block monitor points at a line

Using one of these commands enables the monitor points of that type at the specified line in the specified file. Multiple line numbers can be specified and are separated by spaces on the command line.

The `monitor pregion at` command in Figure 36 enables the parallel region monitor point at line 3 in the calc.f source file. The loop monitor point on the same line remains disabled.

Figure 36
Enabling one monitor point type at a line

File init.f (compiled with `-O3 -pa`)

```
 r     1        SUBROUTINE INIT(A)
       2        INTEGER A(1000,1000)
1 p    3        DO I=1, 1000
1      4           DO J=1, 1000
       5              A(I,J) = I * J
       6           ENDDO
       7        ENDDO
       8        END
       9
 r    10        SUBROUTINE INIT2(A)
      11        INTEGER A(1000,1000)
1 p   12        DO I=1, 1000
1     13           DO J=1, 1000
      14              A(J,I) = I * J
      15           ENDDO
      16        ENDDO
      17        END
```

Enables one monitor point type at a line

```
(CXpa) monitor pregion at calc.f:3
```

File calc.f (compiled with `-O3 -pa`)

```
 r     1        SUBROUTINE CALC(A)
       2        INTEGER A(1000,1000)
1 P    3        DO I=1, 1000
       4          A(I,1)=A(1,I)+A(I,1)
       5        ENDDO
       6        END
```

File mul.f (compiled with `-pab`)

```
       1        SUBROUTINE MULT(A)
       2        INTEGER A(1000, 1000)
b      3        DO I=1, 100
b      4           A(2,I) = A(I,2) * 2
b      5           A(3,I) = A(I,3) * 3
b      6           A(4,I) = A(I,4) * 4
b      7        ENDDO
       8        END
```

Disabling monitor points

You can disable monitor points that have been enabled. This makes it easy to enable a large number of routines and then later disable the monitor points within those routines at which you no longer want to collect performance data.

Using the X interface, disable monitor points in the same way that you enable them. Move the routine whose monitor points you want disabled from the Enabled list to the Disabled list, as described in "Using the Set Monitor Points dialog box," on page 48.

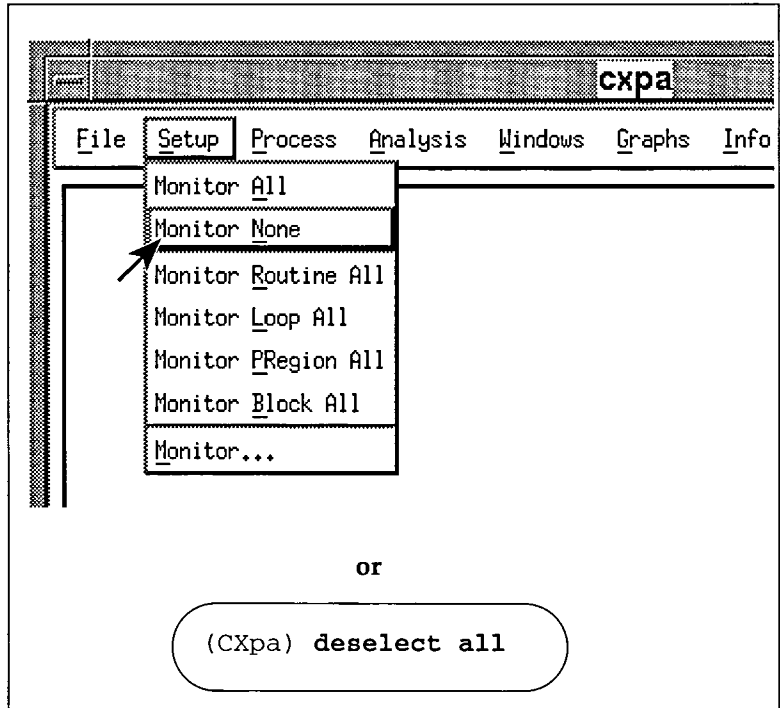
At the command line, you can disable monitor points using the `deselect` command. The variations of the `deselect` command are described in Table 10.

Table 10 Commands to disable monitor points

Monitor Point		Command to disable
Type	**Location**	
All	All routines	`deselect all`
All	In a routine	`deselect`
All	At a line	`deselect at`
Block	All routines	`deselect block all`
Block	In a routine	`deselect block in`
Block	At a line	`deselect block at`
Loop	All routines	`deselect loop all`
Loop	In a routine	`deselect loop in`
Loop	At a line	`deselect loop at`
Routine	All routines	`deselect routine all`
Routine	In a routine	`deselect routine`
Parallel region	All routines	`deselect pregion all`
Parallel region	In a routine	`deselect pregion in`
Parallel region	At a line	`deselect pregion at`

You can disable all monitor points at once by selecting the Monitor None option from the Setup menu, or by using the `deselect all` command. Figure 37 shows you how to disable all monitor points.

Figure 37
Disabling all monitor points in all routines



Using monitor points

Listing monitor points

When enabling a subset of monitor points, it is helpful to get a list of the available monitor points in your program. CXpa displays your original source code with special annotations indicating the location, type, and status of all the monitor points in your program.

When listing monitor points with CXpa, your source code is displayed with special annotations, as shown in Table 11.

Table 11 Monitor point annotations in source listings

Type	Enabled	Disabled
Routine	`R`	`r`
Loop	`L`	`l`
Parallel region	`P`	`p`
Block	`B`	`b`

Search paths and listing monitor points

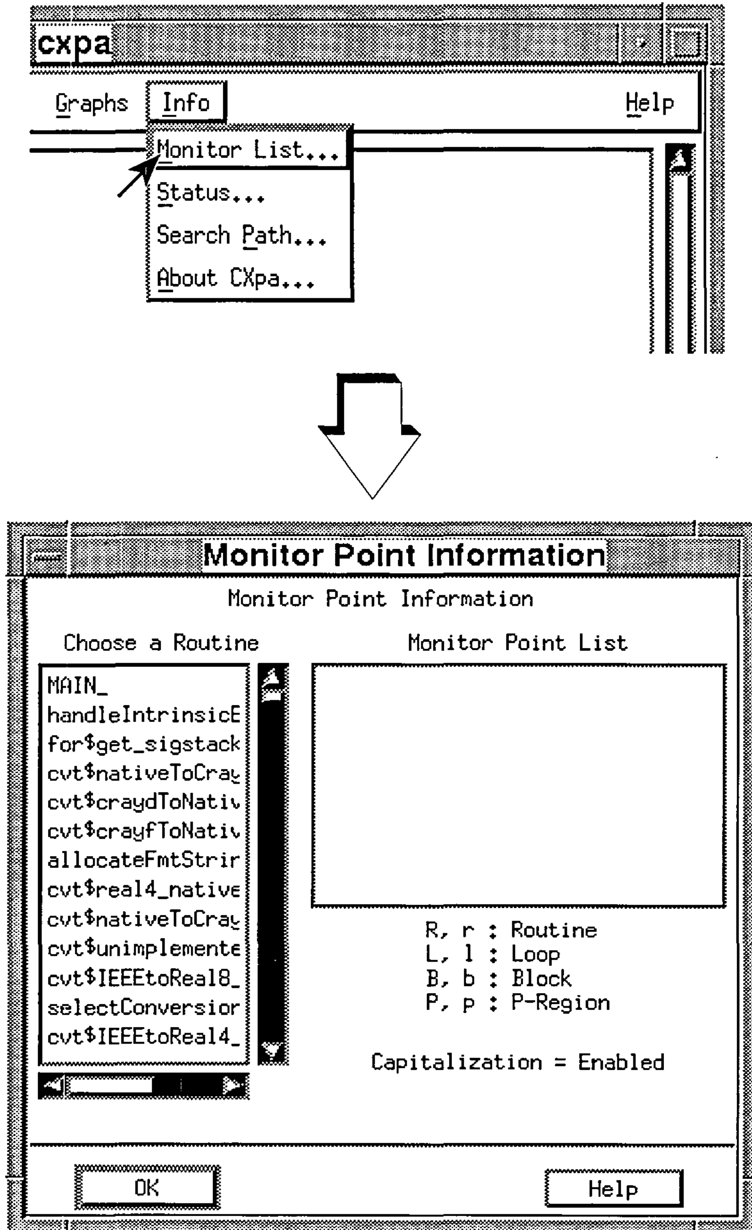
For CXpa to display the original source code, it must be able to locate the source file. CXpa uses a search path to locate these files. If CXpa cannot find the necessary source file, it will display an error message.

You can update the search path so CXpa can find the files. For more information on setting the search path, refer to "Specifying the search path," on page 74.

Listing monitor points from the X interface

To display the monitor points in a routine, select the Monitor List option under the Info menu. The Monitor Point Information dialog box appears, as shown in Figure 38.

Figure 38
Opening the Monitor Point Information dialog box



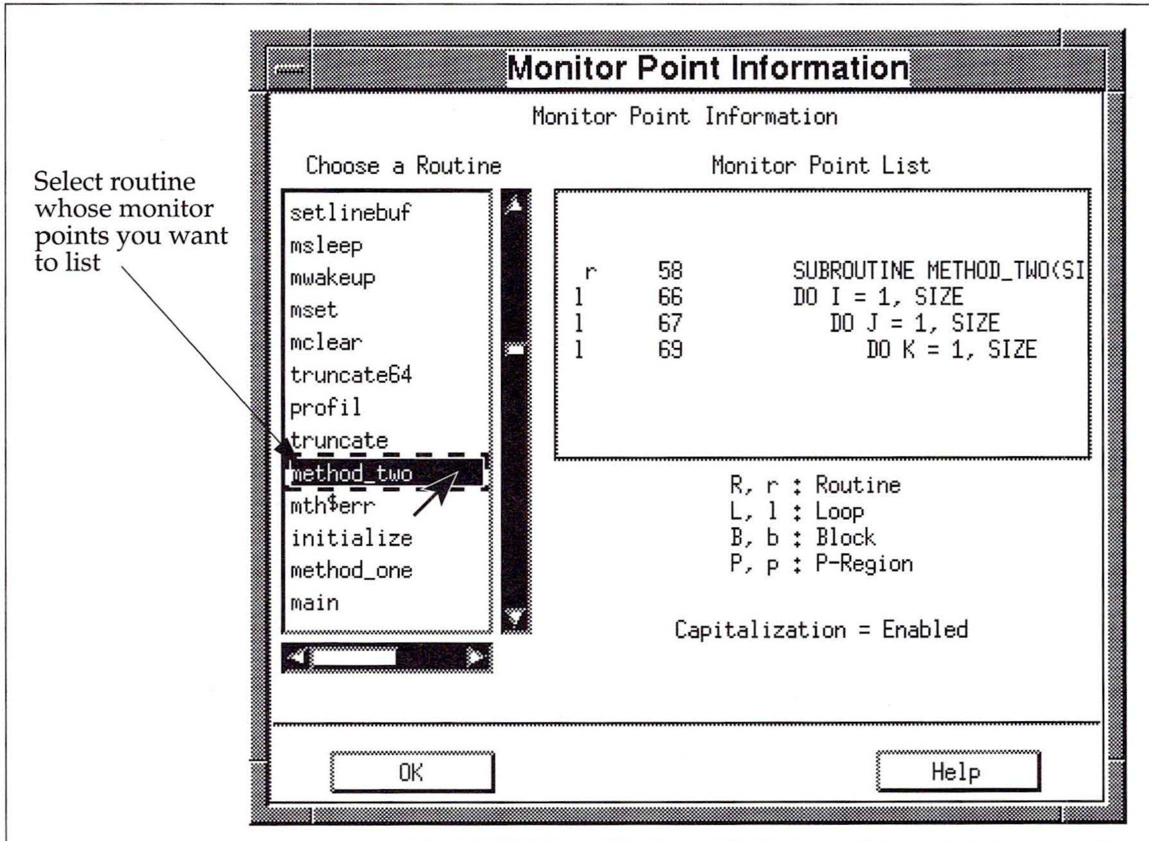
Using monitor points

Select the routine whose monitor points you want to display from the list of routines. You can scroll the list using the scroll bars.

When you select a routine, any monitor points in that routine are displayed, along with the source line they are monitoring, in the box to the right.

Figure 39 illustrates the use of this dialog box.

Figure 39
Using the Monitor Point Information dialog box



Click the OK button to close the Monitor Point Information dialog box after you are finished displaying monitor points.

Listing monitor points from the command line

CXpa has two commands for displaying the source code for your program with annotations showing the location, type, and status of monitor points:

- `list monitors`—Displays only lines contain monitor points in your program. It is useful if you do not need to look at the source code surrounding the monitor point.
- `list`—Displays the source code for your program, including lines with monitor points. Use the list command to display your source code from within CXpa.

If you do not specify a file name with either command, CXpa lists the lines from the current source file. The current source file is either the last source file specified in a CXpa command, or the source file that corresponds to the first object file given to the linker to form the current executable. You can display the current source file using the `info cxpa` command.

The commands are explained more fully in the following two sections. For complete details on both of these commands, refer to the *CONVEX CXpa Reference.*

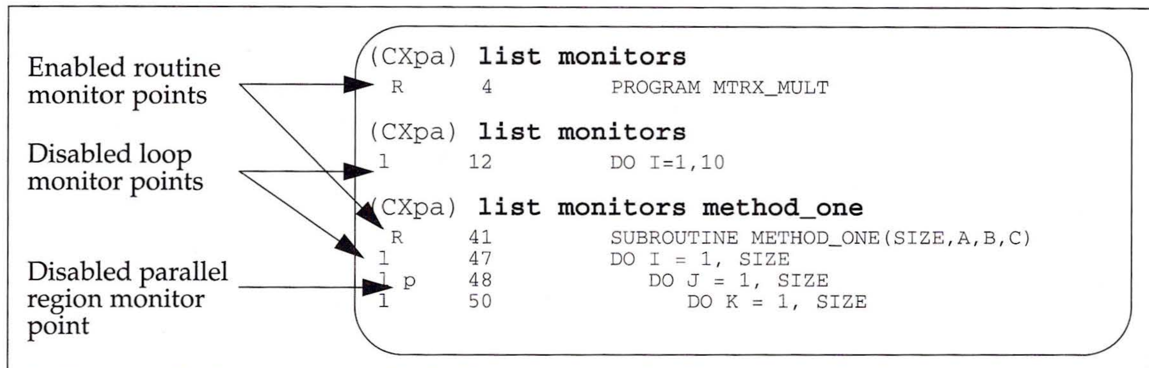
Listing only monitor points

The `list monitors` command displays only the source lines of your program that contain monitor points. When entered without parameters, the `list monitors` command lists the monitor points in the next 10 lines of the current source file.

You can provide the `list monitors` command with the name of a routine or line numbers to list. You can precede either with a file name to list from a different file.

Several examples of the `list monitors` command are given in Figure 40.

Figure 40
Listing monitor points from the command line



In Figure 40, the first `list monitors` command lists the monitor points in the first 10 lines (1 through 10) of the current source file. An enabled routine monitor point (indicated by the letter `R`) exists at line 4 of the source file.

The second `list monitors` command displays monitor points in the next 10 lines (11 through 20) of source code. A disabled loop monitor point exists at line 12.

The third `list monitors` command lists all of the monitor points in the routine `method_one`. The routine has an enabled routine monitor point, three disabled loop monitor points, and one disabled parallel region monitor point.

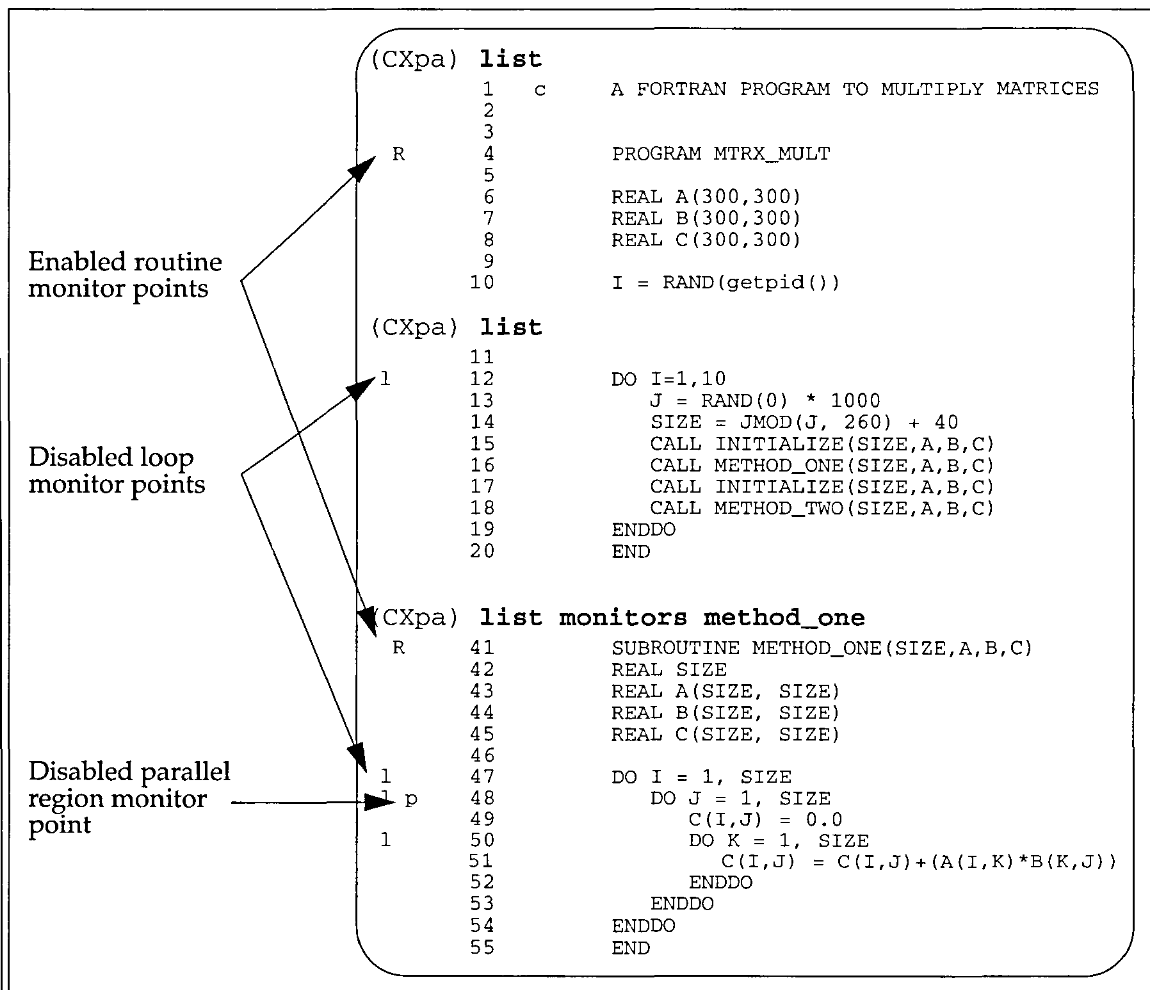
Listing all source lines

To look at all lines of source code in your program, you can use the `list` command. When entered without parameters, the `list` command lists the next 10 lines of the current source file.

You can provide the `list` command with the name of a routine or line numbers to list. You can precede either with a file name to list from a different file.

Several examples of the `list` command are shown in Figure 41.

Figure 41
Listing source lines from the command line



In Figure 41, the first `list` command lists the first 10 lines (1 through 10) of the current source file. An enabled routine monitor point (indicated by the letter `R`) exists at line 4 of the source file.

The second `list` command displays the next 10 lines (11 through 20) of source code.

The third `list` command lists all of the lines in the routine `method_one`. The routine has an enabled routine monitor point, three disabled loop monitor points, and one disabled parallel region monitor point.

Specifying the search path

The CXpa search path is a list of directories CXpa uses to find the source files for your program.

Initially, the search path is set to the directory from which you invoked CXpa. If you specify an executable or a performance data file (PDF) on the command line, the directories of these files are added to the search path as well.

If CXpa cannot find a source file during a session, it displays an error message. You can then update the search path so CXpa can find the file.

There are three methods for changing the search path from its initial settings:

- From the command line
- Through the Change Search Path dialog box
- From the shell prompt

Each of these methods is described in the following sections.

Setting the search path from the command line

To list the search path at the command line, use the `info path` command, as shown in Figure 42.

Figure 42
Displaying the current search path from the command line

```
(CXpa) info path
       /usr/data/
       /usr/data/input/
       /usr/data/output/
```

To add a directory to the search path, use the `add path` command, as shown in Figure 43.

Figure 43
Adding to the search path from the command line

```
(CXpa) add path /usr/smith/project/src
(CXpa) info path
       /usr/data/
       /usr/data/input/
       /usr/data/output/
       /usr/smith/project/src/
```

Finally, you can completely replace the current search path with a new search path using the `path` command, as shown in Figure 44.

Figure 44
Replacing the search path from the command line

```
(CXpa) path /usr/data /usr/smith/project/src
(CXpa) info path
       /usr/data/
       /usr/smith/project/src/
```

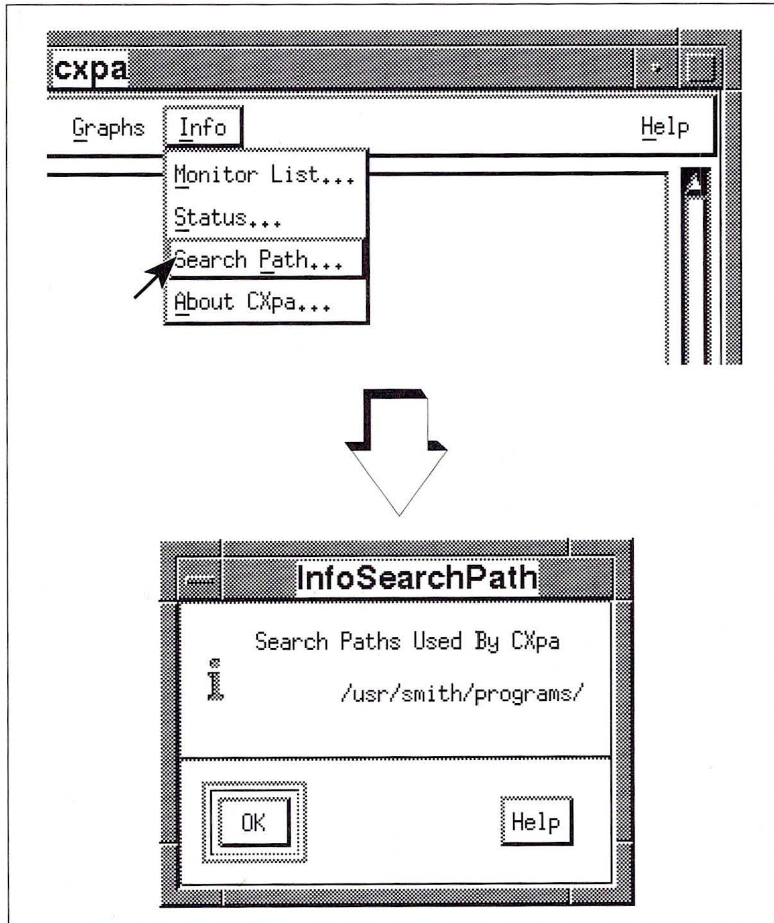
Using monitor points

Setting the search path from the X interface

You can list, add or remove directories from the search path. You can also clear the search path completely and then add new directories.

To list the directories in the search path, select the Search Path option from the Info menu of the CXpa window. Figure 45 demonstrates how to list the directories in the search path.

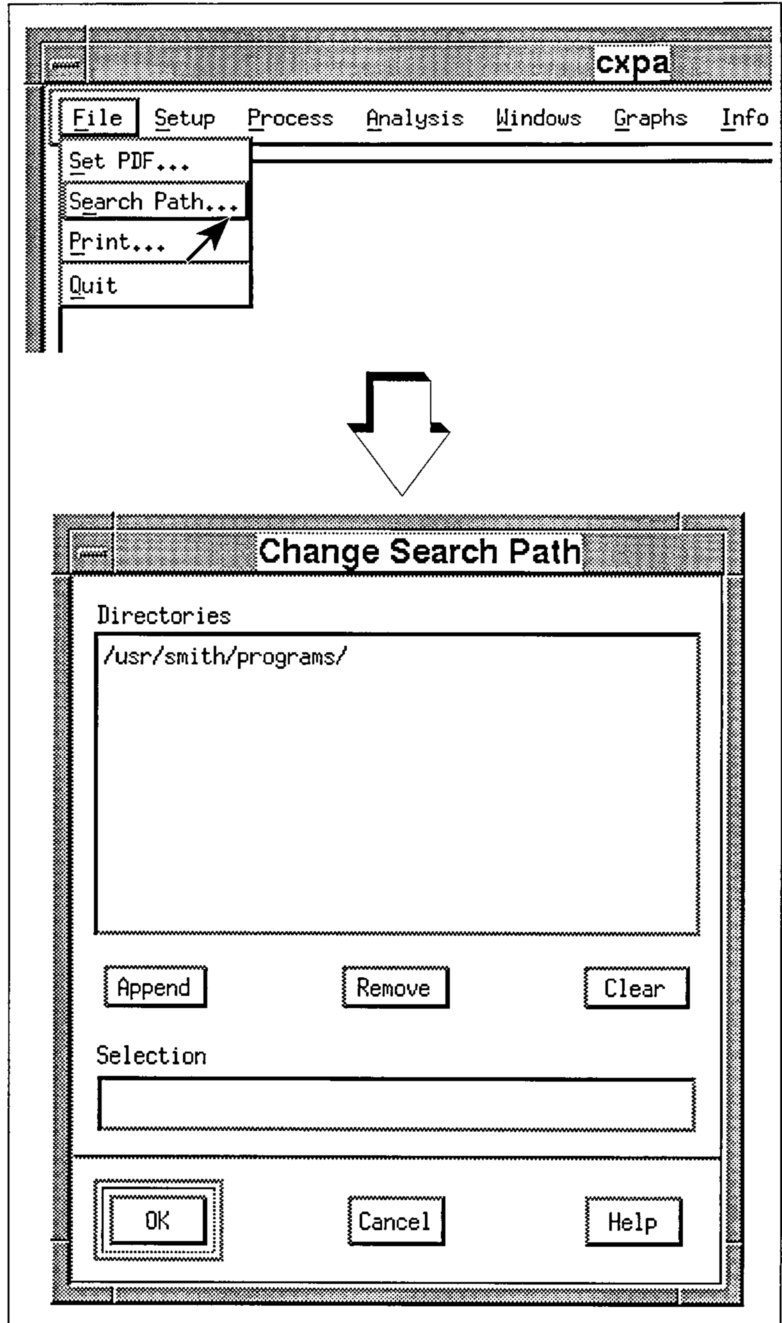
Figure 45
Displaying the search path from the X interface



Close the Info Search Path dialog box by clicking on the OK button or pressing **RETURN**.

To change the directories in the search path, select the Search Path option under the File menu. The Change Search Path dialog box appears, as shown in Figure 46.

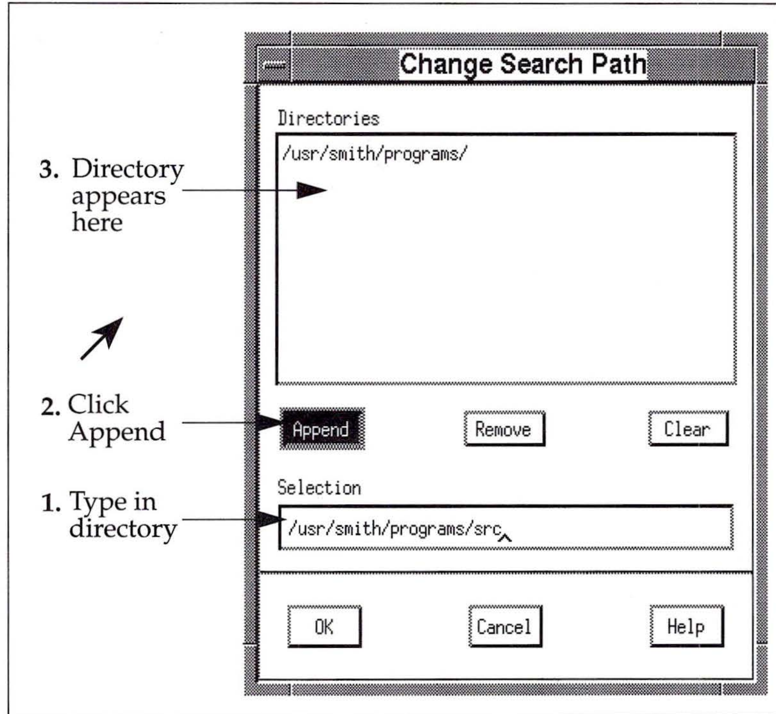
Figure 46
Opening the Change Search Path dialog box



Using monitor points

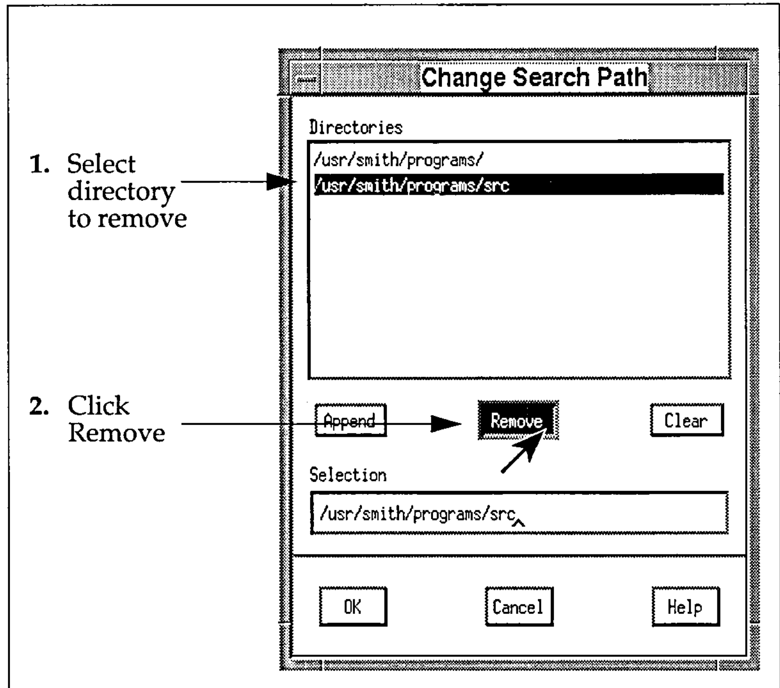
To add a new directory to the search path, enter the name of the directory in the text field, then click the Append button. The directory appears in the top of the window, as shown in Figure 47.

Figure 47
Adding to the search path from the X interface



To remove a directory from the search path, select the directory from the list, then click the Remove button, as shown in Figure 48. The directory is removed from the list at the top of the window.

Figure 48
Removing a directory from the search path

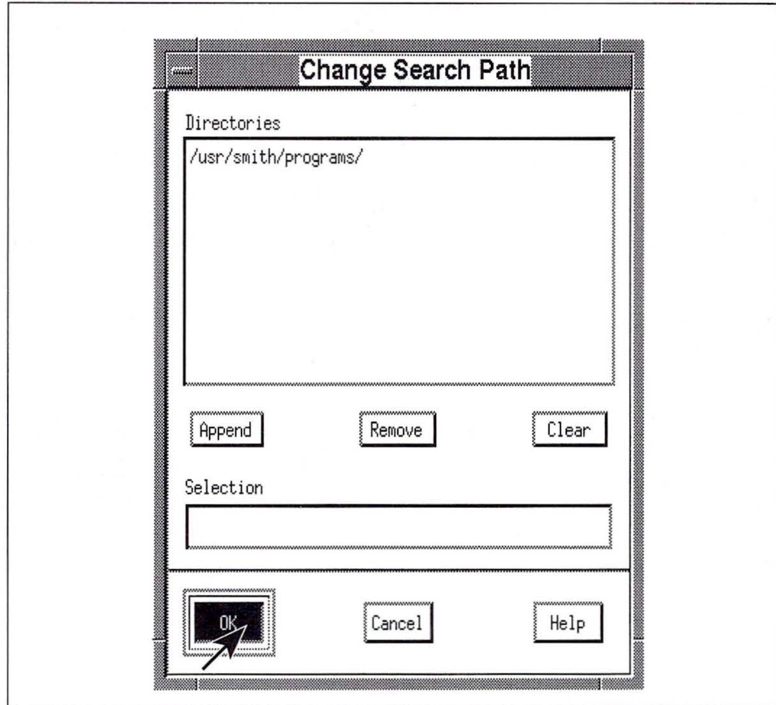


To clear the search path of all directories, click the Clear button. All directories are removed from the Directories field.

Using monitor points

To apply the changes you have made, click the OK button, as shown in Figure 49.

Figure 49
Applying changes to the search path from the X interface



Once you click the OK button, CXpa updates the search path to reflect the new list of directories shown at the top of the Change Search Path dialog box. CXpa then closes the dialog box.

If you do not want to update the search path, click the Cancel button. The dialog box closes, and the search path remains unchanged.

Setting the search path at the shell prompt

You can add directories to the search path using the `-path` command line option when you invoke CXpa, as shown in Figure 50.

Figure 50
Setting the search path using the shell prompt

```
% cxpa -path /usr/smith/project/src
```

In Figure 50, the directory /usr/smith/project/src is added to the CXpa search path. The search path also includes the directory from which CXpa was invoked.

Controlling profiling

5

This chapter explains how to run your program under CXpa. As your program runs, CXpa collects performance data at each enabled monitor point.

If your program takes a long time to profile, you can pause profiling, analyze the results collected to that point, and then resume or stop profiling. This enables you to determine the performance of long runs without having to wait until execution finishes.

The topics covered in this chapter include:

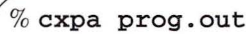
- Specifying an executable
- Specifying the performance data file (PDF)
- Running your program
- Pausing profiling
- Continuing profiling
- Stopping profiling
- Rerunning your program

Specifying an executable

During a CXpa session you can only profile a single executable. The executable must have been compiled for use with CXpa. To profile a different executable, you must quit and restart CXpa.

To specify an executable, invoke CXpa with the name of an executable, as shown in Figure 51.

Figure 51
Specifying an executable when invoking CXpa



```
% cxpa prog.out
```

The above command invokes CXpa with the executable named `prog.out` that exists in the current directory.

Note

You do not need to specify your executable on the command line if it is named `a.out`. If an executable is not specified, CXpa looks for the executable file `a.out` in the current directory. If the file exists, it becomes the executable being profiled.

If `a.out` does not exist, no executable can be profiled for this session. However, you can still use CXpa to analyze performance data files created during previous CXpa sessions.

Setting the performance data file (PDF)

A performance data file (PDF) is a binary file that contains the data CXpa collects from a single run of your program.

The PDF is used in two cases:

- CXpa writes performance data to this file when you run your program. If the PDF does not yet exist, CXpa creates it. If it does exist, CXpa will overwrite it.
- CXpa analyzes the data in this file when it generates reports. The PDF must exist for CXpa to be able to analyze it.

By default, the PDF is named cxpa.pdf in the current directory (the directory from which you invoked CXpa). You can profile your program and generate a report without ever needing to specify the PDF. Each time, CXpa will continue to use the default, thus overwriting cxpa.pdf.

However, you can change the name of the PDF. You may want to do this for two reasons:

- To prevent CXpa from overwriting the PDF
- To analyze a different PDF

For example, suppose you want to compare the performance of a program with two different sets of input data. Before the first run, you could change the name of the PDF to data1.pdf. Before the second run you could change the PDF to data2.pdf. Generating a report at this point would use data2.pdf (it is the PDF). To analyze the data in the first PDF, you would change the PDF back to data1.pdf.

PDFs you create may have any name, and do not need to end with a .pdf extension.

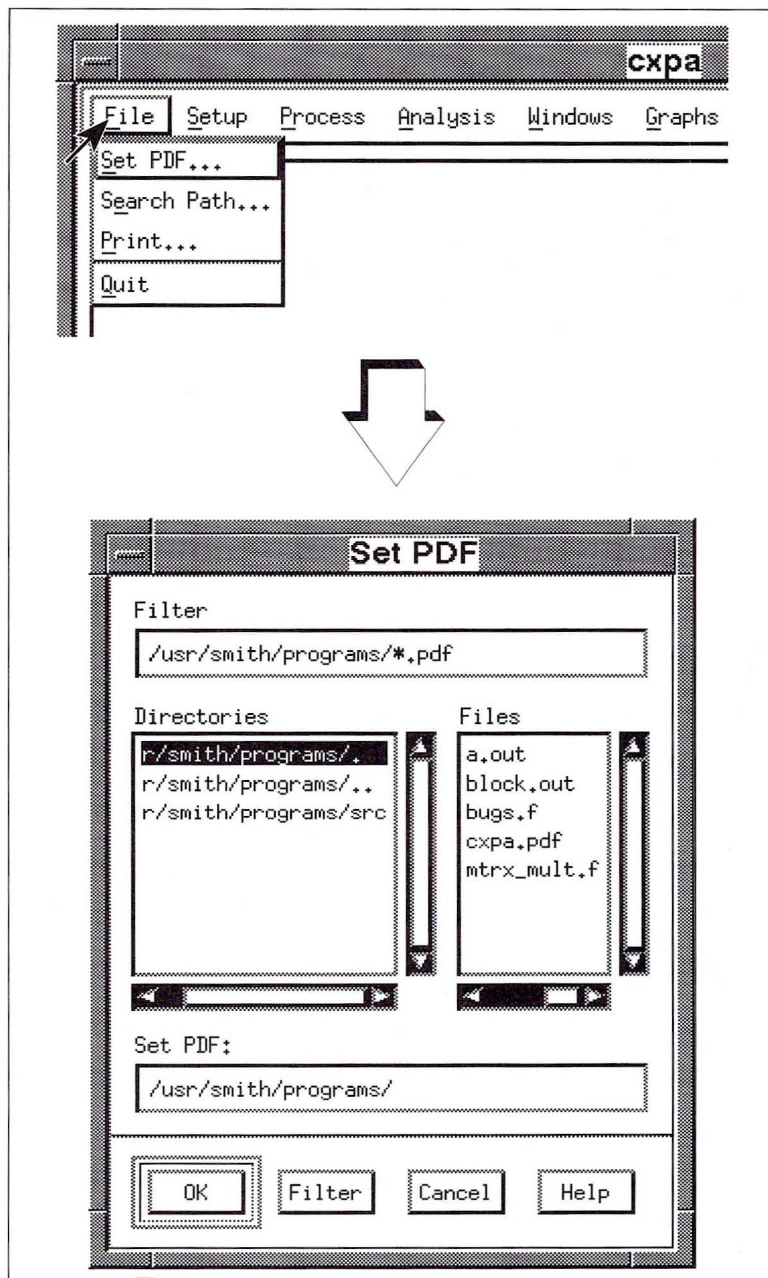
Note

In order to analyze data from a PDF, the PDF must have been created with the executable that you are currently profiling. To analyze a PDF created wtih a different executable, invoke CXpa without specifying an executable.

Setting the PDF from the X interface

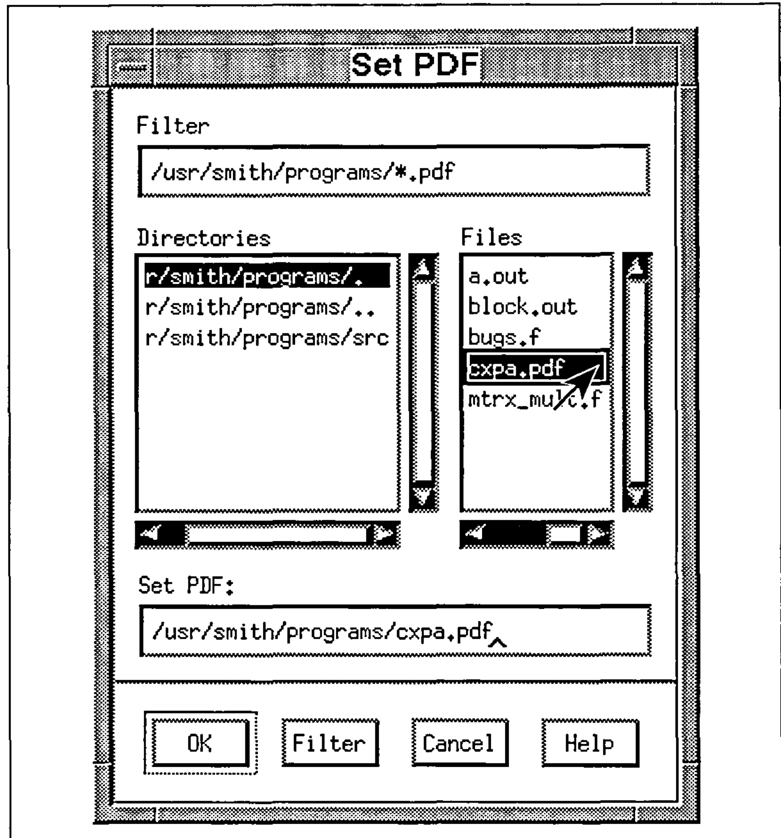
Set the PDF by selecting the Set PDF option from the File menu from the main window. This is demonstrated in Figure 52.

Figure 52
Setting the current PDF from the X interface



When you select the Set PDF option, the Set PDF dialog box opens. This is a standard Motif file selection dialog box.

Figure 53
Set PDF dialog box



You can use the dialog box to select a PDF and to filter the list of files.

Selecting a PDF

You can select a PDF in the following ways:

- Double-click on a file name in the Files list.
- Select a file in the Files list and click the OK button.
- Type the full path name to the file in the Set PDF field and click the OK button or press **RETURN**.

Filtering files

When the file selection box appears, the Filter field will contain the path to the current directory with *.pdf appended to it. To use the filter feature:

Step 1 Alter the path name in the Filter field by typing in the field or by clicking on one of the directories. You can use wildcards (* or ?).

Step 2 Press the Filter button. The Files list displays files, and the Directories list displays directories that match the specification in the Filter field (usually *.pdf).

Setting the current PDF from the command line

You can also set the new PDF using the `set pdf` command. In Figure 54, new.pdf becomes the PDF. If a program is now run, the data is collected in new.pdf. If a report is generated, CXpa analyzes the data in new.pdf.

Figure 54
Setting the current PDF at the command line

```
(CXpa) set pdf new.pdf
```

Setting the PDF from the shell prompt

Finally, you can use the `-pdf` option when invoking CXpa to specify the name of the PDF. In Figure 55, CXpa is invoked with the PDF named data1.pdf. When you invoke CXpa using the `-pdf` option, CXpa does not look for an executable file.

Figure 55
Invoking CXpa with a specified PDF

```
% cxpa -pdf data1.pdf
```

Once you have specified the `-pdf` option on the shell command line, you can only use CXpa to analyze PDFs created by different executables in the same session.

Running your program

When you run your program, CXpa collects performance data at all enabled monitor points in your program. The data collected is stored in the PDF.

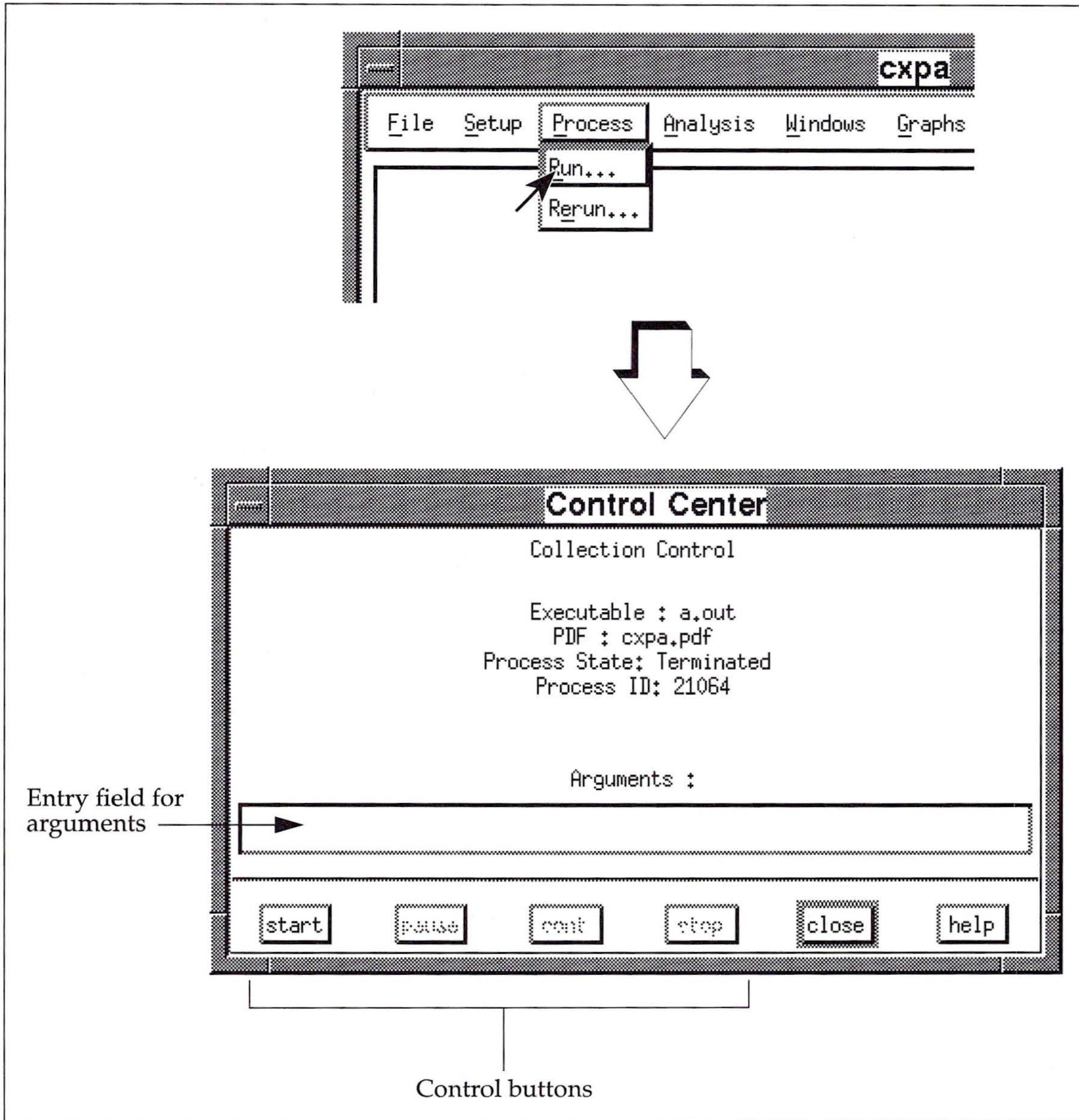
Before the program begins execution, CXpa creates a process from the executable and inserts calls to the CXpa timing routines at all enabled monitor points in your program. These timing routines enable CXpa to collect the performance data. After CXpa finishes inserting the calls, execution begins.

After these calls have been inserted, you can pause the profiling at any time and analyze the collected data. This enables you to get preliminary performance results for long runs without having to wait until the profiling is finished. Pausing and continuing profiling are discussed later in this chapter.

Running the program from the X interface

Run your program by selecting the Run option from the Process menu. The Control Center dialog box appears, as shown in Figure 56.

Figure 56
Opening the Control Center dialog box



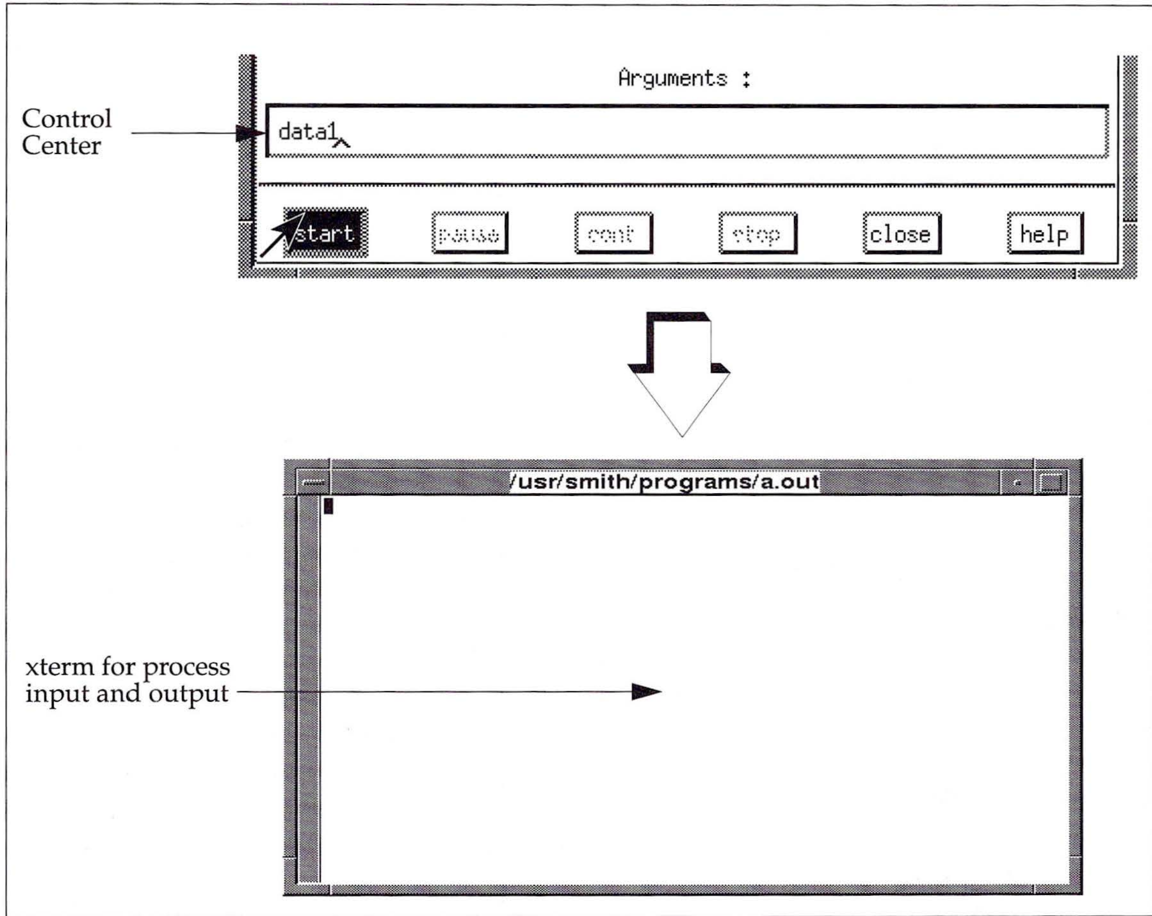
In Figure 56, the Run option opens the Control Center dialog box. This dialog box controls the execution of your program and remains open until you click the close button.

The parts of the dialog box are explained below:

- **Executable**—The name of the executable being profiled.
- **PDF**—The name of the PDF where the performance data is being collected.
- **Process State**—One of the following:
  - **No process**—No process yet exists (profiling has not started).
  - **Running**—The process is running.
  - **Paused**—Process is suspended. It may be continued or terminated.
  - **Terminated**—Process abnormally terminated by a signal.
  - **Finished**—Process exited normally.
- **Process ID**—The operating system process ID (0 indicates no process).
- **Arguments entry field**—Text entry field for entering the arguments to pass to your program. If the entry field is left empty, your program is run without arguments. You can use this field to enter C shell input and output redirection.
- **Control buttons**—These buttons control profiling. The buttons perform the following actions:
  - **start**—Starts the profiling and execution of your program.
  - **pause**—Pauses the profiling of your program. This button is desensitized (unavailable) until profiling is started.
  - **cont**—Continues profiling. This button is desensitized until profiling is paused.
  - **stop**—Stops profiling and terminates the process. Any performance data collected before stopping is saved in the PDF, and is available for analysis. This button is desensitized until profiling is started.
  - **close**—Closes the Control Center. This does not change the state of profiling. For example, if profiling is paused, it will still be paused after you close the Control Center.
  - **help**—Displays a help page for the Control Center dialog box.

To start profiling, click the start button. An xterm window opens to handle input and output to your program, as shown in Figure 57.

Figure 57
Running the program from the X interface



Profiling continues until the process completes execution (whether normally or abnormally) or until you pause profiling.

For more information on pausing a program, refer to "Pausing profiling from the X interface," on page 94.

When the process completes execution, CXpa stops data collection, closes the current PDF, and terminates the process.

The Control Center remains visible, displaying the status of the process. The xterm window remains open as well, enabling you to examine the output of your program.

Running the program from the command line

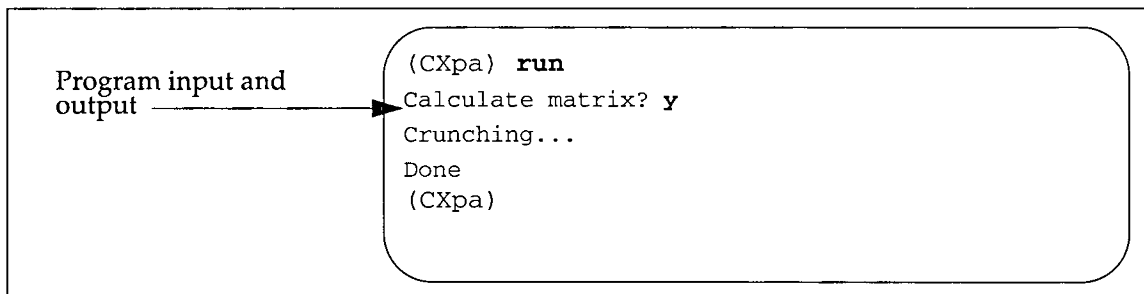
When you run your program from the command line in the CRT interface, the CXpa prompt disappears. Because the CRT window acts as the input and output window for the program, you cannot continue to enter CXpa commands. The program must either run to completion or be paused before entering additional CXpa commands.

In contrast, when running CXpa using the command window of the X interface, program input and output is sent to an xterm window. The CXpa prompt returns, enabling you to perform additional CXpa tasks while profiling.

When profiling is complete, CXpa stops data collection, closes the PDF, and terminates the process.

An example of running the program from the command line in CRT mode is illustrated in Figure 58.

Figure 58
Starting to profile using the command line



In Figure 58, the `run` command begins profiling. The program's input and output is handled by the CRT window. When the program completes, the CXpa prompt returns.

Pausing the profiling of a program

With CXpa, you can pause the profiling of your program at any point during the execution of the program. You can view a performance report based on the collected data without having to wait for the entire profile to complete.

Performance data for regions that are executing when the process is paused will be incomplete and may not be meaningful. When viewing a performance report of a paused profiling session, make sure the regions you are interested in are not paused (paused regions are marked with a `p` in the process status column of every report).

You may resume profiling (CXpa continues to collect performance data), stop the profiling, or restart profiling.

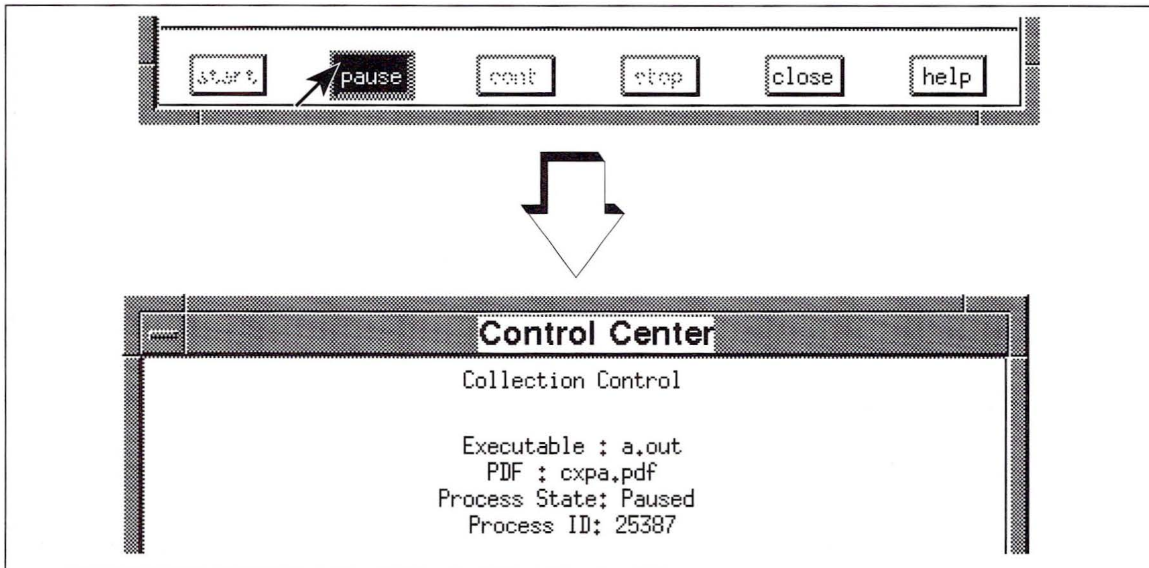
Note

When profiling is paused, you cannot change the set of enabled monitor points.

Pausing profiling from the X interface

To pause profiling using the X interface, click the pause button in the Control Center dialog box, as shown in Figure 59.

Figure 59
Pausing profiling from the X interface



When you click the pause button, profiling is paused, the process is stopped, and the Process State is displayed as `Paused`.

At this point, you could generate a report from the collected data, continue profiling, or stop profiling.

Pausing profiling at the command line

Press CTRL-c to pause profiling in the CRT interface. If you are using the command window, use the `pause` command to pause profiling.

After you press CTRL-c, profiling is paused, and the CXpa prompt returns.

Figure 60 demonstrates how to pause profiling using the command line in the CRT interface.

Figure 60
Pausing the program from the command line

```
(CXpa) run
Calculate matrix? y
Crunching...
^C
(CXpa)
```

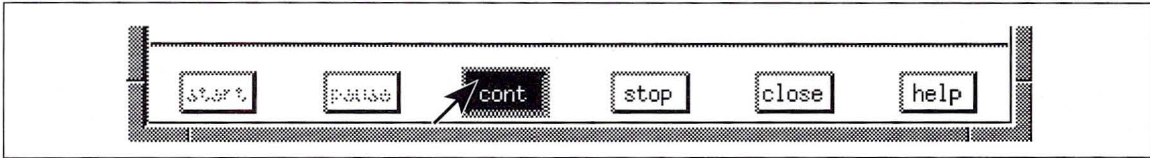
Continuing the profiling of a program

If you pause profiling, you can resume collecting performance data using the `continue` command. There is no limit to how many times you may pause and continue profiling.

Continuing profiling from the X interface

From the X interface, you can continue profiling by clicking the cont button at the bottom of the Control Center dialog box, as shown in Figure 61.

Figure 61
Continuing profiling from the X interface



Continuing profiling at the command line

From the command line, you can continue profiling using the `continue` command, as shown in Figure 62.

Figure 62
Continuing profiling from the command line

```
(CXpa) run
Calculate matrix?: y
Crunching...
^C
(CXpa) continue
Done
(CXpa)
```

In the above example, the `run` command begins profiling. The `pause` command (CTRL-c) pauses profiling, and the CXpa prompt reappears. The `continue` command resumes profiling. The program then runs to completion.

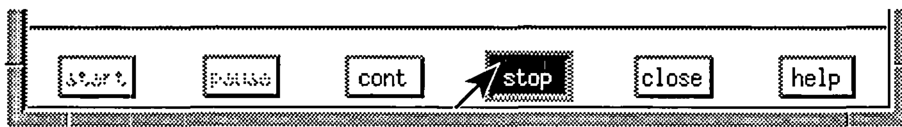
Stopping the profiling of a program

You can stop the profiling of a program once profiling has been paused. When you stop profiling, data collection is stopped, the current PDF is closed, and the process is terminated. You can still analyze all performance data that was collected.

Stopping profiling from the X interface

To stop profiling, click the stop button at the bottom of the Control Center dialog box, as shown in Figure 63.

Figure 63
Stopping profiling from the X interface



Stopping profiling at the command line

To stop profiling at the command line, use the `stop` command. This is demonstrated in Figure 64.

Figure 64
Stopping profiling from the command line

```
(CXpa) run
Calculate matrix? y
Crunching...
^C
(CXpa) stop
 Process 14995 terminated with signal: SIGKILL.
(CXpa)
```

In Figure 64, the `run` command begins profiling. The `pause` command (CTRL-c) pauses profiling, and the CXpa prompt reappears. The `stop` command stops profiling and terminates the process.

Rerunning the program

If you want to rerun the program with the same arguments as the previous run, you can do so automatically. Rerunning the program saves you the trouble of retyping arguments each time you want to run the program.

If you have not run the program previously, rerunning your program will begin profiling without passing any arguments to your process.

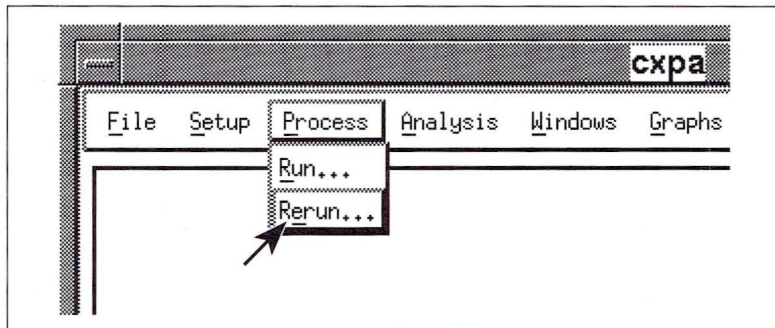
Note

Rerunning the program does not duplicate shell redirection. If you redirected process input and/or output, rerunning the program will not use the previous shell redirection.

Rerunning the program from the X interface

To rerun the program, select the Rerun option from the Process menu on the main window, as shown in Figure 65.

Figure 65
Rerunning the program



When you select the Rerun option, the Control Center opens, and profiling begins (with the same arguments as used in the previous run, or with no arguments if this is the first run). The xterm window opens for the program's input and output.

Rerunning the program from the command line

At the command line, use the `rerun` command, as demonstrated in Figure 66.

Figure 66
Rerunning the program from the command line

```
(CXpa) run data1 > output_data1
(CXpa) rerun
Using data1...
Done
```

In the above example, the `run` command runs the program with the argument `data1` and redirects program output to the file output_data1 (`> output_data1`).

The `rerun` command runs the program again, with the same argument (`data1`), but the output redirection is not duplicated.

Generating and interpreting reports

6

This chapter describes how to generate and interpret CXpa's performance reports. These reports provide performance measurements, such as execution timings for routines or loop iteration counts.

There is a separate report for each type of region profiled by CXpa. The available reports are:

- Routine Report
- Loop Report
- Parallel Region Report
- Basic Block Report

The beginning of this chapter explains how to generate and print the reports. It also describes features that are common to all reports. Each of the reports is then described in detail.

The end of the chapter describes the routine performance bar chart, available from the X interface. The bar chart is a graphical summary of routine performance.

Generating reports

CXpa builds performance reports by analyzing the data in the performance data file (PDF). The reports provide performance measurements on the monitored regions of your program.

CXpa produces a report for each type of region you can monitor (routines, loops, parallel regions, and basic blocks).

CXpa can generate a particular report if data for that region was collected during profiling. CXpa only collects data at executed regions with enabled monitor points. You can generate all reports available from the PDF or individual reports.

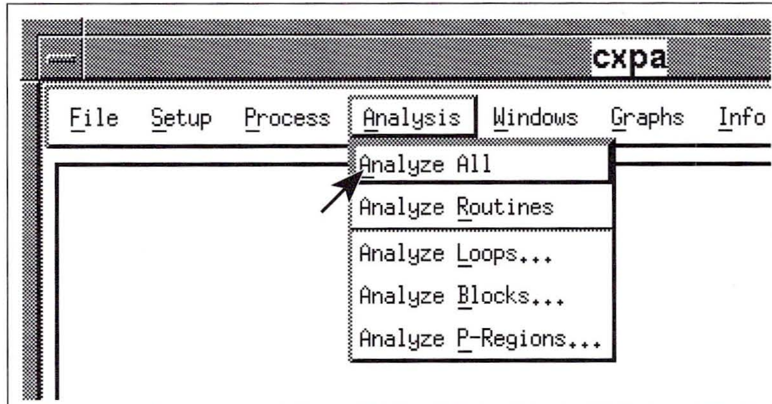
Note

For all reports, CPU times are given in seconds. Times less than .001 seconds are given in milliseconds and are followed by the abbreviation `m`**.**

Generating reports from the X interface

Reports are generated using the Analysis menu on the CXpa window. To generate all possible reports from the PDF, select the Analyze All option from the Analysis menu, as illustrated in Figure 67.

Figure 67
Generating all reports from the X interface



To generate an individual report, select that report from the menu. For example, to generate the Loop Report, select the Analyze Loops option from the Analysis menu.

Generating reports from the command line

You generate reports using the `analyze` command. CXpa displays reports at the command line using the pager specified by your `$PAGER` environment variable. If no pager is specified, CXpa uses the `more` utility.

To generate an individual report, use the `analyze` command followed by the keyword (`routine`, `loop`, `pregion`, or `block`) corresponding to the report you want to view.

To generate all reports, use the `analyze` command without any parameters.

Figure 68 illustrates how to generate the Loop Performance Analysis report using the `analyze loop` command.

Figure 68
Generating a report from the command line

```
(CXpa) analyze loop
===============================================================================
                           Loop Performance Analysis
                                For: method_one
===============================================================================
Summary:
                          Times      Iteration Count        CPU Time
 Line                     Exec      Min      Max     Avg   (plus inner loops) PS
-------                 -------- ------- ------- ------- ------------------ --
    47                      1486      49     277     185           2.555238
    48                        10      49     277     148           2.568616
```

For more information on the `analyze` command, refer to the *CONVEX CXpa Reference*.

Printing reports

Once you have generated the reports, you can print them to a file or printer. Reports can be printed to a printer or to an ASCII file.

You can print a report from the command line or from the X interface. Using the X interface, you can send the report directly to a printer.

Printing reports from the command line

To print the reports from the command line, redirect the output of the `analyze` command to a file, as shown in Figure 69.

Figure 69
Printing a report from the command line

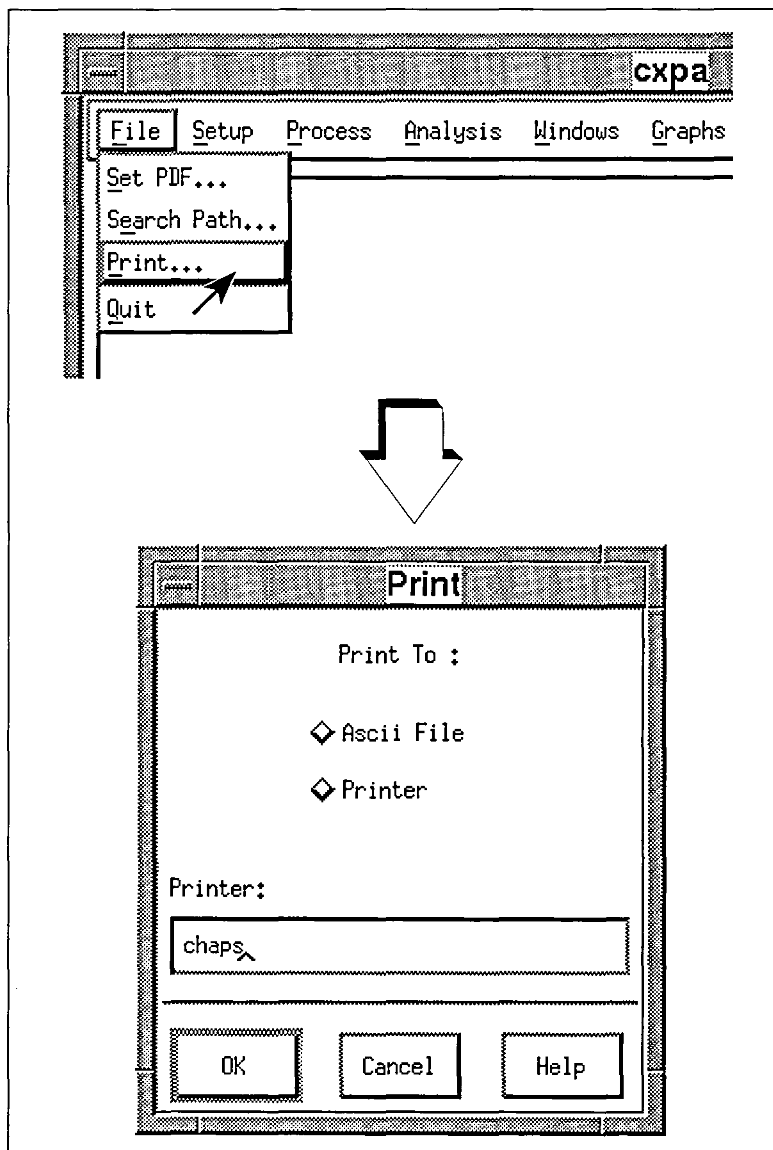
```
(CXpa) analyze > my.report
```

The above command generates all possible reports from the PDF and saves the output to the my.report file in the current directory. You can send the text file to your printer as you normally would.

Printing reports from the X interface

To print the reports being displayed in the CXpa window of the X interface, select the Print option from the File menu. The Print dialog box opens, as shown Figure 70.

Figure 70
Opening the Print dialog box



Printing to a printer

By default, the Print dialog box is set to print to the printer specified by your `$PRINTER` environment variable (if the variable is not set, no printer is specified).

To print the graph to this printer, click the OK button. To print the graph to a different printer, enter the name of the printer in the text field, then click OK.

Printing to a file

Print the report to a file using the following steps:

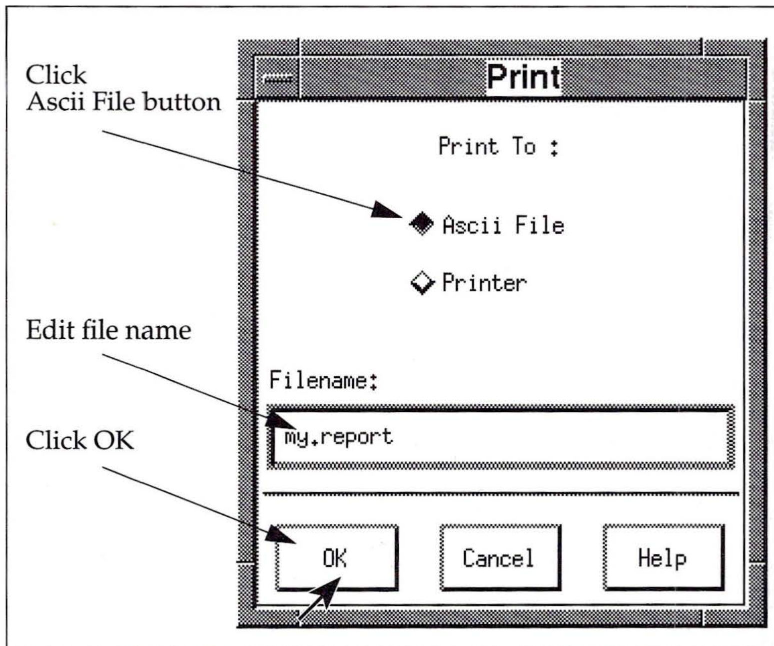
Step 1 Click the Ascii File button.

Step 2 If you do not want to use the default file name of CXpa.report, enter a file name in the Filename entry field.

Step 3 Click the OK button to print the file and close the dialog box.

These steps are demonstrated in Figure 71.

Figure 71
Printing the report from the X interface



Routine Report

The Routine Report gives performance measurements for every monitored routine that was executed, including library functions. The report can contain three tables:

- **Routine Performance Analysis table**—Displays the total time spent in each routine, both with and without time spent in called routines. The table also displays timing statistics for a single call to the routine and the number of times the routine was executed.
- **Dynamic Call Graph table**—Displays the parents (calling routines) and children (called routines) for each routine. Routines are listed in the order they were executed.

Each of these tables is described in the following sections.

Routine Performance Analysis table

The Routine Performance Analysis table is divided into two sections:

- **Summary section**—Provides an overview of routine performance for all calls.
- **Details section**—Provides data on a single call basis.

These are described in the next two sections.

Summary section

The Summary section provides the following information for each monitored routine:

- Total CPU time (both with and without time spent in called routines)
- Number of times the routine was executed
- Profiling status and additional notes about the routine

The routines with the largest CPU times (not including children) are listed first. This helps to quickly identify the routines that take the greatest portion of total execution time, often called the *hot* routines.

An example of the Summary section is shown in Figure 72.

Figure 72
Summary section of the Routine Performance Analysis table

```
===========================================================================
                    Routine Performance Analysis
===========================================================================
Summary:
                CPU Time                    Times
   (less children)      (plus children)    Exec     PS        Routine Name
   ---------------      --------------- --------- --    ----------------------
    31.154    72.1%      31.154    72.1%        10       method_two
    11.675    27.0%      11.675    27.0%        10       method_one
     0.374     0.9%       0.374     0.9%        30 p     initialize
     0.264m    0.0%       0.506m    0.0%         1       f_exit
```

Routine's total CPU time (m indicates milliseconds)

Percentage of program's total CPU time

Profiling status

The Summary section is divided into four major columns for each routine of your program and any library functions called:

- **CPU Time**—Lists the total CPU time and percentage of total execution time taken by the routine. The column is divided into two categories:
  - (less children)—Does not include time spent in the children of the routine.
  - (plus children)—Includes the total time spent in all the children of the routine.
- **Times Exec**—Lists the number of times each routine was executed.
- **PS**—Lists the profiling status of the routine. CXpa uses this column to note the state of a routine or its instrumentation. Table 12 describes the possible annotations used in the PS column.
- **Routine name**—The name of each monitored routine that was called, listed in order of highest CPU Time (less children).

Table 12 Profiling status annotations

Annotation	Meaning
e	The program exited at this point.
m	The monitor point detected invalid time management due to incorrect instrumentation in your program or a library routine. You can work around improper instrumentation in a your own routine by disabling the monitor point for that routine. You can work around improper instrumentation in a library routine by linking your program with uninstrumented libraries (refer to "Linking with uninstrumented libraries," on page 37 for details).
p	The routine was paused, and therefore the performance data is incomplete.
t	The program terminated at this point.
u	The routine includes an uninstrumented loop. CXpa cannot collect performance data for these loops because they are to small to time (typically a loop has become a vector instruction). Routine times are correct. However, loop timings are not collected.

Details section

The Details section lists the minimum, maximum, and average time spent in each monitored routine. The data indicates how execution time varies between different calls to a routine.

An example of the Details section is shown in Figure 73.

Figure 73
Details section of the Routine Performance Analysis table

```
Details:
                          CPU Time
      (less children)                          (plus children)                Routine
 Min        Max        Avg        Min        Max        Avg      PS    Name
-------  ---------  ---------  ---------  ---------  ---------  --  -------
 3.112      3.121      3.115      3.112      3.121      3.115         method_two
 1.166      1.170      1.167      1.166      1.170      1.167         method_one
 0.012      0.014      0.012      0.012      0.014      0.012         initialize
 0.264m     0.264m     0.264m     0.506m     0.506m     0.506m        f_exit
```

The columns in the section are described below:

- **CPU Time**—Lists the minimum, maximum, and average length of time spent in the routine. The column is divided into two categories:
  - (less children)—Does not include time spent in the children of the routine. *Children* are called routines.
  - (plus children)—Includes the total time spent in all the children of the routine.
- **PS**—Lists the profiling status of the routine. CXpa uses this column to note the state of a routine or its instrumentation. Table 12 describes the possible annotations used in the PS column.
- **Routine name**—Lists the name of each monitored routine in your program (including monitored library functions).

Dynamic Call Graph table

The Dynamic Call Graph table lists the routines that were monitored in the order they were executed. CXpa lists the parents and children for each routine, along with the number of times the routine was called.

If you would like to omit the Dynamic Call Graph in the Routine Report, invoke CXpa with the `-ncg` option.

If CXpa cannot determine the parent of a routine (whether because the parent was not monitored or was an uninstrumented function of the operating system), the parent is listed as a spontaneous call. (This is typically true for the main routine of the program.)

Recursive routines are labeled as the start of a cycle.

Figure 74 shows an example of the dynamic call graph for a FORTRAN program.

Figure 74
Dynamic Call Graph table

```
==============================================================================
                              Dynamic Call Graph
                    (in topological order, cycles severed))
==============================================================================

getrlimit:
  parents:  (spontaneous call)

getpid:
  parents:  (spontaneous call)

main:
 parents: (spontaneous call)
 children: f_exit[1] MAIN_[1] profil$_[1] f_init[1] sigvec[13] sigstack[1]
                    .
                    .
                    .
MAIN_:
  parents:  main
  children: method_two[10] method_one[10] initialize[10]

initialize:
  parents:  MAIN_

method_one:
  parents:  MAIN_

method_two:
  parents:  MAIN_
```

In Figure 74, `MAIN_` (the main routine of a FORTRAN program) is called by the routine `main` (a start-up routine), which is in turn called by the operating system. The `MAIN_` routine calls `method_one`, `method_two`, and `initialize`. Each routine is called 10 times, as indicated by the `[10]` notation.

Loop Report

The Loop Report consists of a Loop Performance Analysis table for each executed routine that has monitored loops. Each Loop Performance Analysis table can contain three sections:

- **Summary section**—Lists summary information such as the number of times each loop was executed, the iteration counts for each loop, and the CPU time spent in each loop, including time spent in inner loops.
- **Details section**—Lists detailed information such as the resulting nesting level of each loop after optimization, the compiler optimizations performed on each loop, and the CPU time spent in each loop. By comparing the loops in this table with those in the summary table, you can compare the loops actually executed to the original source code.
- **Vectorized loops section**—Provides a detailed account of the performance of vectorized loops (compiled at optimization level `-O2` or `-O3`). These measurements can help you determine the efficiency of vectorized loops.

Note

The compiler will not generate the information necessary for profiling loops unless you compile your program at optimization level `-O1` or higher.

If the `-nc` option is used when invoking CXpa, then any loops of a routine that are monitored, but not executed, are listed in the Monitored Loops Not Executed table.

During optimization, the compiler can significantly alter the order and flow of a loop to improve performance. In some cases, the compiler will interchange nested loops, separate complex loops into multiple smaller and simpler loops, or remove loops entirely.

CXpa enables you to track the performance of the original loops, even after all of the modifications listed in the Details section.

Summary section

The Summary section details the execution time and iteration count of each monitored loop in each routine. Each loop is identified by the source file line number on which it begins.

Figure 75
Summary section of the Loop Performance Analysis table

```
==============================================================================
                          Loop Performance Analysis
                               For: method_one
==============================================================================
Summary:
                          Times      Iteration Count          CPU Time
 Line                     Exec      Min      Max      Avg   (plus inner loops) PS
-------                 -------- -------- -------- -------- ------------------ --
    47                      1486      49      277      185            2.555238
    48                        10      49      277      148            2.568616
    50                      2970      49      277      204            2.503105
```

The Summary section in Figure 75 gives the performance statistics for the loops in the routine `method_one`. The columns are described below:

- **Line**—Lists the source line number of the loop.
- **Times Exec**—Lists the number of invocations of the loop.
- **Iteration Count**—Lists the minimum, maximum, and average iteration counts.
- **CPU Time (plus inner loops)**—Lists the total CPU time for the loop, including time spent inner loops.
- **PS**—Lists the profiling status and any additional notes for the loop. The possible annotations are listed in Table 12. If the column is empty, then the loop executed normally.

Details section

The Details section describes the loops actually executed. Due to optimizations this may differ from the loops in the original source code. Loops may have been interchanged, unrolled, or even split into multiple loops due to distribution or dynamic selection. An example of the Details section is given in Figure 76.

Note

If the original loop has been completely eliminated by the compiler, CXpa will not list the loop in the Details section of the Loop Performance Analysis table.

Figure 76
Details section of the Loop Performance Analysis table

```
Details:
                Optimization  Times        Iteration Count             CPU Time
 Line     NL      Report      Exec      Min      Max      Avg    (less inner loops) PS
------- --- ----------- -------- ------- ------- ------- ------------------ --
    48a   0 P                 10      49     277     148           0.006555
    47a   1 SM,I,D          1486      49     277     185           0.015695

    48b   0 P                 10      49     277     148           0.006823
    47b   1 SM,Hs           1486      49     277     185           0.036438
    50    2 S               2970      49     277     204           2.503105
```

Figure 76 shows an example of the Details section of the Loop Performance Analysis table. The columns are described below:

- **Line**—Lists the beginning line number of the loop. For compiler-generated loops (distributed loops), CXpa places a letter next to the line number to distinguish multiple loops derived from a single source loop.

  In the above example, the loops at lines 47 and 48 have been replaced with two loops each. Loops 48a and 47a occur together, and loops 48b and 47b occur together.

- **NL**—Lists the resulting nesting level of the loop after optimization. 0 indicates an outer loop, with each inner loop receiving a higher nesting level. In the example above, the loop at line 50 is nested inside of loop 47b, which is nested inside of loop 48b.

- **Optimization Report**—Lists the optimizations performed on the loop. The optimizations are abbreviated, and are listed in Table 13.

- **Times Exec**—Lists the number of invocations of the loop

- **Iteration Count**—Lists the minimum, maximum, and average number of iterations for the loop.
- **CPU Time (less inner loops)**—Lists the time spent executing the loop, not including inner loops. For the example above, the time spent in loop 48a was .006555 seconds, while the time spent in the inner loop (loop 47a) was .015695 seconds.
- **PS**—Lists the profiling status and any additional notes for the loop. The possible annotations are listed in Table 12. If the column is empty, the loop executed normally.

Table 13 describes the abbreviations CXpa uses to annotate the Optimization Report column of the Details section of the Loop Performance Analysis table.

Table 13 CXpa annotations for compiler optimizations

Mark	Transformation	Description of optimization performed by the compiler
`D`	Distributed	Split loop into multiple loops.
`Ds`	Dynamic selection	Generated multiple versions of the loop, with the best version chosen at run time when the trip count is known.
`Hs`	Hoisted	Moved a portion of the loop to the block before the loop.
`I`	Interchanged	Swapped an outer loop with an inner loop.
`V`	Fully vectorized	Vectorized the entire loop (the loop executes as a single vector instruction).
`P`	Parallel	Enabled the loop to execute in parallel.
`PS`	Parallel strip-mined	Enabled the loop to execute in parallel and strip-mined.
`P/V`	Parallel vectorized	Enabled the loop to be executed as vector instructions in parallel.
`pV`	Partially vectorized	Vectorized a part of the loop (but loop includes code that cannot be vectorized).
`S`	Scalar	Scalar optimization performed (as at `-no`, `-O0`, and `-O1`).
`SM`	Strip-mined vector	Converted loop to iterate through vector instructions.
`UL`	Uninstrumentable	Loop was not instrumentable by CXpa.
`Ur`	Unrolled	Replaced loop with a block of code for each iteration.

For full details on each of these optimizations, refer to either the *CONVEX FORTRAN Optimization Guide*, the *CONVEX C Optimization Guide*, or the *CONVEX Ada Optimization Guide*.

Vectorized loop section

The Vectorized loop section of the Loop Performance Analysis table summarizes the performance of vectorized loops in your program. An example of the floating point table is given in Figure 77.

Figure 77
Vectorized loop section of the Loop Performance Analysis table

```
                Static Profile                    Estimated  Mflops
              Vector  Vector  Chime  (less inner loops)  (plus inner loops)
  Line      NL Spills  Flops   Count      Avg       Peak      Avg       Peak    PS
  ------- --- ------ ------- ------- --------- --------- --------- --------- --
     31    0      0       5       4   26.667    31.250    26.667    31.250

     35    0      0       3       2   29.538    37.500    29.538    37.500
```

The columns of this section are briefly discussed below:

- **Line**—Lists the source line number of the loop.
- **NL**—Lists the resulting nesting level after optimizations.
- **Vector spill**—Lists the number of times a vector was replaced with a temporary value and then read in again.
- **Vector flops**—Lists the number of arithmetic vector floating point instructions in the loop. This count does not include load or store instructions.
- **Chime count**—Lists the number of chimes occurring within a given loop. Chime is an acronym for CHained Instruction MEasurement. The greater the chime count, the more resource conflicts, and therefore the less vector chaining.
- **Estimated Mflops**—Lists the number of millions of vector floating point operations per second. CXpa reports the peak performance and the average performance. Peak performance is the theoretical best that the code can do on that machine. Average is the measured performance.
  - (less children)—Does not include Mflops of nested loops.
  - (plus children)—Includes Mflops of nested loops.
- **PS**—Lists the profiling status and any additional notes for the loop. The possible annotations are listed in Table 12. If the column is empty, the loop executed normally.

A complete explanation of the Vectorized loop section is given in Appendix A, "Interpreting Mflops data."

Parallel Region Report

The Parallel Region Report provides performance data for monitored parallel regions. There is one Parallel Region Performance Analysis table for each executed routine with a monitored parallel region. The table is divided into two sections:

- **Summary section**—Lists the total time spent in each parallel region. The total CPU time, wall-clock time, and process virtual time (PVT) is given. The process virtual time is defined as the total CPU time during which at least one thread is executing in a parallel region.

  CXpa also lists the concurrency factor (CPU time divided by PVT time) for the parallel region. The concurrency factor measures the efficiency of code executing in parallel.

- **Details section**—Lists the CPU time, wall-clock time, and chore count of each thread in each parallel region. The chore count for a thread is the number of chores it performed inside the parallel region. Typically, a chore is a single iteration of a parallelized loop.

There are several ways of using the information to estimate parallel efficiency (the speed-up due to parallelism). For information on the calculations used in estimating parallel efficiency, refer to "Parallel efficiency theory," on page 123.

An understanding of parallel processing, threads, and parallel regions is necessary to fully utilize the Parallel Region Report. If you are already familiar with parallel processing and parallel regions, skip the next three sections. The report is described beginning on page 121.

CONVEX C Series parallel processing

A thread is an independent sequence of instructions that are executed by a single CPU.

Parallel optimizations performed by the compiler at the `-O3` level generate an executable that can be multithreaded. Each thread in a parallel region executes an identical set of instructions. This type of parallel processing is called symmetric and typically occurs when the compiler parallelizes a loop.

Asymmetric parallel processing occurs when a program spawns threads directly by using the `pfork` instruction. Asymmetric parallel processing cannot be profiled using CXpa.

Parallel regions

A loop that the compiler executes in parallel (typically around a nested vectorized loop) represents a set of chores. A *chore* is a single unit of work in a parallel region.

Typically, a parallel region consists of a parallel loop, and each chore is a single iteration of that loop. In this case, the number chores for the region is equal to the iteration count of the loop.

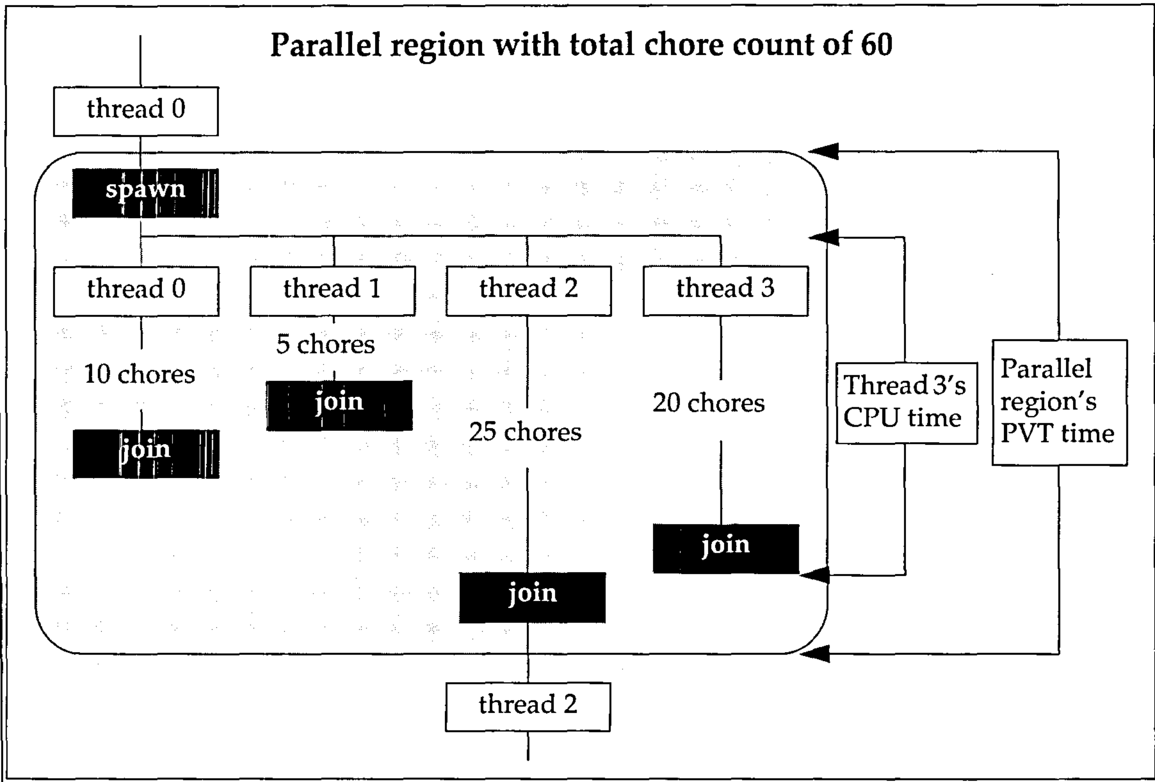
On a machine with multiple processors, the chores of a parallel region are divided among multiple threads. The number of threads available is dependent upon fixed scheduling, the number of CPUs, and CPU availability.

The number of iterations of a parallel loop represents the total number of chores for a parallel region. The more evenly these chores are distributed across the threads executing in the parallel region, the faster the time to solution. An uneven distribution indicates the total CPU time for the region, and therefore total wall clock time is longer because one thread is taking longer than the others.

Chore distribution is controlled by the processors themselves; you cannot directly affect the number of chores a given thread executes. CXpa reports this information to give you a better understanding of parallel region performance.

An example of a parallel region with a total chore count of 60 executing on a four-processor machine is shown in Figure 78.

Figure 78
Parallel region



In the above example, the parallel region has a chore count of 60. At the beginning of the parallel region, three additional threads are spawned, and the chores are divided up among four threads.

In this example, the chores are not evenly distributed between the threads. (Ideally, you would hope for a distribution of 15 chores per thread.)

The total CPU time for the region is the sum of all the CPU times for all threads executing in the parallel region.

The process virtual time (PVT) represents the CPU time where one or more threads were executing. If all threads execute simultaneously, the PVT time will be very close to the CPU time of a single thread.

As shown in Figure 78, each thread has its own CPU time and chore count.

Profiling with fixed scheduling

Fixed scheduling ensures that all CPUs of a machine are made available for each timeslice of the process.

In a time-sharing environment, you may want to enable fixed scheduling for two reasons:

- To make the profile of parallel programs more independent from resource conflicts with other processes
- To eliminate register bank conflicts between single-threaded programs (non-parallel) and other running applications

Fixed scheduling does *not* ensure that multiple threads are created. A single thread may complete all of the chores of a parallel region before another CPU could spawn a thread.

To invoke CXpa with fixed scheduling enabled, include the `-f` option on the CXpa command line, as shown in Figure 79.

Figure 79
Invoking CXpa with fixed scheduling enabled

```
% cxpa -f a.out
```

Once you have enabled fixed scheduling, all processes run under CXpa will run with fixed scheduling.

Summary section

The Summary section provides performance measurements for an entire parallel region. It is a summary of the time spent by individual threads executing in the region. The section includes:

- Line number at which the parallel region begins
- Number of times the parallel region was executed
- CPU time
- Wall-clock time
- Process virtual time (PVT)
- Concurrency factor (CPU time / PVT time)

Figure 80 shows an example of the Summary section.

Figure 80
Summary section of the Parallel Region Analysis table

```
==============================================================================
                    Parallel Region Performance Analysis
                              For: method_one
==============================================================================
Summary:
           Times            All Regions            Process Virtual
 Line      Exec        CPU Time   Wall Clock Time    Time (PVT)      CPU/PVT   PS
-------  --------   ------------ --------------- ---------------  ---------  --
   48          10      0.033652        0.080898        0.072091       0.467
   48          10      4.746450        7.145293        4.768849       0.995
```

In Figure 80, the performance data is shown for the parallel region in the routine `method_one`.

The columns of the section are described below:

- **Line**—Lists the line number for the beginning of the parallel region. This is the original line number of the source code.

  In the example above, statistics are given for two parallel regions, both of which start at line 48. In this example, the loop at line 48 was first distributed by the compiler into two separate loops. Each of these loops executed as independent parallel regions. The parallel regions describe two completely different loops, but both originated from the loop at line 48 of the source code.

- **Times Exec**—Lists the number of executions of the region.

- **CPU Time**—The total time spent in the parallel region.
- **Wall Clock Time**—The total wall-clock time during execution of the parallel region.
- **Process Virtual Time**—Total PVT for the parallel region. The PVT represents total time spent in user mode, not including time spent in kernel mode or waiting for I/O.
- **CPU/PVT**—The concurrency factor for the parallel region. The concurrency factor represents the degree to which the CPUs are being utilized. The closer the concurrency factor is to the number of processors on the machine, the better.

Details section

The Details section describes the performance of the threads that executed each parallel region. Each line represents a single thread. For a two-processor machine, each parallel region would have two lines describing thread performance for that region.

An example of the Details section is given in Figure 81.

Figure 81
Details section of the Parallel Region Analysis table

```
Details:
                          By Region
 Line                CPU Time   Wall Clock Time    Chore Count            PS
-------            ------------ ---------------   -------------           --
   48                0.033425         0.079406             2033
                     0.000227         0.003157               11
   48                3.866300         7.013111             1749
                     0.880150         3.763494              295
```

The columns of the Details section are described below:

- **Line**—The starting line number for the parallel loop as it exists in the original source code.
- **CPU Time**—The total CPU time spent by each thread in the parallel region.
- **Wall Clock Time**—The total wall-clock time spent by each thread in the parallel region.

- **Chore count**—The total number of chores executed by the thread. In the example above, the chore counts are unevenly distributed between the two threads for both of the parallel regions. This may indicate room for improvement in parallel optimization.
- **PS**—The process state of the parallel region. CXpa annotates paused parallel regions using this column. Possible annotations are given in Table 12.

Further interpretation of the profiling report can be done by applying parallel efficiency theory.

Parallel efficiency theory

There are two major methods of calculating parallel efficiency. The first method can be used when you have profiled the program at optimization level `-O3` and do not want to profile the program at a lower optimization level.

This method uses the concurrency factor (CPU time for the parallel region divided by the PVT). The concurrency factor is divided by the number of processors to get an approximation of the parallel efficiency.

Concurrency per processor method

The following equation demonstrates how to calculate the approximate parallel efficiency:

$$\frac{\textit{Concurrency factor}}{\textit{Number of processors}} \times 100\% \cong \textit{Parallel efficiency}$$

For example, on a two-processor machine, the first parallel region at line number 48 (shown in Figure 80), the equation becomes:

$$\frac{0.467}{2} \times 100 \cong 23.35$$

The parallel efficiency for the first parallel region at line 48 is 23%. This could be used as a basis for comparison if the parallel region was modified and profiled again.

Serial CPU time versus parallel PVT method

The second method requires you to profile the program without parallel optimizations (at optimization level `-O2` or lower). This compares the serial CPU time to the parallel PVT time to determine an approximation of the parallel efficiency.

The following formula can be used after you have compiled and profiled a parallel region (such as a loop) at optimization level `-O2` or lower:

$$\frac{Serial\ CPU\ time}{Parallel\ PVT} \times 100\% \cong Parallel\ efficiency$$

Basic Block Report

If you compile your program with the `-pab` option, you can profile the basic blocks of your program.

The Basic Block Report provides performance data for the basic blocks of each monitored routine. A basic block is a section of code that has a single entry point and a single exit point. This report consists of one Basic Block Performance Analysis table for each executed routine with monitored basic blocks.

The Basic Block Performance Analysis table provides the percentage of total block executions spent executing each basic block.

The Basic Block Performance Analysis table is made up of two sections:

- **Summary**—Lists the percentage of basic blocks in the routine that were executed at least once.
- **Details**—Lists the line number, PC Value, and number of times executed for each basic block.

Because the Summary section consists of a single line, both the Summary section and the Details section are illustrated in Figure 82.

Figure 82
Basic Block Performance Analysis table

```
==============================================================================
                    Basic Block Performance Analysis
                           For: method_two
==============================================================================
Summary:
                48.1% of all basic block(s) executed at least once.
Details:
                                      Times        % of
 Line           PC Value              Exec      Total Exec                   PS
-------       -----------           ---------  ----------                    --
    67         0x80001eda                1711        0.1%
    67         0x80001c4c                   0        0.0%
    67         0x80001e8e                  16        0.0%
    67         0x80001eca                1711        0.1%
    67         0x80001c34                   0        0.0%
```

Figure 82 shows the Basic Block Performance Analysis table for the `method_two` routine. The Summary section lists the percentage of basic blocks executed in the routine. This helps to identify routines containing a significant amount of code that is never executed.

The columns of the Details section are described below:

- **Line**—Lists the starting line number for the basic block. This helps track basic blocks back to the original source code.

  It is possible for many basic blocks to begin at the same line number, as in the preceding example.

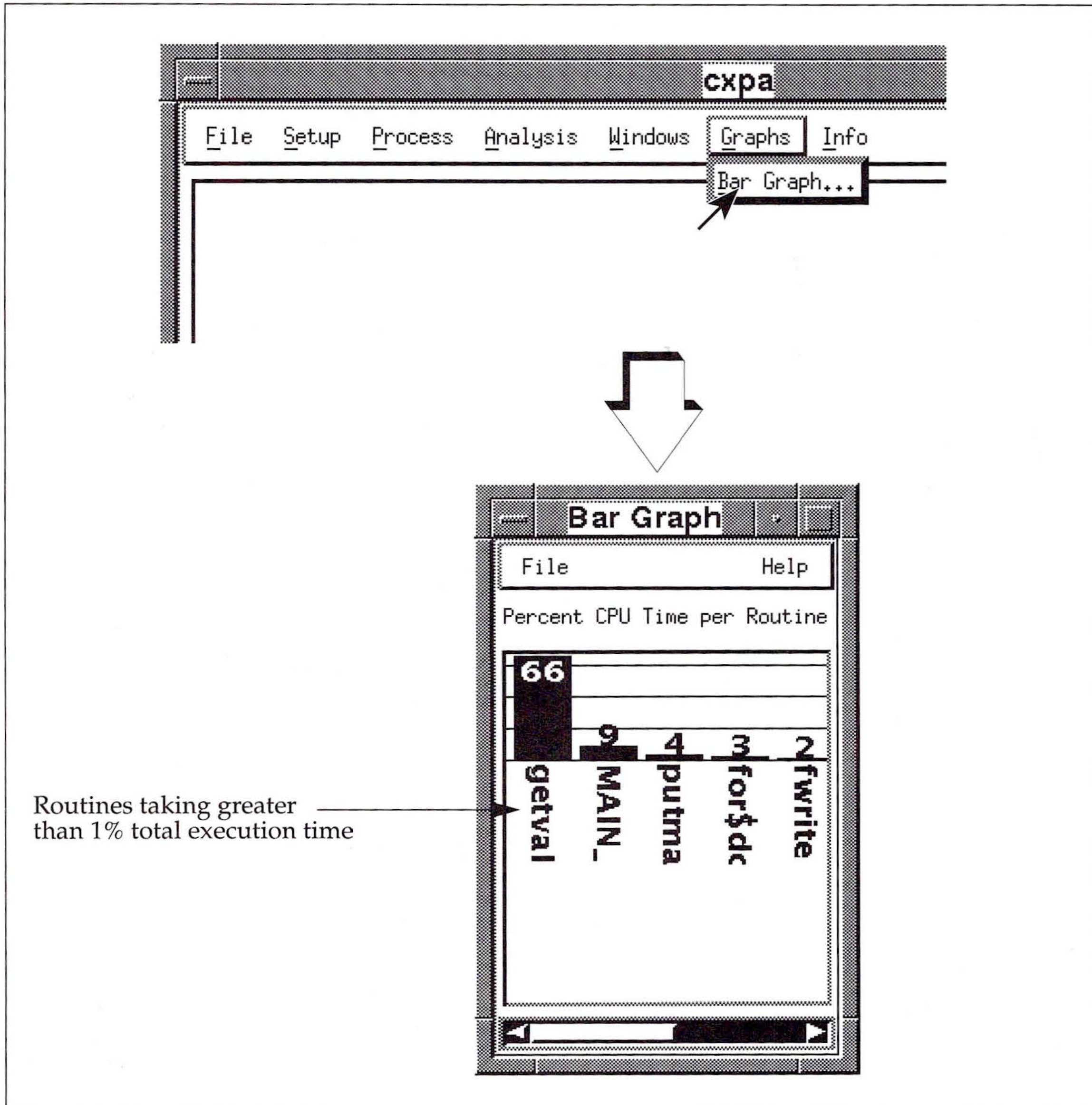
- **PC Value**—Lists the starting address of the basic block. Because of optimizations, the PC value is often more accurate than the original starting line number.
- **Times Exec**—Lists the total number of times the basic block was executed.
- **% of Total Exec**—Lists the percentage of total block executions in this routine.
- **PS**—Lists the profiling status of the routine. CXpa uses this column to note the state of a routine or its instrumentation. Table 12 describes the possible annotations used in the PS column.

Using the Bar Graph window

The Bar Graph window displays a bar graph comparing the total CPU time in seconds of routines taking more than 1% of total program execution time.

The bar graph is only available through the X interface. Open the bar graph by selecting the Bar Graph option from the Graphs menu of the CXpa window, as shown in Figure 83.

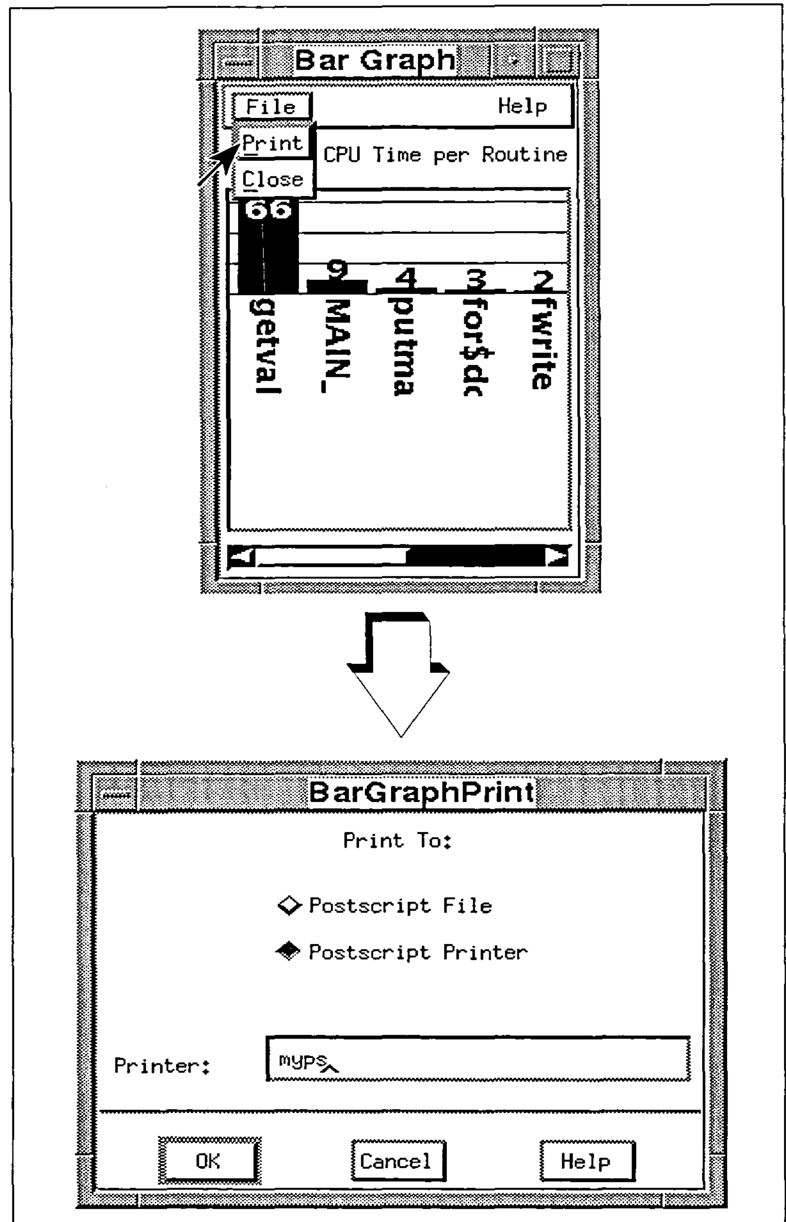
Figure 83
Opening the Bar Graph window



Printing the bar graph

To send the bar graph to a PostScript printer or file, select the Print option under the File menu of the Bar Graph window. The Bar Graph Print dialog box opens, as shown in Figure 84.

Figure 84
Print graph dialog box



Printing to a printer

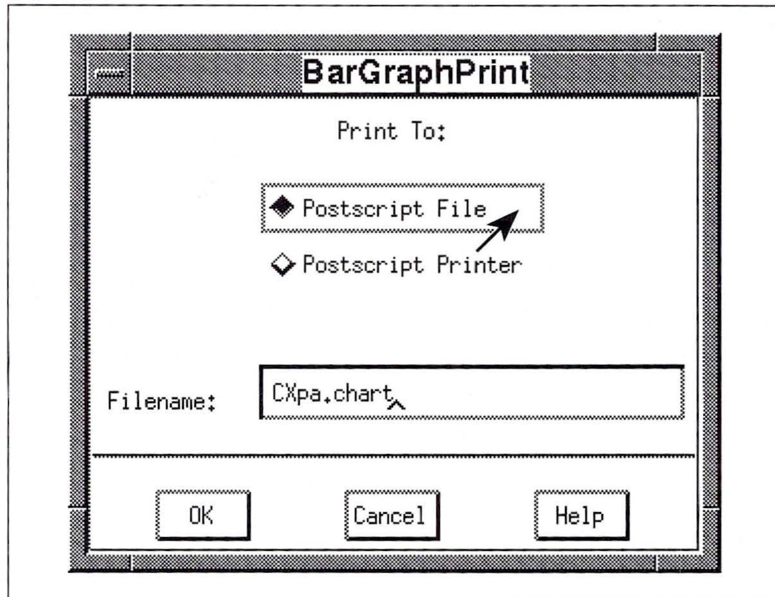
By default, the print dialog box is set to print to the printer specified by your `$PRINTER` environment variable (if the variable is not set, no printer is specified).

To print the graph to this printer, click the OK button. To print the graph to a different printer, enter the name of the printer in the text field, then click the OK button.

Printing to a file

If you want to print to a file rather than a printer, click the Postscript File button, as shown in Figure 85.

Figure 85
Printing the bar graph to a file



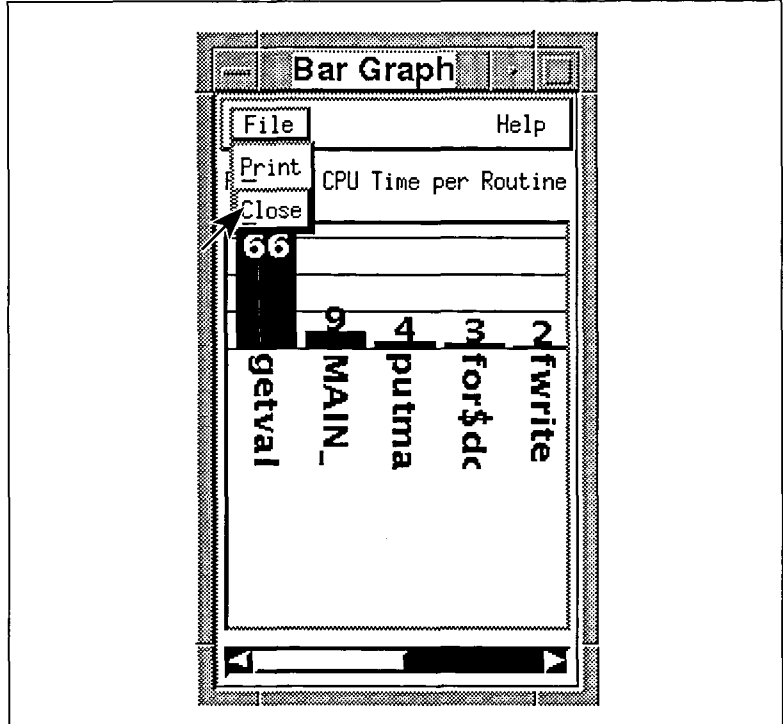
When you do so, CXpa automatically fills the Filename text field with the default file name of CXpa.chart. You can edit this text field to save the graph to a different file.

Click the OK button to print the graph to the file.

Closing the bar graph

To close the bar graph, select the Close option from the File menu, as shown in Figure 86.

Figure 86
Closing the bar grapht



Interpreting Mflops data

A

The Vectorized loop section of the Loop Performance Analysis table includes various metrics for vectorized loops, such as the chime count and estimated millions of floating point operations per second (Mflops).

This appendix introduces the metrics included in the table and explain how these metrics are calculated. The information presented in this appendix is not intended as a guideline for improving vector performance.

All performance data for vectorized loops is based solely on *vector* floating point operations. This is because the CONVEX architecture, compilers, libraries, and tools are designed for vector computing.

The information in the Vectorized loop section of the Loop Performance Analysis table represents the theoretical best that your code can do on a given machine, and is an estimate of how the code actually performed. The measurements are not intended to be used as a comparison to other architectures, or as an evaluation of what constitutes good vector performance.

The first section briefly introduces the Vectorized loop section and vector chaining. The remaining sections explain how CXpa calculates chime counts and vector Mflops.

The topics covered include:

- Vectorized loop section
- Vector chaining
- Chime counts
- Vector Mflops

Vectorized loop section

The last section of the Loop Performance Analysis table displays metrics and performance measures for vectorized loops.

The Vectorized loop section is only generated for executed monitored loops that include vector instructions. The Mflops data is only generated for loops containing vector floating point operations. If the vectorized loop section does not appear in the table, no vectorization was done by the compiler for that loop or nest of loops.

An example of the Vectorized loop section is shown in Figure 87.

Figure 87
Vectorized loop section of the Loop Performance Analysis table

```
                Static Profile                 Estimated  Mflops
             Vector  Vector  Chime  (less inner loops)  (plus inner loops)
 Line     NL Spills  Flops   Count     Avg        Peak       Avg        Peak    PS
------- --- ------ ------- ------- --------- --------- --------- --------- --
     31   0      0       5       4    26.667    31.250    26.667    31.250

     35   0      0       3       2    29.538    37.500    29.538    37.500
```

The parts of the Vectorized loop section are briefly discussed below:

- **Line**—Source line number of the loop.
- **NL**—Nesting level of the loop.
- **Vector spill**—Number of times a vector was written out of a register only to be read in again later.
- **Vector flops**—Number of arithmetic vector floating point instructions in the loop. This count does not include vector load or store operations.
- **Chime count**—Number of chimes occurring within a given loop. Chime is an acronym for CHained vector Instruction MEasurement. The greater the chime count, the more contention for available vector functional units, and therefore the less vector chaining.

  Chimes, functional units, and vector chaining are explained in the following sections.

- **Estimated Mflops**—Estimated number of millions of vector floating point operations per second. There are two subcategories:
  - **Avg**—Average number of Mflops. This is based on the average CPU time per invocation of the loop.
  - **Peak**—A theoretical maximum number of Mflops for this loop on this architecture.

  These values are given for each loop with and without including nested loops.

  Dividing average Mflops by Peak Mflops gives the percent utilization of the vector unit. However, full utilization of the vector unit does not imply well-written code or fastest possible time to solution. It simply represents the best the existing code can do on that machine.

Vector chaining

This section explains how vector chaining occurs and describes two situations that prevent vector chaining. This information will make it easier to understand how CXpa calculates chimes and how to utilize the vector units more effectively.

The CONVEX C Series architecture enables certain vector operations to be performed concurrently, or to be overlapped with other vector operations. This ability, known collectively as vector chaining, can enable vector operations to occur simultaneously, rather than one after another.

This concurrency offers significant performance improvement. The chime count (chained instruction measurement) for a loop is a theoretical measure of the efficiency of vector chaining.

Vector chaining and functional units

On a C Series machine, the vector processor enables the compiler to load 128 elements of an array into a vector register, and then operate on all 128 elements as a unit. Because of the speed at which vector register operations are performed, such vector operations are often assumed to be happening to all 128 elements simultaneously.

A vector functional unit operates on one element of a vector register at a time. A vector functional unit is an independent part of a CPU that performs a designated operation, such as addition or multiplication.

Because of this, the the first element from the result of a vector operation is ready before the second, which is ready before the third, and so on.

Vector chaining takes advantage of this by using the output of one functional unit as the input to the next functional unit.

Consider the following two vector instructions:

```
V2 = V1 + V0
V4 = V3 * V2
```

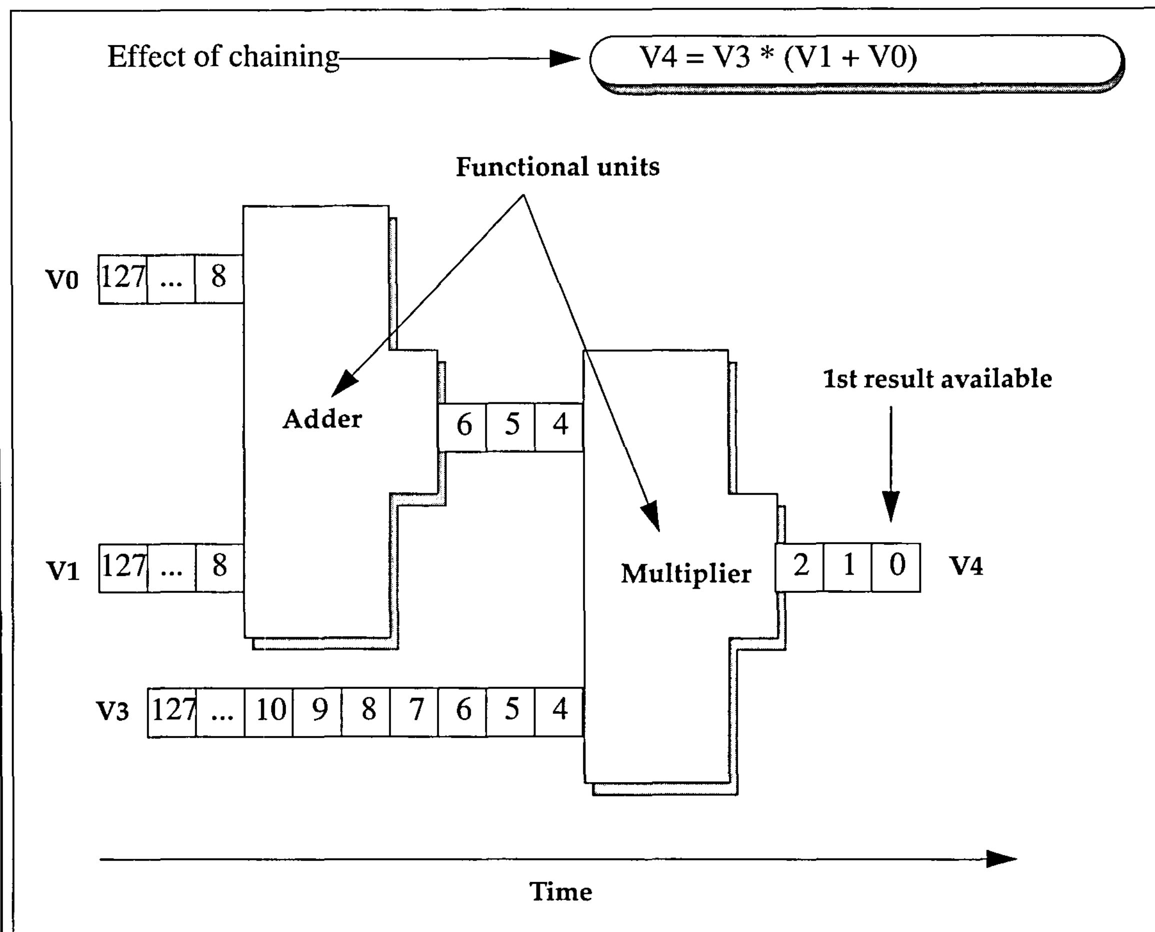
In this example, each element of V0 is added to the corresponding element of V1, and the result is stored into the corresponding element of V2. Then each element of V2 is multiplied with the corresponding element of V3, and the result is stored in V4. Without vector chaining, all of the elements of the first operation (the addition) would have to be completed before the multiplication operation could begin.

With vector chaining, the output of the addition operation can be fed directly into the multiplication operation. This is possible because the add operation and the multiply operation are calculated by two different functional units. This could be represented as:

```
V4 = V3 * (V1 + V0)
```

This is demonstrated in Figure 88.

Figure 88
Effect of vector chaining



As shown in Figure 88, V0 and V1 are both passed to the Adder functional unit. As the results are output, they are fed directly into the Multiplier functional unit along with V3. As the results are generated, they are stored into V4.

If the values in V4 were to be stored into an array, this store could be chained with the previous two instructions. On the CONVEX C Series architecture, loading and storing of vector registers is handled by a third functional unit, called the Loader).

The amount of vector chaining available depends on the underlying architecture, as discussed in the next section.

Functional unit reservation

Vector chaining is only possible when a functional unit is available. CONVEX C Series machines have three vector functional units per CPU. Each functional unit has a specific set of vector instructions it is capable of processing.

When a vector instruction cannot be chained with the previous instruction because the appropriate functional unit is not available, functional unit reservation occurs, and chaining is prohibited.

On the C200 and C3200 architectures, the following vector instructions could not be chained:

```
V2 = V1 + V1
V4 = V3 + V2
```

For these architectures, the add operation can only be performed by the Adder functional unit. The first add operation occupies the Adder functional unit. The second add operation must wait until the Adder is again available before beginning.

The ability of the CPU to chain vector instructions together depends on what vector operations can be performed by each vector functional unit of the CPU.

The vector functional units found in the CONVEX C Series architectures are listed in Table 14. Names (Adder, Multiplier, Divider, or Loader) have been given to the functional units to identify the types of vector operations they perform. The table marks with a check each vector operation each vector functional unit can perform.

You can use this table to determine whether functional unit reservation could be prevented by different instruction scheduling.

Table 14 C Series vector functional units

Generic operation	C200 and C3200 Vector func. units			C3400 Vector func. units			C3800 Vector func. units		
	Adder	Multiplier	Loader	Adder	Multiplier	Loader	Divider	Multiplier	Loader
Add/subtract	✔			✔	✔		✔	✔	
Logical compare	✔			✔	✔		✔	✔	
Bit count	✔			✔	✔		✔	✔	
Shift	✔			✔	✔		✔	✔	
Multiply		✔			✔			✔	
Divide		✔			✔		✔		
Type conversion		✔			✔		✔	✔	
Edit	✔			✔	✔		✔	✔	
Load/store			✔			✔			✔
Square root		✔			✔		✔		

If an operation can be performed on two functional units, it is said to be redundant. For example, the add operation is redundant on C3400 and C3800 Series CPUs.

On the C3400 Series CPU, redundant operations are passed to the Adder functional unit, because it does not have any operations that it alone can perform (a subsequent add could be passed to the Multiplier).

On the C3800 Series CPU, redundant operations are passed to the Multiplier functional unit (it has fewer operations that only it can perform).

Interpreting Mflops data

Register bank reservation

Vector chaining can also be prevented when a vector instruction accesses a vector register that is not available. This is known as register bank reservation.

A register bank holds a pair of vector registers. The pairs are as follows:

- V0 and V4
- V1 and V5
- V2 and V6
- V3 and V7

At most, two reads and one write can be made to a register bank during a single clock cycle. Vector instructions cannot be chained together if this would cause more read and writes than possible to a single register bank.

On C3400 and C3800 Series CPUs, two reads to the same register bank count as a single read.

For example, the following two instructions cause register bank reservation on all CONVEX C Series architectures:

```
V0 = V2 + V1
V4 = V3 * V1
```

The first instruction requires one write to the V0/V4 register bank. The second instruction cannot be chained with the first because of the additional write to the V0/V4 register bank.

Chimes

A loop whose vector instructions are chained together executes more quickly because chaining enables multiple instructions to occur simultaneously.

A CHained Instruction MEasurement, or chime, represents a vector instruction that could not be chained with the previous instructions, due to functional unit reservation or register bank reservation. Every vectorized loop will have a minimum of one chime.

Ideally, each vector instruction would be chained together, thus resulting in a single chime. However, this is seldom possible. The higher the chime count for a loop, the less vector chaining achieved. Therefore, the number of chimes in a loop, called the chime count, can be an effective performance indicator for vectorized loops.

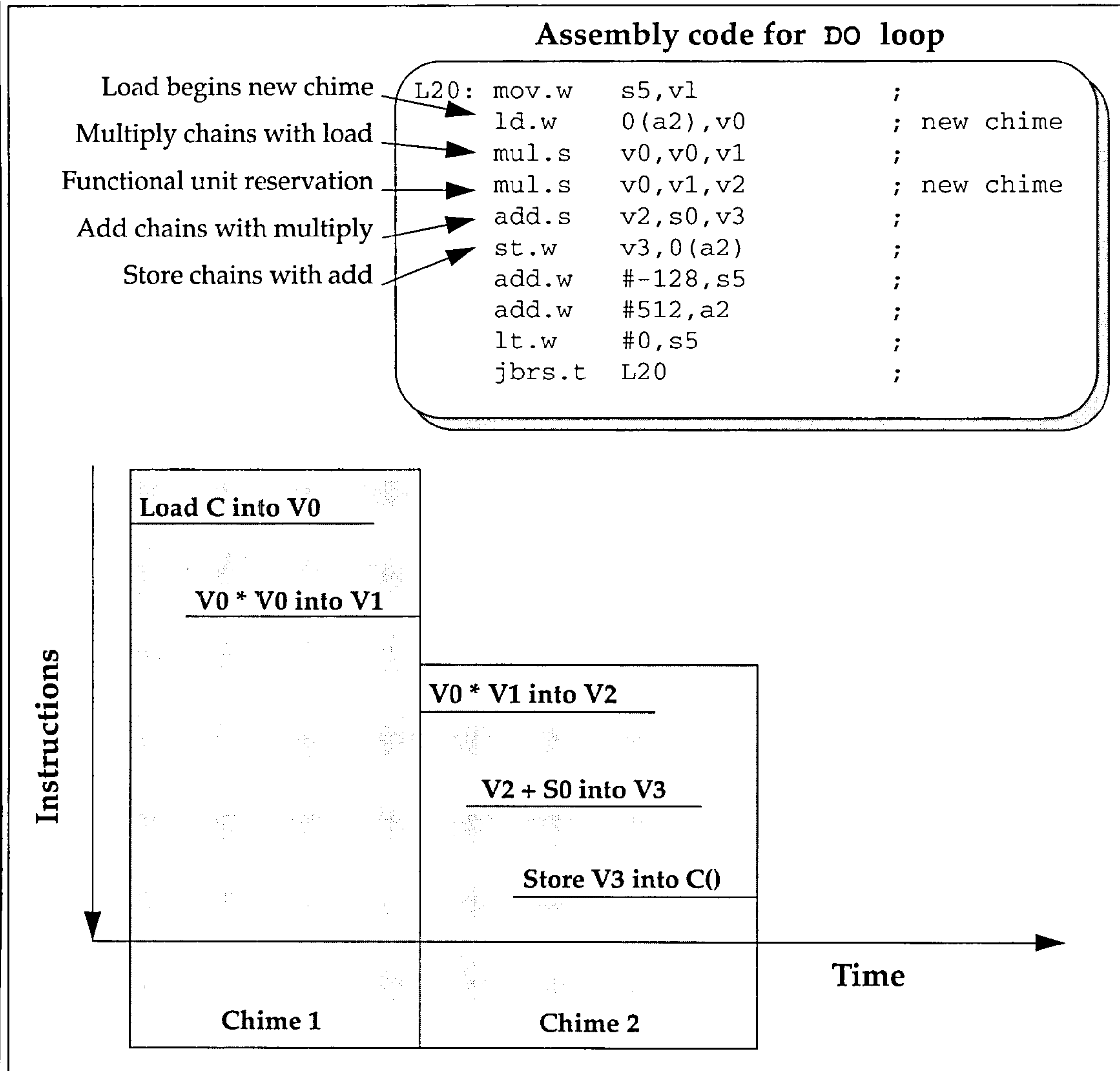
Calculating chimes

To demonstrate the calculation of a loop's chime count, consider the following FORTRAN loop:

```
DO I = 1, 256
   C(I) = C(I) * C(I) * C(I) + 3.0
ENDDO
```

The chime count for this loop is shown in Figure 89.

Figure 89
Chime calculation on C3200 Series



In Figure 89, the first multiply instruction (`mul.s v0,v0,v1`) can chain with the load instruction. The second multiply cannot chain with the first, because the C3200 Series CPU has only one functional unit to perform the multiply operation. The second multiply cannot begin until the first multiply completes and the multiply functional unit is again available.

Vector Mflops

Most scientific and engineering applications use floating point numbers. Due to differences in hardware and software, the performance of such code is better represented by millions of floating point operations performed per second (megaflops or Mflops) than by millions of instructions per second (Mips).

A floating point operation is simply an arithmetic instruction operating on one or more floating point numbers.

CXpa distinguishes two types of floating point operations:

- **Scalar**—Operating on a single floating point number. For example, taking the square root of 3.14.
- **Vector**—Operating on a vector of floating point numbers. For example, subtracting 3.14 from the first 128 elements of an array of real numbers.

CXpa reports both the estimated vector Mflops and the peak vector Mflops. These values are reported for the loop with and without nested loops.

The estimated number of vector Mflops indicates how well the loop ran on that machine. The closer the estimated Mflops is to the peak, the greater the utilization of the vector functional units available on the machine.

Calculating estimated Mflops

The estimated flops for a loop is calculated by dividing the total number of executed vector floating point operations in the loop by the total number of seconds spent in the loop.

The estimated flops for a loop is calculated as:

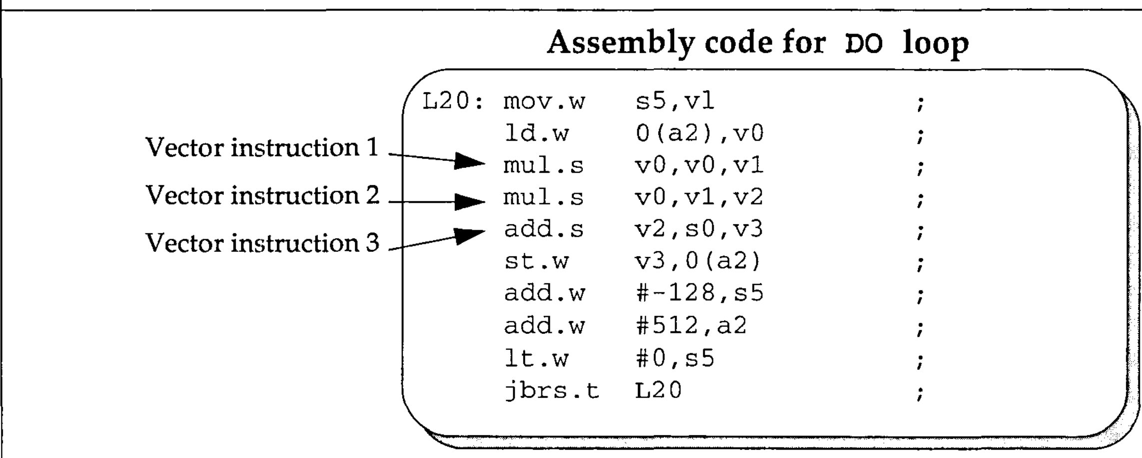
$$estimated_flops = \frac{floating_point_operations(flop) \times trip_count}{total_time(sec)}$$

Returning to the previous example, you can calculate the estimated Mflops for the loop:

```
DO I = 1, 256
   C(I) = C(I) * C(I) * C(I) + 3.0
ENDDO
```

There are 256 iterations of the loop, so the trip count is 256. You can determine the number of vector floating point operations in the loop by looking at the assembly code. As shown in Figure 90, there are three vector floating point instructions inside the loop.

Figure 90
Calculating the vector flops of a loop



Note

The vector floating point instruction count does not include load/store operations, nor does it subtract for masked components (assumes no mask).

Assume that the loop took .000026 seconds to execute on a C200. The calculation becomes:

$$estimated_flops = \frac{3 \text{ flop} \times 256}{0.000026 \text{ sec}} = 29538462 \text{ flop/sec}$$

$$estimated_Mflops = 29.538$$

The estimated Mflops for this loop would be 29.538 Mflops.

Calculating peak Mflops

Peak Mflops represents the maximum floating point performance for a particular loop on a particular machine. To calculate the peak Mflops, use the following equation:

$$peak_flops = \frac{floating_pt_ops}{chime_count} \times \frac{1}{machine_clock_cycle(sec)}$$

The clock rate of the machine rate per head is as follows:

- C200—40 ns
- C3400—40 ns
- C3800—16.67 ns

For the loop used in the previous example, the peak rate becomes:

$$peak_flops = \frac{3 \text{ flop}}{2} \times \frac{1}{40 \times 10^{-9} \text{ sec}}$$

$$peak_Mflops = 37.5$$

The peak Mflops for this loop is 37.5 Mflops.

Index

Symbols

$DISPLAY 13
$PAGER 10
$PRINTER 106, 128
$TERMCAP 10
.cxpainit initialization file 9

A

about cxpa 26
About CXpa dialog box 26
About CXpa option 26
Ada, compiling limitations 40
add path 75
Adder functional unit 136
Analysis menu
  Analyze All option 102
  Analyze Loops option 102
analyze 103
Analyze All option 102
analyze loop 103
Analyze Loops option 102
annotations
  monitor points 42, 68
  profiling status 109
arguments to the program
  entering from the X interface 91
ASCII terminals 9
assistance xviii
associated documents xviii
audience xv

B

B monitor point annotation 68
b monitor point annotation 68
bar graph
  closing 129
  opening 126
  printing 127
Bar Graph option 126
Bar Graph Print dialog box 127
Bar Graph window 126
Basic Block Performance Analysis table 124
Basic Block Report
  Basic Block Performance Analysis table 124
  overview 124
basic blocks. *see* blocks
bibliography. *see* associated documents
blocks
  as a region type 34
  basic, defined 2
  profiling limitation 39

C

-c compiler option 35
Change Search Path dialog box 77
child routines, defined 107
chimes 138
chore, defined 118
command files 9
command syntax xvi
Command Window 12
commands
  about cxpa 26
  add path 75
  analyze 103
  analyze loop 103
  continue 96
  cxpa 9, 13
  deselect 66
  deselect all 67
  help 27, 29
  info path 75
  info status 24
  list 71
  list monitors 71
  monitor 59
  monitor at 63
  monitor block all 57
  monitor block at 64
  monitor block in 61
  monitor loop all 57
  monitor loop at 64
  monitor loop in 61
  monitor pregion all 57
  monitor pregion at 64
  monitor pregion in 61
  monitor routine all 57
  more 10
  path 75
  pause 95
  quit 32
  rerun 99
  run 93
  set pdf 88
  source 9
  stop 97

Index

compiler
  Ada 39
  C 39
  FORTRAN 39
  options
    -c 35
    described 34
    -L 38
    -O1 34
    -O2 124
    -O3 37
    -p 34
    -pa 34, 39
    -pab 34, 39
    -par 34
    -pb 34
    -pg 34
compiling
  and linking in one step 35
  and linking separately 35
  FORTRAN limitations 39
  limitations 37
  linking with uninstrumented libraries 37
  OPTIONS statement 39
  options. *see* compiler, options
  overview 34
  with both -pa and -pab 39
concurrency factor
  and parallel efficiency 123
  defined 122
constant-width font xvii
contact utility xix
continue 96
continuing profiling 96
  from the command line 96
  from the X interface 96
Control Center dialog box 90
conventions xvi
CRT interface
  described 9
  key bindings 11
CTRL-c to pause profiling 95
CXpa
  command files 9
  compiling. *see* compiling
  CRT interface. *see* CRT interface
  features 3
  general steps 6
  getting started 5
  initialization files 9
  introduction to 1
  invoking 9
  monitor points. *see* monitor points
  monitor routines 36
  options
    -f 120
    -nc 112
    -ncg 110
    -nw 9
    -path 81
    -pdf 89
  quitting 32
  running the program. *see* running the program
  search path. *see* search path
  status information 24
  version information 26
  windows
    Bar Graph 126
    Command 12
    CXpa 13
    Help 28
  X interface. *see* X interface
cxpa 9, 13
CXpa Status dialog box 25
CXpa window 13
CXpa.report, as default report name 106

D

D optimization abbreviation 115
deselect 66
deselect all 67
dialog boxes
  About CXpa 26
  accessing help 27
  Bar Graph Print 127
  Change Search Path 77
  Control Center 90
  CXpa Status 25
  Info Search Path 76
  Monitor Point Information 69
  Print 105
  Quit CXpa 32
  Set Monitor Points 48
  Set PDF 87

disabling monitor points 66
Divider functional unit 136
Ds optimization abbreviation 115
Dynamic Call Graph table 110

E

e profiling status 109
ellipsis
  horizontal xvii
  vertical xvii
enabling monitor points
  from the command line 55
  from the X interface 45
enter xvii
environment variables
  $DISPLAY 13
  $PAGER 10, 103
  $PRINTER 106, 128
  $TERMCAP 10
executable. *see* files, executable

F

-f CXpa option 120
File menu
  Print option 105
  Quit option 32
  Search Path option 77
  Set PDF option 86
files
  command 9
  executable
    compiling 34
    specifying 84
  initialization 9
  listing source 72
  object 35
  performance data. *see* PDF
  setting search path for 74
  source 68, 71, 72, 74
fixed scheduling and CXpa 120
fonts
  constant-width xvii
  italics xvii
FORTRAN, compiling limitations 39
functional units
  defined 133
  described 136
  reservation 136

G

generating reports
  from the command line 103
  from the X interface 102
getting started 5
Graphs menu
  Bar Graph option 126

H

help 27, 29
Help button 27
Help menu
  Help option 27
Help option 27
Help Window
  example of 28
  see also online help
help, by phone xviii
help. *see* online help
hot routines, defined 107
how to use this book xv
Hs optimization abbreviation 115

I

I optimization abbreviation 115
Info menu
  About CXpa 26
  Monitor List 69
  Search Path 76
  Status 25
info path 75
Info Search Path dialog box 76
info status 24
initialization files 9
instrumentation, incorrect 109
instrumented code 34
introduction 1
invoking CXpa 9
italic font xvii

K

key bindings
  Command Window 12
  CRT interface 11
keyboard shortcuts 13

L

-L compiler option 38
L monitor point annotation 68
l monitor point annotation 68
linking. *see* compiling, linking
list 71
list monitors 71
listing
  current source file 71, 72
  monitor points
    from the command line 71
    from the X interface 69
    see also monitor points, listing
  source files 72
Loader functional unit 136
Loop Performance Analysis table 112
Loop Report
  Loop Performance Analysis table 112
  overview 112
loops
  as a region type 34
  uninstrumented 109

M

m abbreviation for milliseconds 102
m profiling status 109
Mflops
  calculating estimate 141
  calculating peak 142
  defined 140
mnemonics 13
monitor 59
Monitor All option 45
monitor at 63
monitor block all 57
monitor block at 64
monitor block in 61
Monitor List option 69
monitor loop all 57
monitor loop at 64
monitor loop in 61
Monitor None option 67
Monitor option 48
Monitor Point Information dialog box 69
monitor points
  annotations for 42, 68
  defined 42
  disabled vs enabled 43
  disabling 66
  enabling
    from the command line 55
    from the X interface 45
  listing
    and search paths 68
    overview 68
  overview 42
  types of 42
monitor pregion all 57
monitor pregion at 64
monitor pregion in 61
monitor routine all 57
monitor routines 36
more 10
more used for paging 103
Multiplier functional unit 136

N

-nc CXpa option 112
-ncg CXpa option 110
notational conventions xvi
notes xvii
-nw CXpa option 9

O

-O1 compiler option 34
-O2 compiler option 124
-O3 compiler option 37
object file. *see* files, object
online help
  accessing 27
  on a topic 29
  searching 30
optimization
  abbreviations in Loop Report 115
  effect on loops 112
OPTIONS statement limitation 39
options, compiler
  see compiler, options
ordering documents xviii

P

-p compiler option 34
P monitor point annotation 68
p monitor point annotation 68
P optimization abbreviation 115
p profiling status 109

P/V optimization abbreviation 115
-pa 34, 39
-pab 34, 39
-par 34
parallel efficiency theory 123
parallel processing
  asymmetric 117
  explained 117
  symmetric 117
Parallel Region Performance Analysis table 117
Parallel Region Report
  overview 117
  Parallel Region Performance Analysis table 117
parallel regions
  as a region type 34
  compiling for use with CXpa 39
  defined 118
parent routines, defined 107
path 75
-path CXpa option 81
pause 95
pausing profiling 94
  from the command line 95
  from the X interface 94
-pb compiler option 34
PDF
  and the search path 74
  defined 85
  specifying
    from the command line 88
    from the shell prompt 89
    from the X interface 86
-pdf CXpa option 89
performance data file. *see* PDF
pfork instruction and CXpa 117
-pg compiler option 34
Print dialog box 105
Print option (CXpa window) 105
printing reports
  from the command line 104
  from the X interface 105
process input and output
  and the command window 93
  and the CRT interface 93
  and the X interface 93
Process menu
  Rerun option 98
  Run option 90
process virtual time (PVT)
  defined 117
  example 119
profilers
  CXpa 3
  defined 1
  prof 1
profiling
  and CXpa 3
  benefits 2
  by statistical sampling 1
  compiling for 34
  continuing. *see* continuing profiling
  overview 1
  pausing. *see* pausing profiling
  regions 34
  starting 92
  steps 6
  stopping. *see* stopping profiling
profiling status annotations 109
prompt xvii
PS optimization abbreviation 115
pV optimization abbreviation 115
PVT. *see* process virtual time

Q

quit 32
Quit CXpa dialog box 32
Quit option 32
quitting CXpa 32

R

R monitor point annotation 68
r monitor point annotation 68
recursive routines, and Dynamic Call Graph 111
regions, types of 34
register bank reservation 138
related documents xviii
reports
  Basic Block Report. *see* Basic Block Report
  generating
    from the command line 103
    from the X interface 102
  Loop Report. *see* Loop Report
  Parallel Region. *see* Parallel Region Report
  printing
    from the command line 104
    from the X interface 105
  Routine Report. *see* Routine Report
rerun 99
Rerun option 98
rerunning the program 98
  and input/output redirection 98
  from the command line 99
  from the X interface 98
reservation of functional units 136
Routine Performance Analysis table 107
Routine Report
  Dynamic Call Graph table 110
  overview
  Routine Performance Analysis table 107

routines
  as a region type 34
  calling unmonitored routines 38
  child, defined 107
  hot, defined 107
  parent, defined 107
  recursive and Dynamic Call Graph 111
`run` 93
Run option 90
running the program 89
  from the command line 93
  from the X interface 90

S

`S` optimization abbreviation 115
search path
  adding to
    from the command line 75
    from the shell prompt 81
    from the X interface 78
  changing
    from the X interface 77
  listing
    from the command line 75
    from the X interface 76
  overview 74
  removing directory from
    from the X interface 79
  setting
    from the command line 75
Search Path option 76, 77
Set Monitor Points dialog box 48
`set pdf` 88
Set PDF dialog box 87
Set PDF option 86
Setup menu
  Monitor All option 45
  Monitor None option 67
  Monitor option 48
shortcuts, keyboard 13
`SM` optimization abbreviation 115
`source` 9
source files. *see* files, source
specifying an executable 84
spontaneous call, in Dynamic Call Graph 111
starting to profile 92
Status option 25
`stop` 97
stopping profiling 97
  from the command line 97
  from the X interface 97
syntax of commands xvi

T

`t` profiling status 109
tables
  Basic Block Performance Analysis 124
  Dynamic Call Graph 110
  Loop Performance Analysis 112
  Parallel Region Performance Analysis 117
  Routine Performance Analysis 107
TAC xviii
technical assistance xviii
Technical Assistance Center (TAC) xviii
terminal types and CXpa 10
thread, defined 117
typographic conventions xvii

U

`u` profiling status 109
`UL` optimization abbreviation 115
uninstrumented loop 109
`Ur` optimization abbreviation 115
using this book xv

V

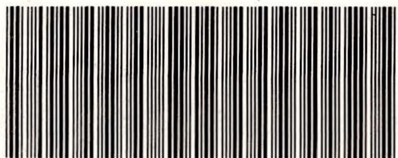
`V` optimization abbreviation 115
vector chaining 133
vector Mflops. *see* Mflops
Vectorized loop table 132
version of CXpa 26
VT100 terminals 9

W

windows
  Bar Graph 126
  Command 12
  CXpa 13
  Help 28

X

X interface
  described 12
  key bindings in Command Window 12
X Window System 12
xterm for process I/O 92



Order Number
DSW-251

Document Number
710-007230-005