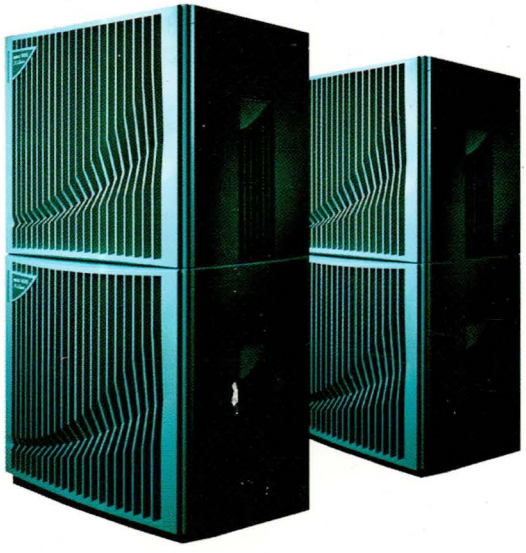
HEWLETT® PACKARD

Exemplar
S-Class and
X-Class Servers



CXpa Reference

Third Edition

Hewlett-Packard Company
Convex Division
3000 Waterview Parkway
P.O. Box 833851
Richardson, TX 75083-3851
United States of America

CXpa Reference

Exemplar S-Class and X-Class Servers

B5639-90002

Third Edition

January 1997

Hewlett-Packard Company
Convex Division
Richardson, Texas
United States of America

CXpa Reference

Exemplar S-Class and X-Class Servers

B5639-90002

Notice

The information contained in this document is subject to change without notice.

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

This entire book is recyclable.

Printed in the United States of America

Revision information for

CXpa Reference

Exemplar S-Class and X-Class Servers

Edition	Document No.	Description
Third	B5639-90002	Released with CXpa V4.0, January, 1997.
Second	710-004730-009	Released with V3.5 of CXpa software on Exemplar SPP Series systems.
First	710-004730-008	Initial release, June 1995. Released with V3.3 of CXpa software. For Exemplar SPP Series systems, this book replaces the *CXpa Reference*, Second Edition (DSW-253).

Contents

Using this book xi
Purpose and audience xi
Supported systems xi
Organization xii
Notational conventions xiii
Command syntax xiii
General conventions xiii
Notes xiv
System-specific information xiv
Associated documents xiv
Ordering documentation xvi
Technical assistance xvi

Part 1 Using CXpa

1 Introduction 3
CXpa overview 5
What is CXpa? 5
Why use CXpa? 6
How to use CXpa 6
Available metrics 7
Off-processor events and latency metrics 8
On-processor events and latency metrics 8
CXpa interfaces 8
GUI mode 9
Line mode 9
Batch mode 9
Graphical analysis of performance data 9
Call Graph window 10
Performance reports 11
CXpa limitations 13

2 Preparing programs for profiling. 15
Compiling for CXpa 17
Compiling and linking in one step 19
Compiling and linking separately 19
Profiling routines that call uninstrumented routines . 21
Using `cxoi` to instrument object files and archive libraries 23

Makefile example 25
Limitations 25
Known problems 26

3 Profiling with CXpa 27
Learning CXpa quickly 29
Basic profiling using CXpa's GUI 29
Basic profiling using CXpa's line mode interface . . 35
Editing the command line 38
Profiling strategy 41
Profiling intrusion—what is it, and what causes it? . 41
Profiling intrusion—an example 42
Recommended profiling strategy 42
Performance data files 49
Changing the PDF name 49
In GUI mode 49
In line mode or batch mode 50
Analyzing PDFs only 51
Analyzing PDFs only in GUI mode 51
Opening other PDFs for analysis 52
Choosing an active PDF 52
Analyzing PDFs only in line mode 53
Opening other PDFs for analysis 54
Using pre-instrumented executables 55
Using the PROFDIR environment variable 56
Using pre-instrumented executables in line mode . . 56
Using pre-instrumented executables in GUI mode . 58
Profiling MPI and PVM applications with CXpa . . . 61
Generating PDF files 61
Analyzing profiling data from multiple PDF files . . 63
CXpa and the mpa utility 67
Using batch mode 69
Using batch mode from the command line 69
Command file input using the `-x` option . . . 69
Argument input using the `-e` option 70
Using batch mode from a script 71
Using online help 73
Using the buttons and scroll bars 73
Using the menus 75
Selecting a topic 76
Searching for a topic 76
Printing a help topic 77
Exiting the help system 77

4 Choosing performance data to collect . . 79
Introducing source code regions 81
Selecting source code regions 81
Guidelines for selecting regions to profile 82
Source code annotations for regions 83
Selecting regions in GUI mode 85
Selecting source code regions to profile 86
Selecting loop nesting levels 89
Selecting and deselecting regions in line mode 93
Selecting or deselecting one type of region in all routines 94
Selecting or deselecting one type of region in specific routines 95
Selecting or deselecting a type of region at specific lines 96
Selecting or deselecting all regions in specific routines 97
Selecting or deselecting all regions at specific lines . 98
Selecting or deselecting all regions in all routines . . 99
Introducing metrics 101
Metrics available on all architectures 101
Event metrics 102
Terms and definitions 103
Off-processor events (Exemplar S2000, Exemplar X2000, and SPP1600 Series systems) . . . 109
On-processor events (SPP1200 and SPP1600 Series systems) 110
Selecting metrics in GUI mode 113
Selecting metrics in line mode 117
Choosing metrics to collect at the command line . . 117
Event types 118
Off-processor events (Exemplar S2000, Exemplar X2000, and SPP1600 Series systems) . . 119
On-processor events (SPP1200 and SPP1600 Series systems) 120

Part 2 Reference pages

5 Reports 123
Report types 123
Displaying report data for individual threads 124
Line mode 124
GUI mode 124
Viewing reports 125
GUI mode 125
Line mode 125

Report header 126
Dynamic Call Graph report 127
Loop reports 131
Computation 133
Time to solution 133
Events 134
Cache misses and latency 135
Data cache accesses
(SPP1200 and SPP1600 Series only) 137
Instructions completed and clock cycles
(SPP1200 and SPP1600 Series only) 138
Data and instruction TLB misses
(SPP1200 and SPP1600 Series only) 139
Parallel Region reports 143
Time to solution 144
Events 146
Cache misses and latency 147
Data cache accesses
(SPP1200 and SPP1600 Series only) 149
Instructions completed and clock cycles
(SPP1200 and SPP1600 Series only) 150
Data and instruction TLB misses
(SPP1200 and SPP1600 Series only) 151
Report fields 155
Routine reports 161
Call counts 162
Computation 162
Time to solution 163
Events 164
Cache misses and latency 165
Data cache accesses
(SPP1200 and SPP1600 Series only) 167
Instructions completed and clock cycles
(SPP1200 and SPP1600 Series only) 167
Data and instruction TLB misses
(SPP1200 and SPP1600 Series only) 169

6 Windows 173
2D Profile window 175
3D Profile window 181
Analysis Control window 187
Selecting a different PDF file to analyze 188
Opening additional PDF files for analysis 188
Invoking CXpa with multiple PDF files for analysis . 189
Merging PDF files 189
Analysis Report window 193
Call Graph window 197

Changing the number of routines to display . . . 199
Selecting a different metric to rank routines 199
Viewing detailed information and source code for routines 199
Searching for a routine 199
Displaying information for specific threads 200
Executable Manager window 203
Profiling a program 204
Filter Profile dialog 209
Filter Report dialog 211
Find Routine dialog 213
Help window 217
Info Executable dialog 221
Info PDF dialog 223
Info Session dialog 225
Merge PDFs dialog 227
New PDF dialog 231
Specifying a new PDF 232
Open PDF dialog 235
Selecting a file 236
Product Information dialog 239
Profile Selection dialog 241
Selecting source code regions to profile 242
Selecting loop nesting levels 246
Selecting metrics to collect 249
Region Subset Selection dialog 255
Routine Detail dialog 259
Save Executable As dialog 263
Save Profile dialog 267
Save Report dialog 271
Sort dialog 273
Source Code Selection dialog 277
Source Code window 279
Annotations 280
Changing source files 280
Source Search Path dialog 283
Adding a directory to CXpa's search path 283
Removing a directory from CXpa's search path . . 284
Subcomplex Selection dialog 285
Thread Selection dialog 287
X defaults 289
Generic application resources 289
2D and 3D graph resources 290
Session behavior resources 290
Zoom dialog 293
2D graph zoom options 294
3D graph zoom options 294

7 Commands 297

add path 299
analyze 301
collect 307
continue 309
cxpa 311
Preparing programs for profiling with CXpa . . . 314
Starting CXpa in GUI mode 314
Starting CXpa in line mode 315
Using the CXPA environment variable 316
deselect 319
help 321
info 325
Executable information 325
PDF information 325
File and directory information 326
list 327
list selectable 331
merge 335
path 339
quit 341
rerun 343
run 345
save executable 349
select 353
set events 357
set pdf 361
set subcomplex 363
set visibility 365
Loop nesting level filters 366
Process/thread filters 367
source 369
stop 371
version 373

8 Messages 375

Glossary 395

Index 411

Using this book

Purpose and audience

The *CXpa Reference* is both a user's guide and a reference. This book introduces new users to CXpa, an interactive performance analysis tool, and describes the metrics, graphs, reports, commands, and interfaces that are available in the CXpa software. This book is also available through the CXpa online help system.

This book is written for Hewlett-Packard technical marketing and system engineers, independent software vendors (ISVs), and customers who develop and tune C, Fortran 77, and C++ applications on Exemplar S-Class and X-Class technical servers.

Supported systems

This book provides information about using CXpa on the following supported hardware platforms:

- Exemplar S-Class technical servers (also called Exemplar S2000 systems)
- Exemplar X-Class technical servers (also called Exemplar X2000 systems)
- SPP1200 Series and SPP1600 Series systems

Organization

Use the following table to select which portions of the book you wish to read.

To learn about...	Read...
CXpa and profiling in general	Chapter 1, "Introduction"
Compiling applications for profiling with CXpa and using the `cxoi` utility to instrument object files and libraries for profiling.	Chapter 2, "Preparing programs for profiling"
Profiling strategy, learning CXpa, performance data files (PDFs), profiling MPI and PVM applications, using pre-instrumented executables, running CXpa in batch mode, and using online help.	Chapter 3, "Profiling with CXpa"
Metrics you can collect with CXpa and how to select regions and metrics for profiling using the GUI or line mode interface.	Chapter 4, "Choosing performance data to collect"
CXpa's text reports.	Chapter 5, "Reports"
Using CXpa's windows and dialogs.	Chapter 6, "Windows"
Using commands, including complete syntax and examples.	Chapter 7, "Commands"
CXpa error and informational messages.	Chapter 8, "Messages"
Terms and definitions specific to CXpa metrics and profiling, Exemplar systems, Exemplar compilers, and parallel programming on Exemplar systems.	"Glossary"

Notational conventions

This document uses the following notational conventions.

Command syntax

The syntax for CXpa commands uses the following notational conventions:

```
(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 by the user.

④ The horizontal ellipsis in brackets [, ...] indicates that more than one of the preceding parameter can be specified.

⑤ Either **`a`** or **`b`** must be specified.

⑥ [*<param2>*] indicates an optional parameter. All items in brackets are optional.

General conventions

- **`Bold constant-width font`** identifies user input in examples.
- *Italics*:
  - Designate user-supplied variables in a command-line example when enclosed in <>
  - Indicate document titles
  - Introduce new terms
- `Constant-width font` designates input and output, including:
  - Command names and options
  - System calls
  - Program statements, command output, and error messages returned
- Horizontal ellipsis (...) shows repetition of the preceding item.
- Vertical ellipsis shows that lines 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 press simultaneously. For example, **CTRL-X** indicates that you must press and hold down the **CTRL** key and then press the **X** key.
- The shell prompt is shown as a percent sign (%).
- Unless otherwise indicated, source code examples are in Fortran.

Notes

NOTE: A NOTE highlights information that may be of particular interest regarding the software or your files.

System-specific information

The architecture of the computer system can affect some aspects of CXpa and the program you are profiling, such as number and type of available metrics.

These architecture dependencies are indicated throughout this book by including the name of the system to which the information applies in the title of a section or by a notation in parentheses.

Associated documents

For more information, refer to the following books:

- *Exemplar Programming Guide: S-Class and X-Class Servers* (Order No. B5600-90001)— Describes efficient shared-memory programming techniques for Exemplar S2000, Exemplar X2000, and SPP1600 Series systems. Topics covered include the Exemplar programming model, basic and advanced shared memory programming, compiler optimizations, optimization reports, and Compiler Parallel Support Library (CPSlib) routines.

- *Exemplar C and Fortran 77 Programmer's Guide: S-Class and X-Class Servers* (Order No B5600-90002)—Describes the Exemplar compilers, which are based on Hewlett-Packard's `f77` and `c89` compilers and have been extended to support the Exemplar programming model. Topics include extensions to the HP `f77` and `c89` compilers, transition information for users migrating from SPP Series `fc` and `cc` compilers, the Exemplar linker, and the `mpa` (modify process attributes) utility.
- *Exemplar Architecture: S-Class and X-Class Servers* (Order No. A4716-9001)—Describes the implementation of scalable parallel processing on Exemplar S-Class and X-Class technical servers.
- *Exemplar SPP 1000-Series Architecture* —Describes the implementation of scalable parallel processing on Exemplar SPP1000, SPP1200, and SPP1600 Series systems.
- *HP MPI User's Guide: S-, X-, D-, and K-Class Servers* (Order No. B6011-90001)—Describes the HP Message Passing Interface (HP MPI) product. MPI is a library of C- and Fortran 77-callable routines for message-passing programming. Topics covered include building and running HP MPI applications, analyzing HP MPI applications, and tuning HP MPI applications to improve performance.
- *HP PVM User's Guide: S-Class and X-Class Servers*—Describes the HP Parallel Virtual Machine (HP PVM) product. PVM is a library of C-and Fortran 77-callable routines for message-passing programming. Topics covered include building and running PVM applications, analyzing HP PVM applications, and tuning HP PVM applications to improve performance.
- *Exemplar C++ Programming Guide* (B5639-90001)—Describes how to use the C++ compiler for Exemplar systems.
- *PA-RISC 1.1 Architecture and Instruction Set Reference Manual,* available from Hewlett-Packard (HP Manual Part Number 09740-90039).
- *PA-RISC 2.0 Architecture and Instruction Set Reference Manual,* available from Hewlett-Packard.

Ordering documentation

To order the current edition of this or any other Hewlett-Packard Convex Division document, send requests to:

Hewlett-Packard Company
Convex Division
Customer Service
P.O. Box 833851
Richardson TX 75083-3851 USA

Include the order number (*xxxxx-9xxxxx*) or the exact title of the document.

Technical assistance

If you have questions that are not answered in this book, contact the Hewlett-Packard Convex Technical Assistance Center (TAC) at the following locations:

- Within the continental U.S., call 1 (800) 952-0379.
- From Canada, call 1 (800) 345-2384.
- All other locations, contact your local Hewlett-Packard 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 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.

Part 1 Using CXpa

Part 1 of this book introduces new users to CXpa and contains the following chapters and subjects:

1. Introduction
   - CXpa overview
   - CXpa limitations
2. Preparing programs for profiling with CXpa
   - Compiling for CXpa
   - Using `cxoi` to instrument object files and archive libraries
3. Profiling with CXpa
   - Learning CXpa quickly
   - Profiling strategy
   - Performance data files (PDFs)
   - Analyzing PDFs only
   - Using pre-instrumented executables
   - Profiling message-passing applications with CXpa
   - CXpa and the `mpa` utility
   - Using batch mode
   - Using online help
4. Choosing performance data to collect
   - Source code regions
   - Selecting regions in GUI mode
   - Selecting and deselecting regions in line mode
   - Metrics
   - Selecting metrics in GUI mode
   - Selecting metrics in line mode

You can also access this information through the CXpa online help system.

Introduction 1

This chapter introduces general profiling concepts and the features and capabilities of the interactive performance analysis tool, CXpa. If you are new to profiling or have not used CXpa, read this chapter before using CXpa.

The topics discussed in this chapter include:

- CXpa overview
- CXpa limitations

CXpa overview

This section provides an overview of CXpa features and covers the following topics:

- What is CXpa?
- Why use CXpa?
- How to use CXpa
- Available metrics
- CXpa interfaces
- Graphical analysis of performance data
- Call graph window
- Performance reports

What is CXpa?

CXpa is an interactive runtime performance analysis tool for programs compiled with Exemplar Fortran, C, and C++ compilers or other HP PA-RISC targeting compilers.

Most performance analysis tools, including CXpa, require you to compile your program with a special profiling option. This option tells the compiler to create a special executable file that contains information that the profiler uses to collect performance data. When you select source code regions to profile and run this executable file, the profiler collects performance data and reports the results. These reports and graphs enable you to determine the location of performance bottlenecks.

Different profilers collect performance data in different ways. Some profilers sample a program's performance at measured intervals to get an average of a routine's execution time. This is known as statistical sampling. The `gprof` and `prof` utilities use statistical sampling.

CXpa measures a program's entire execution time and reports the total time spent in individual routines, loops, and parallel regions (parallel loops). This produces profiling results that are more complete than statistical sampling.

Using CXpa, you can profile selected parts of your program, control your program's execution, and view performance information in reports or graphs.

CXpa supports:

- Profiling of routines, loops, and parallel loops for applications compiled with Exemplar Fortran 77, C, and C++ compilers.
- Routine-level profiling of object files and archive libraries created with PA-RISC targeting compilers.
- Display of profiling information for individual threads of execution or across the entire process.
- Pre-instrumented executables. This allows you to write profile selection settings (instrumentation) to the current executable file or to a copy of the current executable. The resulting executable file can be run outside the control of CXpa and profiling data collected in a performance data file (PDF) for later analysis.
- Graphical analysis of profiling results.
- Profiling of MPI and PVM message-passing applications.

Why use CXpa?

Using the information CXpa provides, you can discover which routines and loops slow down your program the most. In some cases, simple modifications to the source code (for example, inserting compiler directives) can result in significant performance improvements. Without profiling, you might not be able to find performance bottlenecks.

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.

Profiling versions of your program that have been compiled at different optimization levels can provide insight into the types of optimizations that work best for a given situation.

How to use CXpa

Profiling a program is an iterative process. You profile your program, make changes to the source code based on the results, and profile again. This profiling experience, along with an understanding of compiler optimizations, is crucial to improving your program's performance.

The following steps suggest ways to gain practical experience using CXpa:

1. Select a program you have written (or are very familiar with) that takes several minutes to execute and contains loops.
2. Read Chapter 2, "Preparing for profiling," and follow the procedures described there to instrument your program for profiling with CXpa.

3. Read "Learning CXpa quickly," which appears in Chapter 3. As you read, perform the profiling steps on your program. Use the default options to profile routines only until you reach Step 5 below.

   The first time you compile your program for CXpa, compile it without optimizations (at optimization level +O0).

4. Recompile your program at a higher optimization level and profile it again. Continue doing this until you have found the optimization level that produces the lowest wall clock time for your program.
5. Identify the routine that takes the longest to execute.
6. Profile the loops in that routine and identify the loop that took the longest to execute.
7. Rewrite the loop (or routine) in another way. Refer to the *Exemplar Programming Guide* for more information about optimizing your code.
8. Recompile and profile the program again. Compare the execution time to the previous execution time.

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

Refer to the "Profiling strategy" section of this book for a step-by-step strategy for profiling with CXpa.

Available metrics

By default, CXpa measures CPU time, wall clock time, and execution counts for selected source code regions of your program. The following additional metrics are provided on all architectures:

- Memory access event counts
- Latency time for memory access events (latency is the time it takes to satisfy a memory access request)
- Dynamic call graph

Available memory access event counts and latency metrics differ according to machine type, as described in the following sections.

Off-processor events and latency metrics

On Exemplar S2000, Exemplar X2000, and SPP1600 Series systems, performance counters located on the processor agent chip, called *off-processor event counters*, can be configured to collect cache miss event counts and latency metrics for:

- Local cache misses—number of times that data not found in the processor cache was found in local memory (memory allocated to that processor's hypernode).
- Remote cache misses—on multinode Exemplar X2000 systems and SPP1600 Series systems, the number of times that data not found in the processor cache was found in remote memory (memory allocated to another hypernode).
- Local and remote cache misses combined.
- Event latency time, event latency/CPU time, and event latency/counts metrics for the above events.

On-processor events and latency metrics

On SPP1200 and SPP1600 Series systems, event counters on the HP PA-RISC 7200 Series processors, called *on-processor event counters*, can be configured to collect the following events:

- Data cache miss counts and latency, data cache access counts, average data cache miss latency, and data cache hit rates.
- Instruction cache miss counts and latency, and average instruction cache miss latency.
- Completed instruction counts, CPU clock cycles, average number of CPU cycles per instruction, and the MIPS rate.
- Data and instruction TLB (translation lookaside buffer) misses.

Refer to the "Introducing metrics" online help topic or section of this book for more information about metrics available on each architecture.

CXpa interfaces

CXpa has three interfaces: an X/Motif graphical user interface (GUI), a character-oriented tty interface (line mode), and a batch interface (batch mode).

You can run your application in line mode or batch mode to collect profiling data, then use CXpa's GUI to view a graphical analysis of the data.

GUI mode

CXpa's X/Motif GUI provides a mouse-oriented interface and graphical analysis of performance data. The GUI has the following features:

- Mouse-driven selection of source code regions (routines and loops) to profile and metrics to collect.
- 2D and 3D graphs of performance data that can display corresponding source code with the click of a mouse button.
- Dynamic call graph window with point-and-click navigation, clickbacks to source code, caller/callee information, and metrics for selected routines.
- Source Code window with source code region annotations.
- Analysis Report window for displaying textual performance reports.
- Analysis-only mode for visualizing multiple performance data files (PDFs) simultaneously, including PDFs that were created on different architectures.

Line mode

Line mode is a character-based, command-line interface to CXpa. Use line mode when you are using a CRT terminal or if you prefer a line-oriented interface in an xterm window. Line mode presents performance information in textual reports.

To use CXpa in line mode, specify the `-nw` option when invoking CXpa from the shell command line. When you start CXpa in line mode with the name of a PDF file only, you can use the `set pdf` and `analyze` commands to access performance data for multiple performance data files (PDFs), including PDFs that were created on different architectures.

Batch mode

CXpa's batch mode enables you to run CXpa from a shell script. You can invoke CXpa in batch mode by:

- Using the `-x` option to supply a command file to CXpa.
- Redirecting input, output, and standard error to and from files.

Graphical analysis of performance data

The 2D and 3D Profile windows allow you to visualize the performance data collected when you run your program. You can open multiple 2D and 3D Profile windows to compare and contrast different aspects of your program's performance simultaneously.

You can obtain different views of your program's performance by selectively defining the source code regions and metrics that are graphed. The 2D Profile graphs the data per process, while the 3D Profile graphs the data per thread (for parallel codes).

Other features provided with the 2D and 3D profile graphs include:

- **Source code correlation**—Click on any bar in the graph to display the source code associated with the code region being graphed.
- **Sorting and zooming options**—The data for both graphs can be sorted alphabetically, by load order, from lowest to highest, or from highest to lowest.

  Zooming (scaling) options for the 2D and 3D profile graphs allow you to view or specify the range or number of data values displayed at one time in the 2D or 3D Profile window. This can be useful when there is a large number of data items to graph and you want to focus on a subset of the data.
- **Saving profiles for printing or export**—You can save profile graphs in PostScript or xwd formats for printing or to ASCII format for export to other graphics packages.
- **3D profile graph rotation**—The 3D profile graph can be rotated by clicking anywhere in the graph with the right mouse button and moving the mouse.

2D and 3D profile graphs are available in GUI mode only. Refer to the "2D Profile window" and "3D Profile window" online help topics or sections in this book for more information.

Call Graph window

CXpa supports collection of dynamic call graph information, which can be displayed in an interactive Call Graph window that features:

- Point-and-click navigation.
- Clickbacks to source code for routines displayed in the graph.
- Display of values for all metrics collected for each routine.
- Listing of callers and callees and their call counts for each routine.
- Options for configuring the number of routines initially displayed in the graph and selecting the metric used to rank the routines.

Performance reports

CXpa displays textual performance reports for:

- Routines.
- All loops (including compiler-generated parallel loops) for modules compiled with Exemplar compilers at optimization levels `+O2` and `+O3`.
- Parallel loops only (that is, parallel loops generated by Exemplar compilers at optimization level `+O3 +Oparallel`).

The metrics available in performance reports vary according to machine type, source code regions selected, and the options used when compiling your program. Textual performance reports are available in all CXpa interfaces (GUI mode, line mode, and batch mode).

Performance analysis reports that can be displayed are as follows:

- **Counts**—Routine call counts.
- **Dynamic Call Graph**—For each routine, inclusive and exclusive wall clock time; inclusive and exclusive CPU time; and call counts for the routine, its parents, and its children are displayed. Routines are sorted from highest to lowest by inclusive wall clock time.
- **Computation**—CPU time and % total CPU time metrics.
- **Time to Solution**—Wall clock time and CPU/wall clock time metrics.
- **Events**—Available event reports and metrics differ according to machine architecture:
  - Off-processor event reports (Exemplar S2000, Exemplar X2000, and SPP1600 Series only)—Locally resolved cache misses, remotely resolved cache misses, or locally and remotely resolved cache misses.
  - On-processor event reports (SPP1200 and SPP1600 Series only)—Data cache misses and latency; data cache accesses; instruction cache misses and latency; instructions completed and clock cycles; and data and instruction TLB (translation lookaside buffer) misses.

For a detailed discussion of each type of report, refer to the "Reports," "Routine reports," "Dynamic Call Graph report," "Loop reports," and "Parallel Region reports" online help topics or sections in this book.

Related Concepts

Analyzing PDFs only
Compiling
Introducing metrics
Learning CXpa quickly
Performance data files
Using cxoi to instrument object files and libraries
Profiling MPI and PVM applications with CXpa
Profiling strategy
Reports
Using pre-instrumented executables

Related Windows

2D Profile window
3D Profile window
Analysis Control window
Analysis Report window
Call Graph window
Executable Manager window

CXpa limitations

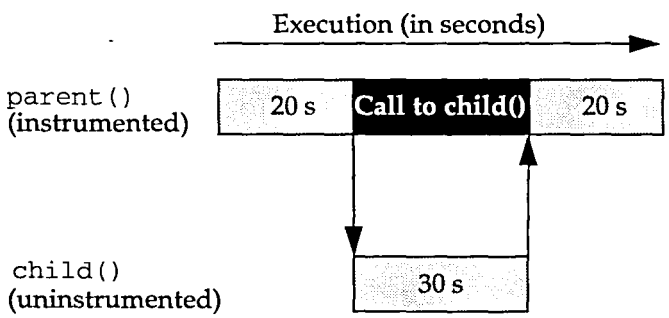
This section lists CXpa's limitations. CXpa can still be used effectively in situations where these limitations occur.

To view current information on CXpa limitations in GUI mode, select Release Notice from the Help menu in the Executable Manager or Analysis Control window. In line mode, enter the command `help release` to view an ASCII text version of the CXpa release notice for the version of CXpa installed on your system.

The limitations include:

- If a process forks, but does not perform an `exec()`, CXpa will only profile the parent process.
- If an instrumented routine calls an uninstrumented routine, CXpa cannot separate the time spent or metrics collected in the uninstrumented child routine from the time spent in the instrumented parent.

  This condition is illustrated in the following figure:



In this figure, routine `parent()` has been instrumented for CXpa while routine `child()` has not. The time spent in `parent()` not including children is reported as 70 seconds because CXpa cannot separate time spent in `child()`. If routine `child()` had been instrumented for profiling with CXpa, CXpa would have correctly reported `parent()` as having executed for 40 seconds not including time spent in `child()`.

This condition can produce misleading results in the dynamic call graph. If the example above is extended to show that routine `child()` calls an instrumented routine `sub1()`, the dynamic call graph would incorrectly show that routine `parent()` called `sub1()`, and all of the time spent in the uninstrumented routine `child()` would be attributed to `sub1()`.

For this reason, make sure that all routines in the program are instrumented for profiling when generating a dynamic call graph.

- Object files and archive libraries instrumented for profiling with `cxoi` do not contain source file line number information. This means that source code correlation and clickbacks for routines within these modules always refers to line 1 of the source file that contains the routine.

  CXpa source code annotations are not displayed in the Source Code window or in source file listings for object files and libraries instrumented with `cxoi`.

- The following constructs cannot be profiled:
  - Loops with no exit point that the compiler can detect. For example:

```
foo() { exit(1); }
.
.
.
while (1) { if (...) foo(); }
```

  - Loops whose exits are implemented by the compiler's code generator via branch tables. For example, the following report fragment indicates that CXpa could not profile a loop region:

```
================================================================================
                          Loop Performance Analysis
================================================================================

Optimized Loops:
                        Times      Iteration Count          CPU Time
 Line    NL Optimization Exec     Min      Max    Avg (less inner)(plus inner)PS
------- -- ------------ -------- ------ ------ ------ ---------- ---------- --
   804   0               N/A    N/A    N/A    N/A        N/A        N/A    u

(u) contains 1 or more uninstrumentable points
```

Preparing programs for profiling

2

This chapter explains how to prepare programs for profiling with CXpa and covers the following topics:

- Compiling for CXpa
- Using `cxoi` to instrument HP PA-RISC object files and archive libraries. `cxoi` is a separate utility shipped with CXpa.

Compiling for CXpa

This section describes how to compile programs for CXpa using the Exemplar C, Fortran, and C++ compilers.

You can also use the `cxoi` utility to prepare object files and archive libraries for routine-level profiling with CXpa. These object files and archive libraries can be created with any HP PA-RISC targeting compiler. Refer to the "Using cxoi to instrument object files and libraries" online help topic or section of this book for instructions.

Syntax

<compiler> **+pa** [*optimization_options*] *<files>*

Parameter	Meaning
compiler	Specifies one of the Exemplar compilers. The Exemplar compilers are based on Hewlett-Packard compilers, but have been extended to support the Exemplar programming model: `/opt/ansic/bin/c89`, `/opt/ansic/bin/cc`—The Exemplar C compiler. `/opt/fortran/bin/f77`—The Exemplar Fortran compiler. `/opt/CC/bin/CC`—The Exemplar C++ compiler.
+pa	Compiles the application for CXpa.
optimization_options	Specifies an optimization level. The types of source code regions that can be profiled vary, depending on the optimization level: `+O0`, `+O1`—Routines. `+O2`, `+O3`—Routines and loops. `+O3 +Oparallel`—Routines, loops, and compiler-generated parallel loops. `+O4` and `+Oall`—The `+O4` and `+Oall` optimization options are not supported for use with `+pa`.

Refer to the c89(1), f77(1), CC(1) compiler man pages on Exemplar systems for more information on the Exemplar compilers.

files — Specifies the name of one or more source files, object files, or libraries.

Description

To compile an application for profiling with CXpa, specify the `+pa` compiler option when compiling and linking. This tells the compiler to add instructions to the executable file that enable CXpa to gather performance data during execution of the program. You must specify the `+pa` option when linking to ensure that the timing and data collection routines (cxpamon.o) are linked into the executable. For example:

```
% /opt/fortran/bin/f77 +pa myprog.f sub1.f sub2.f
```

In the above example, the source files myprogram.f, sub1.f, and sub2.f are compiled and linked into an executable file with the Exemplar Fortran 77 compiler. The `+pa` option tells the compiler to instrument the executable for profiling with CXpa and links in the runtime monitoring routines.

When you compile your program with the `+pa` option, the types of source code regions that can be selected for profiling depend on the optimization level specified:

- `+O0` and `+O1`—Only routines can be selected for profiling. `+O0` is the default optimization level for the Exemplar compilers.
- `+O2` and `+O3`—Routines and loops can be selected for profiling.
- `+O3 +Oparallel`—Parallel loops created by the compiler can also be selected for profiling.

The following compiler options are incompatible with `+pa`:

- `-p` and `-G`. These options are designed for use with other profilers.
- `+O4`
- `+Oall`
- `-s`. This option causes the output of the linker to be stripped of symbol table information). The `strip(1)` command also removes this information.

With CXpa, you can select specific routines and loops to profile. For example, you can profile some or all of the source code regions 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 compiled for use with CXpa can still be executed outside of CXpa in the usual way. 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:

```
% /opt/ansic/bin/c89 +pa +O3 +Onoinline main.c
```

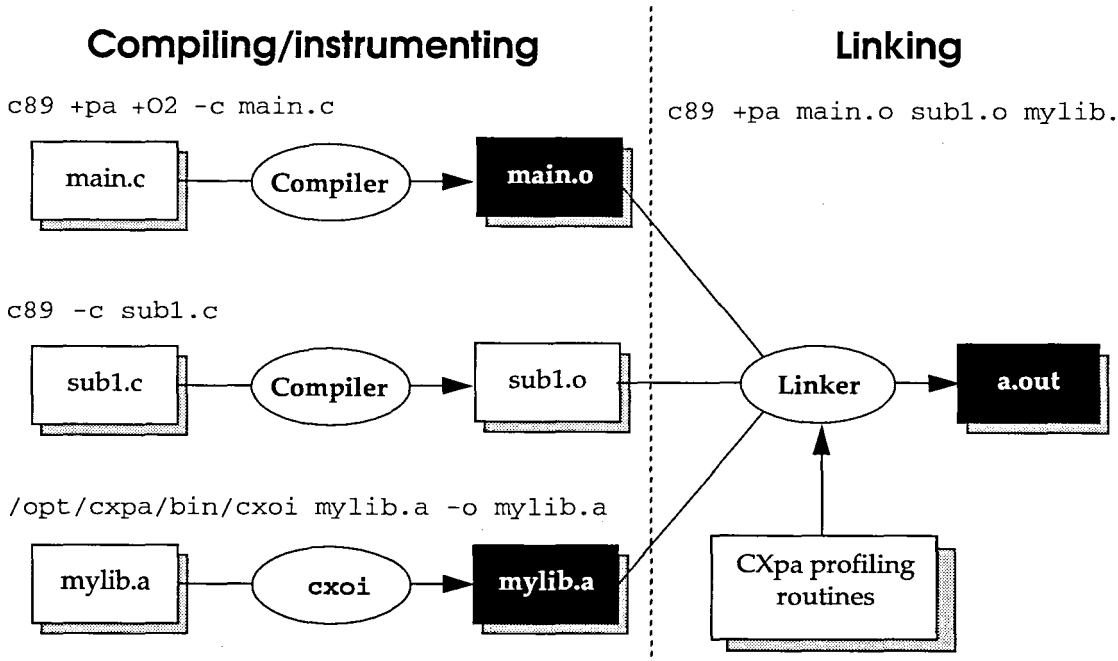
In the above example:

- The source file main.c is compiled at optimization level `+O3` with the `+pa` compiler option to produce the executable `a.out`. Routines and loops are instrumented for profiling with CXpa.
- The `+Onoinline` option suppresses inlining. At optimization level `+O3`, the Exemplar compilers can inline routines that are called within the same source file, and compilation time can increase substantially.

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 linked together into an executable.

When compiling for CXpa, you can compile each source file with the same or different options. You must, however, use the `+pa` option when linking, as shown in the following figure.



In the above figure, the program being compiled has two source files and an archive library:

- The source file main.c is compiled into an object file at optimization level +O2. The +pa option instruments the file for profiling with CXpa.
- The source file sub1.c is compiled into an object file (the -c option suppresses linking) without adding any instrumentation for CXpa.
- The archive library mylib.a is instrumented for profiling with cxoi, an object and archive library file instrumentor shipped with CXpa. The -o option specifies the name of the instrumented file.

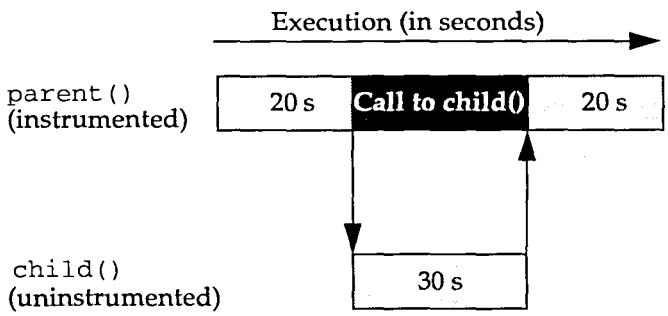
The second call to the compiler

```
% c89 +pa main.o sub1.o mylib.a
```

invokes the linker, which combines the instrumented object files and archive library into an executable. The linker also links CXpa's timing routines (cxpamon.o) into the executable. You cannot profile using CXpa unless these routines are linked into your executable.

Profiling routines that call uninstrumented routines

If an instrumented routine calls an uninstrumented routine, CXpa cannot separate the time spent in the uninstrumented child routine from the time spent in the instrumented parent. This condition is illustrated in the following figure:



In the above example, routine `parent()` has been instrumented for CXpa while routine `child()` has not. The time spent in `parent()` not including children is reported as 70 seconds because CXpa cannot separate time spent in `child()`. If routine `child()` had been instrumented for profiling with CXpa, CXpa would have correctly reported `parent()` as having executed for 40 seconds not including children.

Related Topics

Using cxoi to instrument object files and libraries

Related Commands

`cxpa`

Compiling for CXpa

Using cxoi to instrument object files and archive libraries

Description

Use the object file instrumentor—`/opt/cxpa/bin/cxoi`, a separate utility shipped with CXpa—to instrument object and archive library files produced by any PA-RISC targeting compiler. Only routine-level profiling is enabled with this utility.

The `cxoi` utility only *enables* selection of routine regions for profiling in object and archive library files; it does not automatically select these regions for profiling. Source code region selection is done using CXpa.

Refer to the "Learning CXpa quickly" online help topic or section of this book for more information on selecting regions and metrics for profiling.

Syntax

`cxoi {`*lib.a* | *file.o*`} [-o` *output_file*`] [-t`*x*`,`*name*`]`

Parameter	Meaning
lib.a	Archive library file.
file.o	Object file. You can specify only one per invocation of `cxoi`.
`-o` *output_file*	Write the instrumented *file.o* or *lib.a* to *output_file*. If you do not specify the `-o` option, `cxoi` names the instrumented file *file*`.cxoi.o` or *lib*`.cxoi.a`.
`-t`*x*`,`*name*	Substitute subprocess *x* with *name* where *x* is one or more of a set of identifiers indicating the subprocesses. This option works in two modes: If *x* is a single identifier, *name* represents the full path name of the new subprocess. If *x* is a set of identifiers, *name* represents a prefix to which the standard suffixes are concatenated to construct the full path names of the new subprocesses. The *x* can take one or more of the following values: `a` Assembler (standard suffix is `as`). `l` Linker (standard suffix is `ld`).

Examples

To prepare HP PA-RISC object files and archive libraries for routine-level profiling with CXpa:

1. Use the `cxoi` utility to insert instrumentation slots for collecting routine-level performance information into the object file or archive library.

   – To instrument all routine entry points in an object file for profiling with CXpa, use a command line similar to the following:

   ```
   % /opt/cxpa/bin/cxoi file.o
   ```

   – To instrument all routine entry points in an archive library for profiling with CXpa, use a command line similar to the following:

   ```
   % /opt/cxpa/bin/cxoi libx.a
   ```

   By default, `cxoi` names the instrumented object or library file *file*`.cxoi.o` or *lib*`.cxoi.a`. To specify a different name for the instrumented file, use the `-o` option. For example, the command line

   ```
   % /opt/cxpa/bin/cxoi libc.a -o mylibc.a
   ```

   creates a new archive library file, `mylibc.a`, that contains CXpa instrumentation slots for routine entry points. The original file, `libc.a`, is not modified.

   To modify the original object or library file in place, use the `-o` option and specify the original file name. For example, the command line

   ```
   % /opt/cxpa/bin/cxoi libc.a -o libc.a
   ```

   overwrites the original library file with a version that is instrumented for profiling with CXpa, if you have write access to the file and directory.

   You cannot specify multiple object files or libraries on the `cxoi` command line (that is, command lines such as `cxoi *.o` or `cxoi obja.o objb.o` will not work).

2. Link the instrumented object and archive library files into an executable file using the `+pa` option of the Exemplar compilers. Use a command line similar to the following:

   ```
   % /opt/fortran/bin/f77 +pa file.cxoi.o
   ```

   or

   ```
   % /opt/fortran/bin/f77 +pa file.o libx.cxoi.a
   ```

You can now start CXpa with the name of the new executable and select source code regions and metrics for profiling.

If `cxoi` encounters an object file that is already instrumented for CXpa, it ignores the file, emits a warning message, and exits. If you are instrumenting an archive library and `cxoi` encounters an object file in that library that is already instrumented, it ignores that object file and continues processesing the other object files in the archive.

Makefile example

The following example Makefile for a C program illustrates how to write standard Makefile rules to compile, instrument, and link sources into an executable file for profiling with CXpa.

```
C89     = /opt/ansic/bin/c89
CXOI    = /opt/cxpa/bin/cxoi

OBJECTS       = main.o      bar.o
CXOI_OBJECTS = main.cxoi.o bar.cxoi.o
LIBS          = mylib.a
CXOI_LIBS     = mylib.cxoi.a

%.o: %.c
        $(C89) -c $*.c

%.cxoi.o: %.o
        $(CXOI) $*.o

%.cxoi.a: %.a
        $(CXOI)$*.a

all: foo foo.for.cxpa

foo: $(OBJECTS)
        $(C89) $(OBJECTS) -o foo $(LIBS)

foo.for.cxpa: $(CXOI_OBJECTS) $(CXOI_LIBS)
      $(C89) +pa $(CXOI_OBJECTS) -o foo.for.cxpa $(CXOI_LIBS)
```

Limitations

Note the following limitations:

- `cxoi` cannot be used to instrument shared libraries.
- `cxoi` only supports routine-level profiling. It does not support profiling of loops.

- Using `cxoi` requires space in `/usr/tmp` (or in the directory specified by the environment variable `TMPDIR`) totaling at most three times the size of the file being instrumented. If `/usr/tmp` does not have the required amount of space, you can set the `TMPDIR` environment variable to point to a different directory with sufficient space.
- Routines whose names begin with one or more leading underscores (_), millicode, and routines declared static (C or C++) are never exposed for profiling.
- CXpa may not support source code correlation for any routines exposed for profiling using the `cxoi` utility.
- Object files and archive libraries instrumented for profiling with `cxoi` do not contain source file line number information. This means that source code correlation for routines within these modules always refer to line 1 of the source file that contains the routine.

  CXpa source code annotations are not displayed in the Source Code window or in source file listings for object files and libraries instrumented with `cxoi`.

Known problems

CXpa may display the following message:

```
ERROR D5: Cannot find symbolic support in current
executable.
```

This message can be ignored; performance analysis is not affected.

Related Windows

Executable Manager window Profile Selection dialog

Related Concepts

Glossary
Introducing metrics
Learning CXpa quickly
Profiling strategy
Selecting and deselecting regions in line mode
Selecting metrics in line mode
Selecting metrics in GUI mode
Selecting regions in GUI mode

Related Commands

`analyze`
`cxpa`
`select`
`collect`
`run`
`set events`

Profiling with CXpa

3

This chapter covers the following topics:

- Learning CXpa quickly
- Profiling strategy
- Performance data files (PDFs)
- Analyzing PDFs only
- Using pre-instrumented executables
- Profiling MPI and PVM applications
- CXpa and the mpa utility
- Using batch mode
- Using online help

Procedures and examples in this chapter show how to perform profiling tasks using either the GUI or line mode interface.

Learning CXpa quickly

This section takes you step-by-step through a profiling session and briefly explains how to use CXpa's standard features in GUI mode and line mode.

The basic steps in the profiling process are as follows:

1. Prepare the program for profiling, either by compiling it with the `+pa` option or instrumenting your program's object files and libraries with the `cxoi` utility.
2. Invoke CXpa, specifying the name of the executable you want to profile.
3. Select the metrics you want to collect and the source code regions (that is, routines and loops) at which you want them to be collected.
4. Run the program and generate a performance data file (PDF).
5. Analyze the results in graphs and reports.

You will obtain the most accurate profiling results if you run the program in a standalone environment (that is, on a lightly loaded system or on a dedicated system or subcomplex).

Basic profiling using CXpa's GUI

To profile a program using CXpa's graphical user interface, follow these steps:

1. Compile and link your program with the `+pa` option if you are using an Exemplar compiler (Exemplar C, C++, or Fortran 77):

   For example:

   ```
   % /opt/fortran/bin/f77 +pa prog.f
   ```

   Refer to the "Compiling for CXpa" online help topic for details on compiling applications for CXpa.

   To instrument object files and archive libraries compiled with other HP PA-RISC targeting compilers, use the `cxoi` utility. `cxoi` is a separate utility supplied with CXpa.

   Refer to the `cxoi` man page or to the "Using cxoi to instrument object files and libraries" online help topic or section of this book for instructions.

2. Set your DISPLAY environment variable. For example, using C shell syntax:

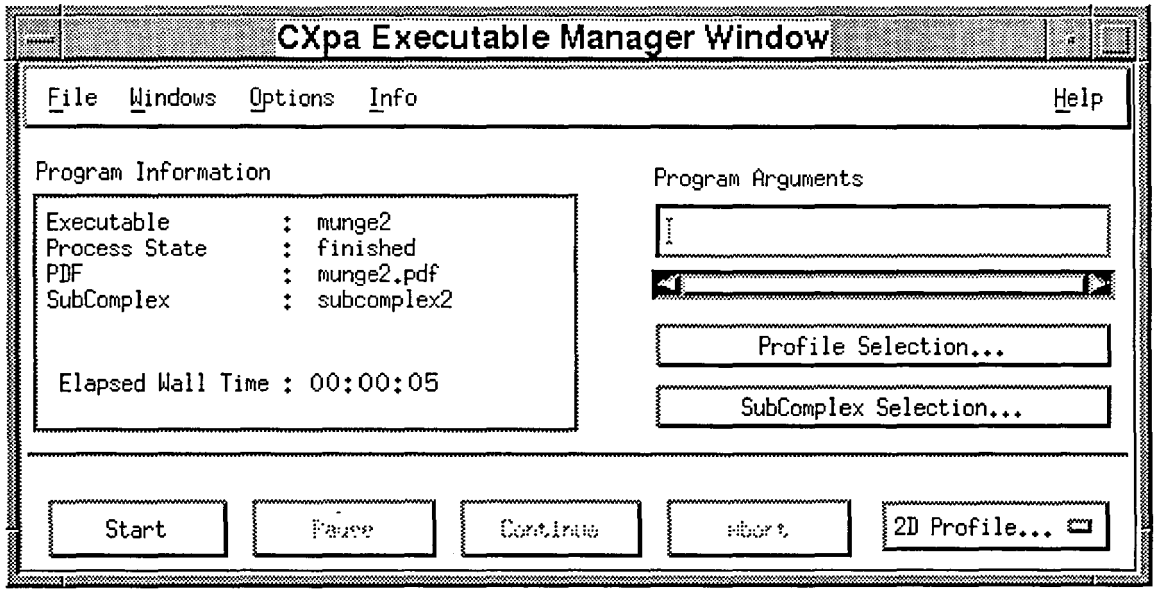
   % **setenv DISPLAY** *display_name*:0.0

   If your DISPLAY variable is not set, CXpa displays a message and starts in line mode.

3. Invoke CXpa with the name of an executable file. For example:

   % **/opt/cxpa/bin/cxpa a.out &**

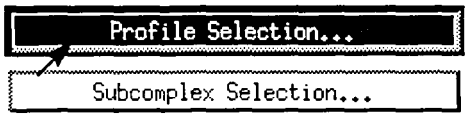
   CXpa opens an Executable Manager window.



By default, CXpa collects CPU time, wall clock time, and execution counts for all routines in your program. Use these defaults the first time you profile a program using CXpa. To accept the defaults, skip to step 5.

To select different source code regions (such as loops or parallel loops) for profiling or collect different metrics (such as events), continue with step 4.

4. Use the Profile Selection button to open a Profile Selection dialog. Select the regions you want to profile and the metrics you want to collect.

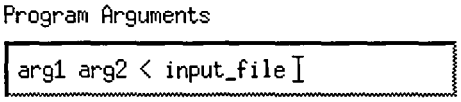


Profiling time and intrusion increase as you select more regions and metrics. Refer to the "Profiling strategy," "Selecting metrics in GUI mode" and "Selecting regions in GUI mode" online help topics or sections in this book for more information.

Steps 5 and 6 describe how to run the program under CXpa's control. You can also select Save or Save as from the File menu to write the instrumentation selections you just made to the executable file or to a copy. You can then exit CXpa and run the executable outside CXpa to generate performance data files.

Refer to the "Using pre-instrumented executables" online help topic or section of this book for more information.

5. Enter arguments to the program you are profiling in the Program Arguments field.



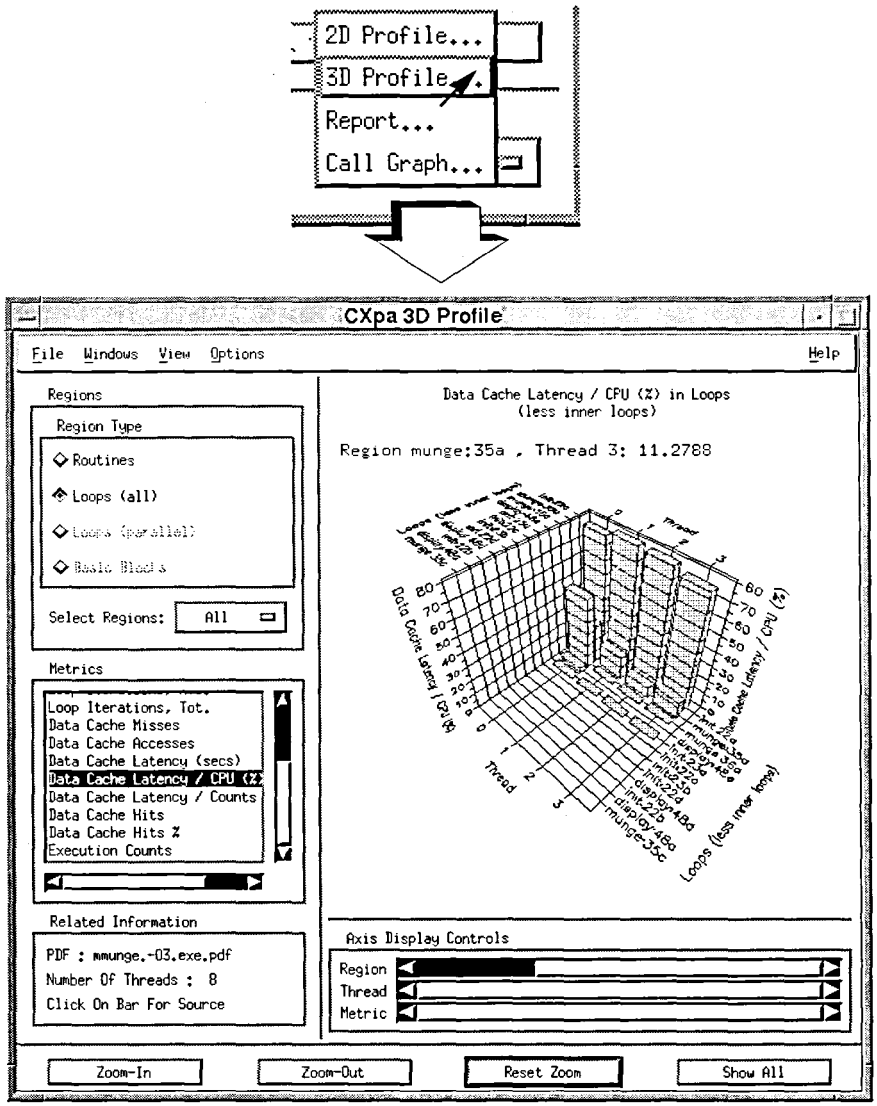
6. Select Start to run your program. An xterm window appears to display your program's input and output. A dialog also appears while CXpa is instrumenting your executable.

   Large programs may take a while to instrument.

   Once your program is instrumented, it starts executing and runs to completion unless you:
   - Select Cancel during instrumention.
   - Select Pause to suspend program execution during profiling. Once your program is paused, you can either select Continue or Abort.
   - Select Abort to terminate a paused program.

   To obtain the most accurate profiling data, do not pause your program during profiling.

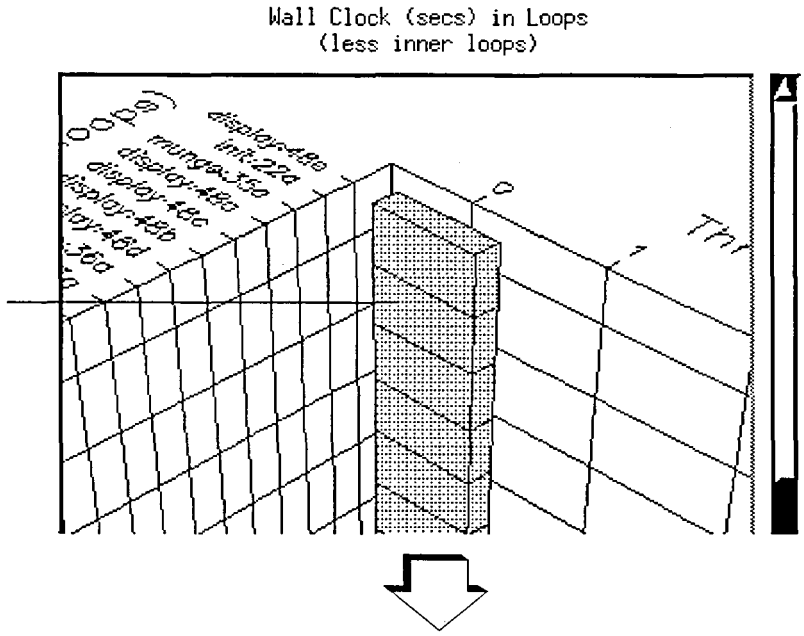
7. Display profile results by selecting one of the options from the button at the bottom-right corner of the Executable Manager window or from the Windows menu:
   - **2D Profile**—Displays a 2D graph of profiling data accumulated across all threads of the process. You can dynamically select regions and metrics to graph and display source code associated with bars in the graph.
   - **3D Profile**—Displays a 3D graph of profiling data for individual threads. You can dynamically select regions and metrics to graph and display source code associated with bars in the graph.
   - **Report**—Displays an Analysis Report window that contains textual performance reports.
   - **Call Graph** —Displays a dynamic call graph window that initially shows the top 10 routines in your program ranked by inclusive wall clock time (that is, including time spent in called routines). The Call Graph window features point-and-click navigation, clickbacks to source code, and access to detailed caller/callee and metric information for each routine.

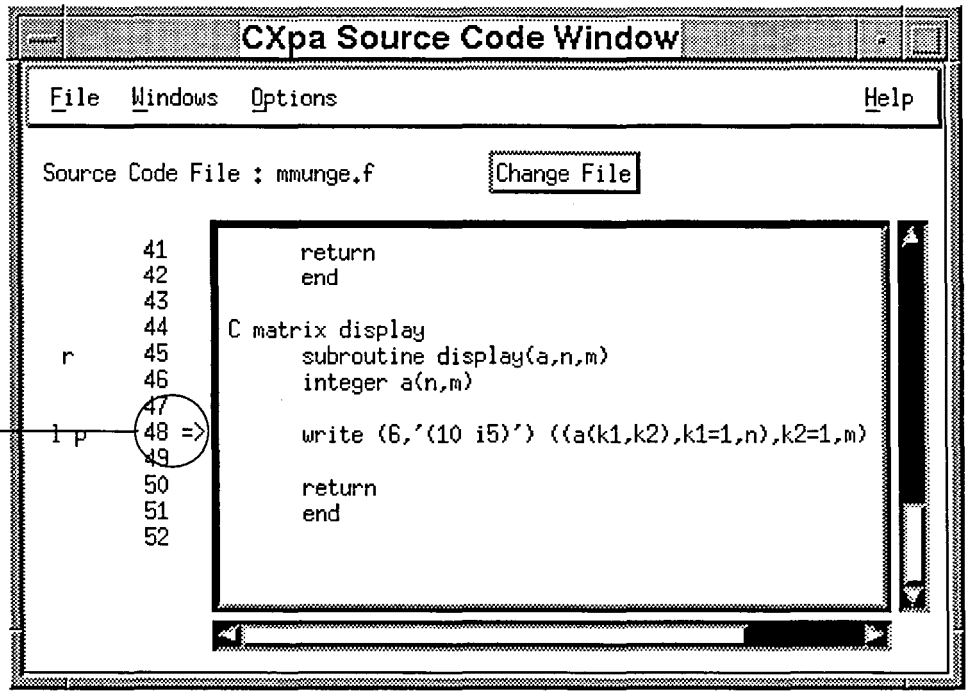


The above example shows a 3D graph of the ratio of data cache miss latency to CPU time (expressed as a percentage) for all profiled loop regions in a program running on an SPP1200 system.

8. To display the source code corresponding to a bar in the 2D or 3D graph, click on the bar with the left mouse button. The Source Code window appears with an arrow (=>) pointing to the start of the section of code represented by the bar you clicked.

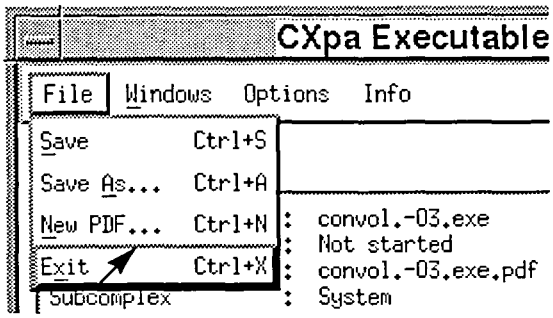
Left click on any bar on the graph to display source code associated with that region.





=> annotation indicates associated source code.

9. Select Exit from the File menu in the Executable Manager window to quit CXpa.



Refer to the "Profiling strategy" section of this book for more information about profiling intrusion and a step-by-step strategy for profiling.

Basic profiling using CXpa's line mode interface

If you are working from a CRT or if you prefer a line-oriented interface in an xterm window, you can use line mode. To invoke CXpa in line mode, you must use the `-nw` option at the shell command line.

To use CXpa in line mode:

1. Compile and link your program with the `+pa` option if you are using an Exemplar compiler (Exemplar C, C++, or Fortran 77).

   For example:

   ```
   % /opt/fortran/bin/f77 +pa prog.f
   ```

   Refer to the "Compiling for CXpa" online help topic or section of this book for more information on compiling applications for CXpa.

   To profile object files and archive libraries or applications compiled with other PA-RISC targeting compilers, use the `cxoi` utility. `cxoi` is a separate utility supplied with CXpa.

   Refer to the `cxoi` man page or to the "Using cxoi to instrument object files and libraries" online help topic or section of this book for instructions.

2. Invoke CXpa with the name of an executable file and the `-nw` (no windows) option:

   ```
   % /opt/cxpa/bin/cxpa -nw a.out
   ```

Your shell prompt changes to the CXpa line-mode prompt:

```
                CONVEX Performance Analyzer

Type 'help' for help.
Reading executable a.out...
Selecting profile a.out.pdf...

(CXpa)
```

As shown in the output above, CXpa displays the name of the executable file to be profiled and the name of the performance data file (PDF file) that profiling data is written to (by default, this file is named <*executable*>.pdf).

3. Select the source code regions of your program you want to profile with a form of the `select` command.

   For example:

   ```
   (CXpa) select routine all
   ```

   The above command selects all routines in your program for profiling. This is the recommended selection for the first time you profile your program under CXpa.

NOTE: In line mode, if you do not select one or more source code regions in your program for profiling with the `select` command, CXpa does not collect any metrics.

4. Choose the type of metrics that you want to collect with the `collect` command. By default, CXpa collects CPU time and wall clock time. The first time you profile your program, use the default selection.

   Each use of the `collect` command overwrites its previous setting.

   The following metrics can be specified with the `collect` command:

   - `cpu`—Collects CPU time.
   - `wall_clock`—Collects wall clock time.
   - `call_graph`—Collects dynamic call graph information.
   - `events`—Collects memory access events, such as data cache miss counts and latency time. If you enable event collection with this parameter, you must use the `set events` command to specify the type of events collected.
   - `counts`—By default, execution counts are always collected. Specify this parameter if you want CXpa to collect only execution counts (for example, `collect counts`).

   For example:

   ```
   (CXpa) collect cpu wall_clock
   ```

5. If you enabled event collection with the `events` parameter of the `collect` command, choose the type of event you want to collect with the `set events` command.

   Each use of the `set events` command overwrites its previous setting.

   For example:

   ```
   (CXpa) set events local_misses
   ```

   The above command tells CXpa to collect the number of times your program had to access hypernode-local memory to find a value due to a processor data cache miss. The `local_misses` parameter is specific to Exemplar S2000, Exemplar X2000, and SPP1600 Series systems.

   The number and type of events you can collect differ according to machine architecture. Refer to the online help topic or reference page for the `set events` command for more information.

   Step 6 describes how to run the program under CXpa's control. You can also write the instrumentation selections you just made to the executable file, exit CXpa, and run the executable outside CXpa to generate performance data files. Refer to the "Using pre-instrumented executables" online help topic or section of this book for more information.

6. Run your program with the `run` command:

   ```
   (CXpa) run
   ```

   Output from your program is sent to stdout as it runs.

   Your program runs to completion unless you press **CTRL-c** to pause it. Once you have paused the program, use the `continue` command to resume execution or the `stop` command to terminate the program.

   To obtain the most accurate profiling data, do not pause your program during profiling.

7. Use the `analyze` command to view performance reports after your program finishes. For example:

   ```
   (CXpa) analyze
   ```

   When you enter the `analyze` command without specifying any parameters, CXpa generates and displays all available performance reports. To display reports for a specific region type, use the `analyze routine`, `analyze loop`, or `analyze pregion` command.

CXpa displays reports in line mode using the pager specified with your PAGER environment variable. If the PAGER environment variable is not set, CXpa uses the `more` command to page the output. You can redirect output of the `analyze` command to a file using redirection operators.

8. Use the `quit` command to exit CXpa.

Refer to the "Profiling strategy" section of this book for more information about profiling intrusion and a step-by-step strategy for profiling.

Editing the command line

CXpa's line mode provides command-line editing functions similar to those available in `tcsh`. Enter **ESC-?** on the CXpa command line to display available editing functions. The editing functions are as follows:

Function	Key sequence
Backward character	**CTRL-b**
Backward word	**ESC-b**
Beginning of line	**CTRL-a**
Capitalize forward word	**ESC-c**
Delete backward character	**CTRL-h**
Delete backward character	**DEL**
Delete backward word	**ESC-h**
Delete forward character	**CTRL-d**
Delete forward word	**ESC-d**
Display key bindings	**ESC-?**
End of line	**CTRL-e**
Erase line	**CTRL-g**
Erase screen	**ESC-g**
Execute current command	**RETURN**
Execute a shell command	**!**<*command*>
Forward character	**CTRL-f**
Forward word	**ESC-f**
Kill to end of line	**CTRL-k**
Lower case word	**ESC-l**
Next command	**CTRL-n**
Previous command	**CTRL-p**
Transpose characters	**CTRL-t**
Transpose words	**ESC-t**
Upper case word	**ESC-u**

Related Topics

Compiling
Introducing metrics
Introducing source code regions
Profiling strategy
Profiling MPI and PVM applications with CXpa
Reports
Selecting and deselecting regions in line mode
Selecting metrics in line mode
Selecting metrics in GUI mode
Selecting regions in GUI mode
Using batch mode
Using cxoi to instrument object files and libraries
Using online help

Related Windows

2D Profile window
3D Profile window
Analysis Control window
Analysis Report window
Call Graph window
Executable Manager window
Help window
Profile Selection dialog

Related Commands

`analyze`
`collect`
`help`
`rerun`
`run`
`select`
`set events`
`set visibility`

Learning CXpa quickly

Profiling strategy

When running your application under CXpa with regions and metrics selected for profiling, you may notice that your code executes more slowly than expected. This can be due to profiling intrusion introduced by CXpa.

This section describes the causes and effects of profiling intrusion and provides a profiling strategy that will help you quickly locate regions of source code that cause performance bottlenecks and minimize profiling intrusion.

Profiling intrusion—what is it, and what causes it?

All methods of profiling are intrusive—that is, the overhead associated with profiling can affect the validity of the results.

When using CXpa, keep in mind that *the greater the number of regions and metrics selected for profiling during a program run, the greater the amount of profiling intrusion introduced—that is, time delays.*

These time delays occur when hardware counters that provide CPU time, wall clock time, cache performance, and instruction counts are accessed at data sampling points.

Each source code region enabled for profiling has a minimum of two data sampling points—a region entry point and a region exit point. The more data sampling points enabled, the greater the amount of profiling intrusion. Additional time delays can also occur when profiling data is stored during a program run. The greater the amount of profiling data that must be stored, the greater the time delay.

When only routines are selected for profiling, the profiling intrusion is minimal. Profiling loops is much more intrusive, because the number of data points that must be sampled is far greater, especially in loop nests or in loops with large iteration counts.

By default, CXpa profiles loops at nesting level 0 (outermost loops) only. This reduces the number of sampled data points and the amount of intrusion introduced when profiling loops. You can change the loop nesting level setting using the `set visibility` command (in line mode) or in the Profile Selection dialog (in GUI mode).

Profiling intrusion—an example

This example uses the following simplified source code region structure (after optimization):

```
ROUTINE (CALLED n TIMES)
   100 ITERATIONS OF LOOP AT NESTING LEVEL 0
      100 ITERATIONS OF LOOP AT NESTING LEVEL 1
         100 ITERATIONS OF LOOP AT NESTING LEVEL 2
```

The increase in the number of data sampling points enabled as more loops are selected for profiling is shown in the following table.

Source code regions selected for profiling	Number of sampled data points
Routines only	$2 * n$
All loops at nesting level 0 only	$(100 * 2) * n = 200 * n$
All loops at nesting level 1 only	$(100 * 2) * n = 200 * n$
All loops at nesting level 2 only	$(100 * 2) * n = 200 * n$
All loops at all nesting levels	$(100 * 2) * (100 * 2) * (100 * 2) * n = 8{,}000{,}000 * n$
All routines, All loops at all nesting levels	$(2 * n) + (8{,}000{,}000 * n)$

As illustrated above, profiling all loops at all nesting levels or profiling all available source code regions during a single program run results in a level of profiling intrusion that can produce misleading results. This is because the number of sampled data points in a loop nest grows by a factor of 2 * the number of iterations of the loop nest with each level of nesting.

Recommended profiling strategy

By using a top-down profiling strategy, you can minimize the number of regions and metrics selected for profiling during each run of your program. This will significantly reduce the amount of profiling intrusion and increase the validity of the results.

Region selection should proceed from coarse-grained (routines) to fine-grained (loops or parallel loops in specific routines) as code regions that exhibit performance problems are identified.

Remember these two key principles:

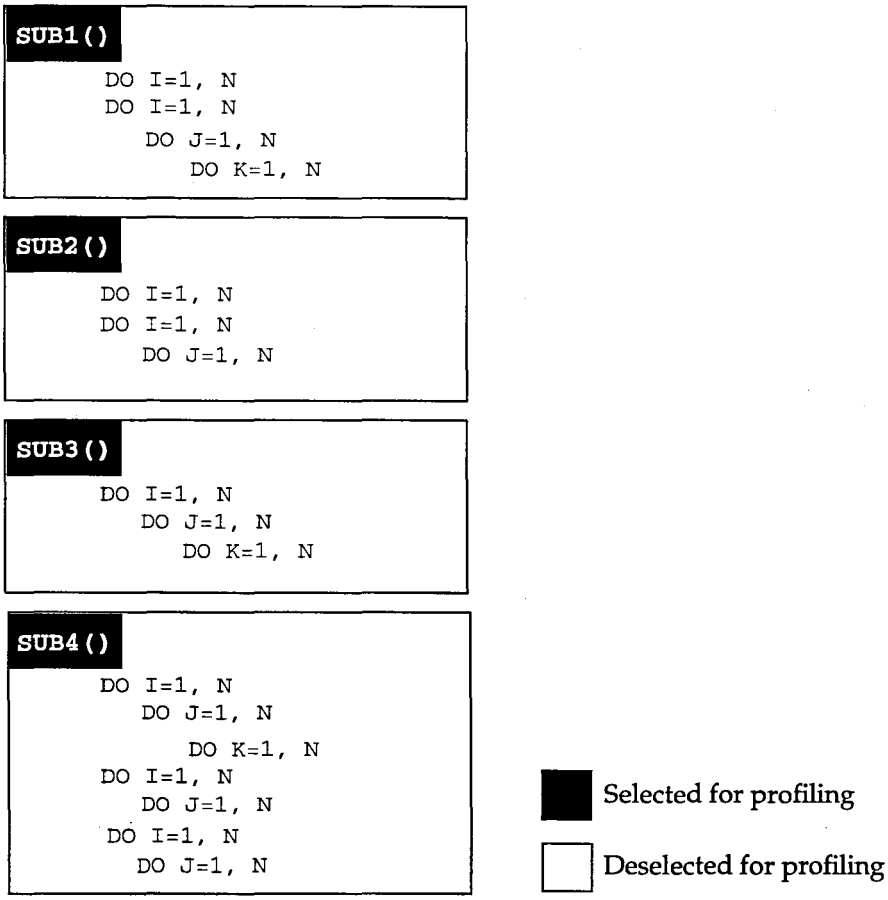
- Minimize the number of sampled data points per program run by selecting only the regions that are significant during each run and following the profiling strategy outlined below.
- Collect only the metrics you are interested in per program run (for example, do not enable event collection for a run unless you really need to examine cache performance or instruction metrics).

The procedure below outlines the recommended profiling strategy:

1. The first time you profile your program under CXpa:
   - Select all routines (or fewer, if you already identified the critical routines) for profiling at the routine region level.
   - Collect wall clock and CPU time metrics.

   This provides a complete picture of your program's performance. Using this information, you can then identify the routines that take the longest to execute. The following example illustrates the first profiling run selections.

Example—First profiling run region selections

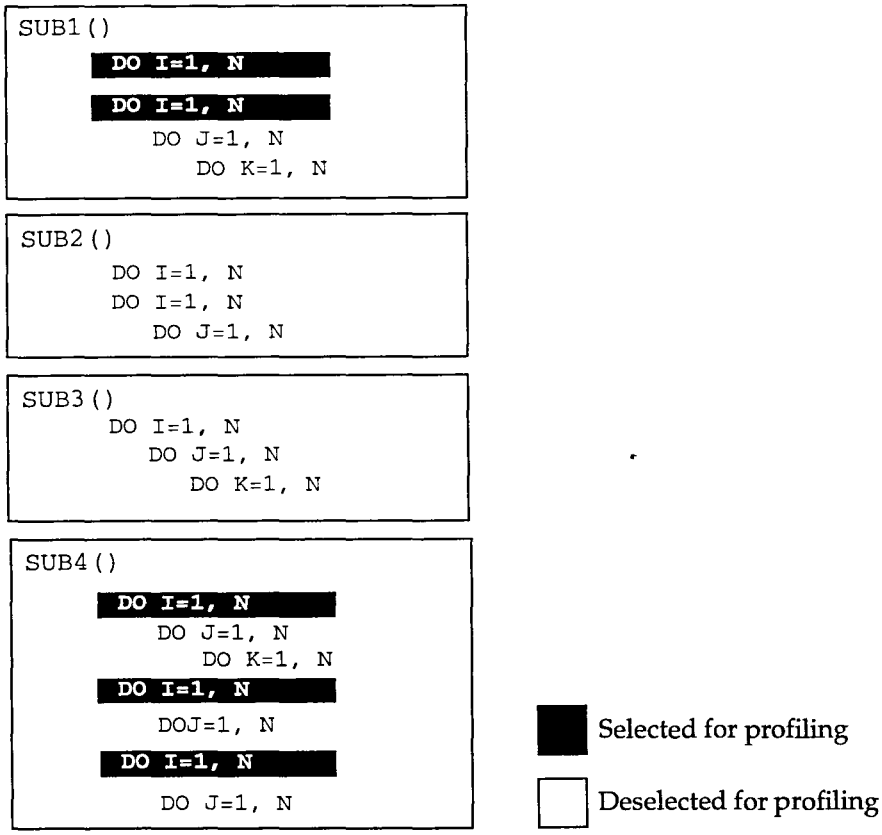


2. Next, profile only the critical routines whose performance you want to improve. Select all loops at loop nesting level 0 within those routines for profiling (as shown in the following example).

   This gives you a loop-level view of these routines without incurring the profiling intrusion associated with selecting all loops at all nesting levels. Continue to collect CPU time and wall clock time.

   Using this method, you are profiling a section (or slice) of the loops in these routines that contains only the outermost loops (loop nesting level 0).

Example—Second profiling run region selections



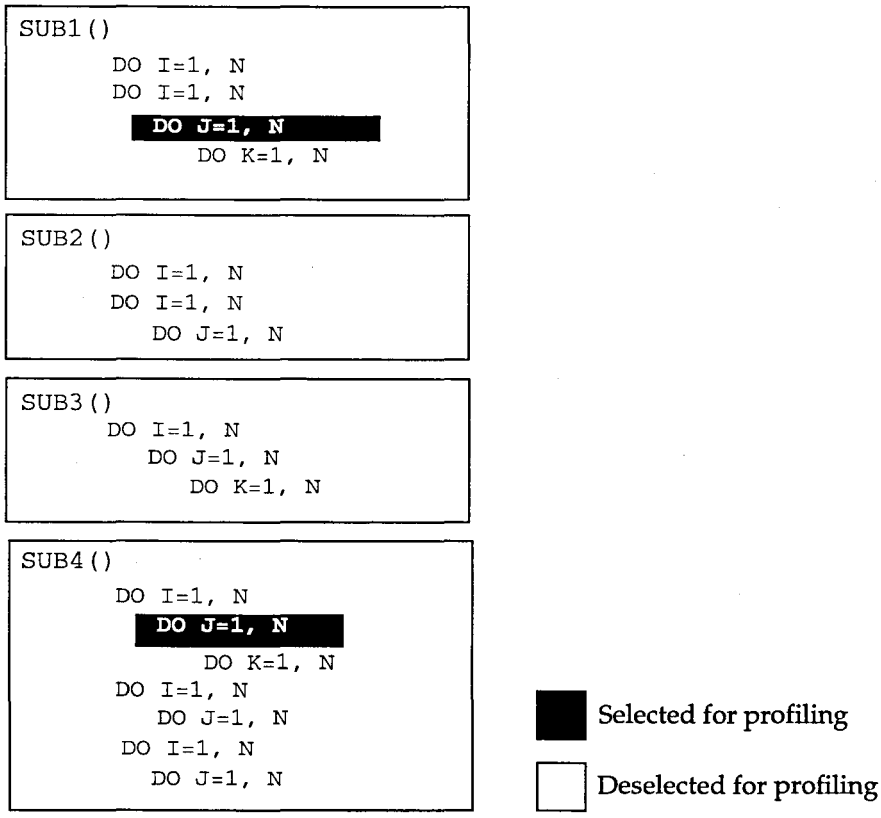
Refer to the following sections of this book for instructions on how to accomplish this using CXpa's GUI or command-line interface:

- Profile Selection dialog
- `select` and `set visibility` commands

Rerun your program under CXpa to identify the loops or loop nests within the selected routines that consume the most wall clock time or CPU time.

3. On subsequent runs of your program, you can select different sections or "slices" of the loops within the routines selected for profiling, as shown in the following example. When specifying a fixed range of loop nesting levels, set the minimum loop nesting loop level equal to the maximum loop nesting level.

Example—Third profiling run region selections



4. Once you have identified the loops that are causing the performance problem, you can also choose to collect different metrics at these regions, such as cache misses and latency. With fewer source code regions selected, less time is spent in the timing routines CXpa uses to collect performance data. As a result, the profiling information obtained is more accurate.

Related Topics

Introducing metrics
Introducing source code regions
Selecting and deselecting regions in line mode
Selecting metrics in line mode
Selecting metrics in GUI mode
Selecting regions in GUI mode

Related Windows

Profile Selection dialog

Related Commands

`collect`
`select`
`set events`
`set visibility`

Performance data files

When you use CXpa to profile a program, it creates a performance data file (PDF) to store profiling data. The PDF is a binary file that contains the performance data for a single run of your program.

The data in a PDF is used to create 2D and 3D profile graphs and analysis reports. If you do not specify a different PDF name, CXpa uses the default PDF name of *<executable>*.pdf.

PDFs are platform independent. For example, the data stored in a PDF created on an Exemplar S2000 system can be viewed (in reports or graphs) on an SPP1600 Series system, or vice versa.

If you start CXpa with an executable, you can only analyze a PDF created during the current CXpa session. To analyze a PDF created with a different executable or multiple PDF files, start CXpa with the name of a single PDF file or a list of PDF files (without specifying an executable).

Changing the PDF name

You can change the name of the PDF to be written to or read from during a CXpa session. You may want to do this for two reasons:

- **To prevent CXpa from overwriting an existing PDF**—If you invoked CXpa with the name of an executable, and there is an existing PDF that has the same name as the current PDF, CXpa overwrites all of the data in that PDF when you run your program. If you do not want this to occur, you must change the name of the PDF between runs of your program, as described in the following sections.
- **To analyze a different PDF**—If you have invoked CXpa with the name of a PDF file only, you can analyze PDFs created in previous CXpa sessions, including PDFs created on different architectures or from different executables. In this analysis-only mode, you can analyze and compare data for several PDFs during a CXpa session.

In GUI mode

To specify a new PDF file in GUI mode, perform the following steps from the Executable Manager or Analysis Control window:

1. Select New PDF from the File menu in the Executable Manager window or Open PDF from the File menu in the Analysis Control window. CXpa opens a New PDF or Open PDF dialog.

2. Specify the new PDF name by highlighting the name of a file in the Files list or by typing a new name in the PDF Selection field.
3. Choose OK.

CXpa will now write to or read from the selected PDF during this profiling session.

- If you invoked CXpa with the name of an executable file, profiling data are written to the specified PDF when you run your program, and performance analysis reports and graphs are generated from the data in that PDF.
- If you invoked CXpa with the name of a PDF file or files only (without specifying an executable file), performance analysis reports and graphs are generated from the data in the PDFs that you specified.

In line mode or batch mode

To choose a new PDF name in line mode or batch mode, use the `set pdf` command:

`(CXpa)` **`set pdf`** *<new_pdf_name>*

CXpa will now write to or read from the specified PDF during this profiling session.

Related Topics

Analyzing PDFs only

Related Windows

Analysis Control window
Executable Manager window
Merge PDFs dialog
New PDF dialog
Open PDF dialog

Related Commands

`analyze`
`cxpa`
`merge`
`set pdf`

Analyzing PDFs only

To view performance data in PDFs that were created in previous CXpa sessions, start CXpa from the shell with the name of a PDF file or files (without specifying an executable). In this analysis-only mode of CXpa, you can analyze profiling data, but you cannot instrument or run your program.

In analysis mode, you can analyze PDFs that were created in previous CXpa sessions, including PDFs that were created on different architectures or from different executables.

You can also merge PDF files created from multiple runs of the same executable into a single PDF file for comparison and analysis. This feature is useful when profiling message-passing applications or in comparing performance of an application when run with different data sets.

Refer to the "merge," "Merge PDFs dialog," and "Profiling MPI and PVM applications" online help topic or section of this book for more information.

To specify automatic creation of profile or report windows when you invoke CXpa in GUI mode with the name of a PDF file (without specifying an executable), use the X resource `Cxpa*defaultWindow`. It accepts the following values: `All`, `None`, `Callgraph`, `2DProfile`, `3DProfile`, `Report`, or `Source`. The default is `None`.

Analyzing PDFs only in GUI mode

To analyze PDFs only in GUI mode, start CXpa from the shell with the name of an existing PDF file or a list of PDF files.

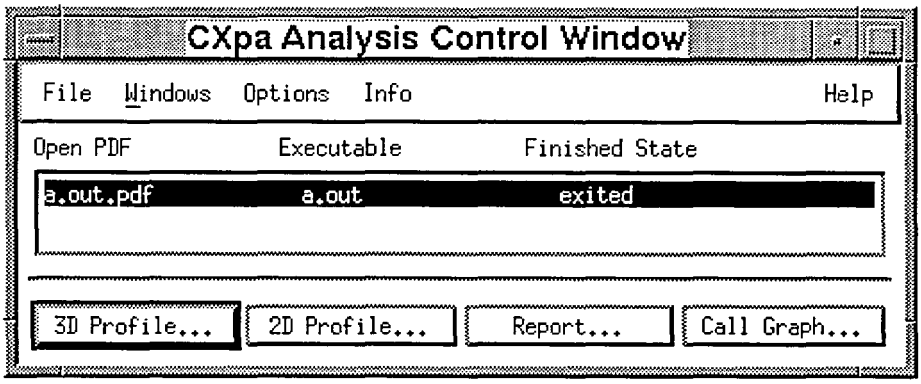
If you specify a list of PDF files, separate each file name with a space. You can also use shell wildcard characters, such as * to specify a multiple PDF files. For example:

```
% cxpa myprog.run1.pdf myprog.run2.pdf &
```

or

```
% cxpa *.pdf &
```

CXpa opens an Analysis Control window.



Each of the specified PDF files is automatically listed in the Open PDF column in the Analysis Control window, and the last PDF file specified is highlighted. Use the 2D Profile, 3D Profile, Call Graph, and Report buttons to analyze the data in the selected PDF.

To select or deselect a PDF file for analysis, highlight its name in the list.

Opening other PDFs for analysis

To open another PDF and display its name in the Open PDF list:

1. Choose Open PDF from the File menu. The Open PDF dialog appears.
2. Specify the new PDF name by highlighting the name of a file in the Files list or by typing a new name in the PDF Selection field.
3. Choose OK. The name of the specified PDF is appears in the Open PDF list in the Analysis Control window.

Choosing an active PDF

In analysis-only mode, the *active* PDF is the PDF file that CXpa reads from to create performance reports and graphs. In GUI mode, the name of the active PDF is highlighted in the Open PDF list in the Analysis Control window.

To specify another PDF file as the active PDF, highlight its name in the Open PDF list in the Analysis Control window.

Analyzing PDFs only in line mode

To analyze PDFs only in line mode, start CXpa from the shell with the name of an existing PDF file or a list of PDF files and specify the `-nw` (no windows) option.

If you specify a list of PDF files, separate each file name with a space. You can also use shell wildcard characters, such as * to specify multiple PDF files. For example:

```
% cxpa myprog.run1.pdf myprog.run2.pdf -nw

                        CONVEX Performance Analyzer

Type 'help' for help.
Selecting profile myprog.run1.pdf...
Selecting profile myprog.run2.pdf...
(CXpa)
```

or

```
% cxpa -nw *.pdf

                        CONVEX Performance Analyzer

Type 'help' for help.
Selecting profile a.out.20703.pdf...
Selecting profile a.out.20704.pdf...
Selecting profile a.out.20705.pdf...
Selecting profile a.out.20706.pdf...
Selecting profile a.out.20707.pdf...
Selecting profile a.out.20708.pdf...
Selecting profile a.out.20709.pdf...
Selecting profile a.out.20710.pdf...
Selecting profile a.out.20711.pdf...
(CXpa)
```

When multiple PDF file names are specified in line mode, each PDF file is listed as it is specified on the shell command line. The last PDF file name listed is selected as the *active* PDF. In analysis mode, the active PDF is the PDF file that CXpa reads from to create performance reports and graphs. Use the `analyze` command to generate reports from the data in the active PDF. To display the name of the active PDF in line mode, use the `info` command.

Analyzing PDFs only

Opening other PDFs for analysis

If you invoked CXpa with a PDF only (without specifying an executable), you can analyze different PDFs without quitting CXpa. To change PDFs, use the `set pdf` command. For example:

```
(CXpa) set pdf myprog.run3.pdf
```

The specified PDF file becomes the active PDF; CXpa uses the data in that PDF to generate performance reports.

Related Topics

Performance data files

Related Windows

Analysis Control window
Executable Manager window
Merge PDFs dialog
New PDF dialog
Open PDF dialog

Related Commands

`analyze`
`cxpa`
`info`
`merge`
`set pdf`

Using pre-instrumented executables

You can write profile selection settings (instrumentation) to the current executable file or to a copy of the current executable. This is referred to as *pre-instrumenting* an executable. Use pre-instrumented executables to:

- Profile with CXpa in environments that do not support CXpa controlling a child process.
- Profile applications in conjunction with tools such as MPI or PVM that replicate processes or with applications where a driver program or script starts the process.

  Refer to the "Profiling MPI and PVM applications with CXpa" online help topic or section of this book for information about using pre-instrumented executables when profiling message-passing applications with CXpa.
- Maintain separate copies of an executable with different regions and metrics selected for profiling. This makes it easier to generate multiple PDF files for comparison and analysis.
- Profile applications run with the `mpa` utility. Refer to the "CXpa and the `mpa` utility" online help topic or section of this book for more information.

When you run a pre-instrumented executable file (outside the control of CXpa), profiling data is collected in a performance data file (PDF) for later analysis. The resulting PDF file will be named *<executable>*`.`*<pid>*`.pdf`. The *pid* is the program's UNIX process ID. You can also profile the new executable in the usual way (that is, by invoking CXpa with the name of the executable and running it under CXpa).

NOTE: When pre-instrumenting an executable on a different architecture than the one the executable will be run on to generate profiling data, you must specify the `-tm` *<architecture>* option when you start CXpa. This ensures that the correct timing routines are called to collect metrics for the target system. Valid values for *<architecture>* are as follows:

`s2000`—Exemplar S2000
`x2000`—Exemplar X2000
`spp1200`—SPP1200 Series
`spp1600`—SPP1600 Series

For example, if you are running CXpa on an SPP1200 to pre-instrument an executable that will be run on the Exemplar X2000, start CXpa as follows:

```
% /opt/cxpa/bin/cxpa -tm x2000
```

When you write instrumentation to a copy of the executable file, the resulting executable is an exact copy of the executable being profiled, except that it contains the current source code region and metric selections. All source code correlation is maintained. The size and permissions of the executable do not change, but the time stamp on the executable does.

Using the PROFDIR environment variable

The directory where the PDF files are created by a pre-instrumented program is controlled by the environment variable `PROFDIR`. If `PROFDIR` does not exist, *<executable>.<pid>*.pdf is created in the directory the application completes execution in (usually the directory from which the application is invoked).

If the `PROFDIR` environment variable is set to

`PROFDIR=`*string*

string/*<executable>*`.`*<pid>*`.pdf` is used as the name for each PDF file, where *pid* is the program's UNIX process ID.

If the `PROFDIR` environment variable is an empty string, no PDF file is created (although data is collected).

The next two sections describe how to use pre-instrumented executables with CXpa's line mode and GUI interfaces.

Using pre-instrumented executables in line mode

The following procedure demonstrates how to use CXpa's command-line interface to:

- Pre-instrument an executable
- Run the executable from the shell to generate a PDF
- View performance graphs and reports generated from the data in the PDF

1. Invoke CXpa from the shell with the name of the executable and the `-nw` option. The executable must be prepared for profiling with CXpa.

```
% /opt/cxpa/bin/cxpa -nw a.out

            CONVEX Performance Analyzer

Type 'help' for help.
Reading executable a.out...
Selecting profile a.out.pdf...
```

2. Select regions and metrics for profiling.

```
(CXpa) select routine all
(CXpa) collect cpu wall_clock
```

The `select routine all` command selects all routine regions in the program for profiling. The `collect cpu wall_clock` command tells CXpa to collect wall clock time and CPU time metrics.

Refer to the "Selecting and deselecting regions in line mode," "Selecting metrics in line mode," and the "Introducing metrics" online help topics or sections of this book for more information about selecting regions and metrics for profiling.

3. Use the `save executable` command to write the instrumentation to the executable or to a copy of the executable:
   - When you execute the `save executable` command without specifying a file name, CXpa writes the instrumentation to the current executable without changing its name.
   - When you specify a file name with the `save executable` command, CXpa writes the instrumentation to a copy of the executable, using the specified file name.

   In the following example, the instrumentation is written to the file `a.out.inst`, which is a copy of the `a.out` executable.

```
(CXpa) save executable a.out.inst
```

4. Quit CXpa.

```
(CXpa) quit
```

5. Run the executable from the shell to generate a PDF file. CXpa automatically names the PDF file *<executable>.<pid>*.pdf.

   For example, if you saved the instrumentation to a copy of the `a.out` executable named `a.out.inst`:

```
% a.out.inst
% ls *.pdf
a.out.inst.1324.pdf
```

6. Invoke CXpa with the name of the resulting PDF file.

```
% /opt/cxpa/bin/cxpa -nw a.out.inst.1324.pdf
```

7. Use the `analyze` command to generate and view text reports of the performance data in the PDF file.

```
(CXpa) analyze
```

Using pre-instrumented executables in GUI mode

The following procedure demonstrates how to use CXpa's X/Motif GUI to:

- Pre-instrument an executable
- Run the executable from the shell to generate a PDF
- View performance graphs and reports generated from the data in the PDF file

1. Set your `DISPLAY` environment variable. For example, using C shell syntax:

   ```
   % setenv DISPLAY display_name:0.0
   ```

2. Start CXpa with the name of the executable. The executable must be prepared for profiling with CXpa.

   ```
   % /opt/cxpa/bin/cxpa a.out &
   ```

   The Executable Manager window appears.

3. Use the Profile Selection button in the Executable Manager window to open a Profile Selection dialog for selecting regions and metrics to profile. When you have finished making selections, choose OK.

   The default selections are CPU time and wall clock time metrics for all routines and should be used the first time you profile your program with CXpa.

   Refer to the "Profile Selection dialog" and the "Introducing metrics" sections of this book for more information about selecting regions and metrics for profiling.

4. Select Save from the File menu in the Executable Manager window to write the instrumentation to the current executable without changing its name

   or

   Select Save As from the File menu in the Executable Manager window. CXpa opens a Save Executable As dialog where you can specify the name of the new executable file.

5. Select Exit from the File menu in the Executable Manager window to quit CXpa.

6. Run the instrumented executable to generate a PDF file. CXpa automatically names the PDF file *<executable>.<pid>*.pdf.

   For example, if you saved the instrumentation to a copy of the `a.out` executable named `a.out.inst`:

   ```
   % a.out.inst
   % ls *.pdf
   a.out.inst.1324.pdf
   ```

7. Invoke CXpa with the name of the resulting PDF file.

   ```
   % /opt/cxpa/bin/cxpa  a.out.inst.1324.pdf &
   ```

   CXpa opens an Analysis Control window.

8. Use one of the buttons at the bottom of the Analysis Control window to generate and view 2D and 3D graphs or text reports from the performance data in the PDF file.

Related Concepts

Compiling
CXpa and the mpa utility
Introducing metrics
Introducing source code regions
Performance data files
Profiling MPI and PVM applications with CXpa
Reports
Selecting and deselecting regions in line mode
Selecting metrics in line mode
Selecting metrics in GUI mode
Selecting regions in GUI mode
Using batch mode

Related Windows

Analysis Control window
Profile Selection dialog
Executable Manager window
Save Executable As dialog

Related Commands

`analyze`
`run`
`select`
`collect`
`save executable`
`set events`

Profiling MPI and PVM applications with CXpa

You can profile an MPI (message-passing interface) or a PVM (parallel virtual machine) application using CXpa by generating a separate perfomance data file (PDF) for each of the application processes. This allows you to simultaneously profile all processes in a program. During analysis, you can combine the data from these PDF files into a single PDF file.

Generating PDF files

To generate profiling data from an MPI or PVM application using CXpa, perform the following steps:

1. Prepare the application for profiling with CXpa as described in Chapter 2, "Preparing programs for profiling."

   For information on building applications using MPI on Exemplar and SPP Series systems, refer to the *HP MPI User's Guide.*

   For information on building applications using PVM on Exemplar and SPP Series systems, refer to the *HP PVM User's Guide.*

2. Use CXpa to instrument the application. At the shell prompt, invoke CXpa with the name of the executable. For example:

   ```
   % /opt/cxpa/bin/cxpa my_app.exe
   ```

3. Select regions and metrics for profiling from within CXpa:

   - In line mode, use the `select`, `collect`, and `set events` commands to select regions and metrics for profiling. For example:

     ```
     (CXpa) select routine all
     (CXpa) collect cpu wall_clock events
     (CXpa) set events local_misses
     ```

     The above series of commands selects all routines for profiling and enables collection of CPU time, wall clock time, and locally resolved cache miss metrics for all routines.

   - In GUI mode, use the Profile Selection dialog to select source code regions and metrics for profiling.

4. Write the CXpa instrumentation selections to the executable file (that is, pre-instrument the executable):
   - In line mode, use the `save executable` command. Executing the `save executable` command without specifying a filename writes the instrumentation to the executable, modifying it in place. To write the instrumentation to a copy of the executable, specify a filename with the `save executable` command.
   - In GUI mode, select Save from the File menu in the Executable Manager window to modify the executable in place, or select Save As from the File menu to open a dialog where you can specify the name of a copy of the executable file to write the instrumentation to.
5. Exit CXpa.

   ```
   (CXpa) quit
   ```
6. Run the MPI or PVM application from the shell as you normally would.

When you run the executable, a separate PDF file is generated for each application process. The process ID (PID) of each process is inserted into the name of the PDF file generated, uniquely naming each of the resulting PDF files using the format *<executable>.<pid>*.pdf.

This is necessary because all the processes are likely to have the same name (*<executable>*.pdf, by default), which would result in CXpa successively overwriting the PDF file associated with each process.

The PDF file for the parent process usually (but not always) has the lowest-numbered PID.

NOTE: If a process forks, but does not perform an exec(), CXpa will only profile the parent process.

The directory where the PDF files are created by a pre-instrumented executable is controlled by the environment variable `PROFDIR`. If `PROFDIR` does not exist, *<executable>.<pid>*.pdf is created in the directory the application completes execution in (usually the directory from which the application is invoked).

If the `PROFDIR` environment variable is set to

`PROFDIR=`*string*

string`/`*<executable>*`.`*<pid>*`.pdf` is used as the name for each PDF file, where *pid* is the program's UNIX process ID.

If the `PROFDIR` environment variable is an empty string, no PDF file is created (although data is collected).

Analyzing profiling data from multiple PDF files

Once the PDF files are generated, use the following procedure to analyze the profiling data using CXpa's X/Motif GUI.

For a line mode (tty interface) example, refer to the "merge" command online help topic or section of this book.

1. Set your DISPLAY environment variable. For example, using C shell syntax:

   ```
   % setenv DISPLAY display_name:0.0
   ```

2. Start CXpa in GUI mode, specifying the names of the PDF files generated by the PVM or MPI application. For example:

   ```
   % /opt/cxpa/bin/cxpa my_app.exe.*.pdf
   ```

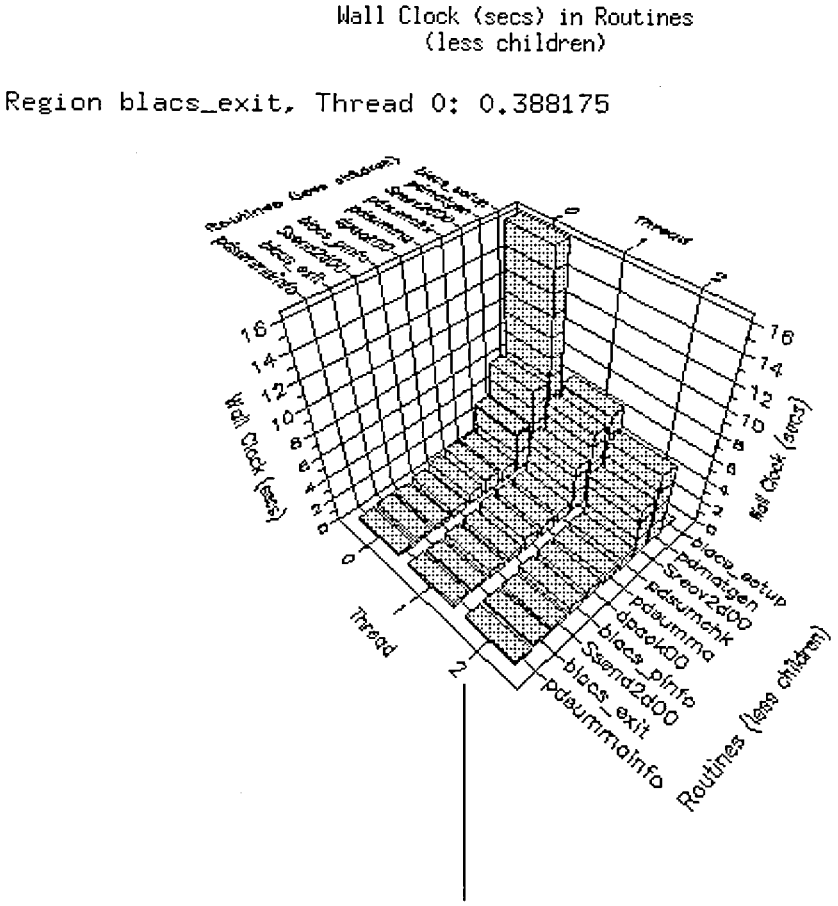
   The above command invokes CXpa with multiple PDF files in analysis-only mode using its X/Motif GUI interface.

   The Analysis Control window is displayed, and the name of each PDF file specified is displayed in the PDF list. The name of the last PDF file specified on the command line is selected as the current PDF.

   Once the PDF files have been loaded into CXpa, use the Merge feature to combine the data from the specified PDFs into a single PDF for analysis.

3. Select Merge from the File menu. CXpa opens a Merge PDFs dialog.
4. Specify the name of the PDF file that will contain the merged data, then close the dialog.
5. Select Open PDF from the File menu. CXpa opens an Open PDF dialog.
6. Specify the name of the PDF file that contains the merged data, then close the dialog. The new PDF name becomes the current PDF and is highlighted in the PDF list.
7. Use the buttons at the bottom of the Analysis Control window (3D Profile, 2D Profile, Report, or Call Graph) to display profiling data from the merged PDF file.

   In the 3D Profile window, the profiling data for each process is mapped to the thread axis in the 3D Profile graph, as shown in the following example.

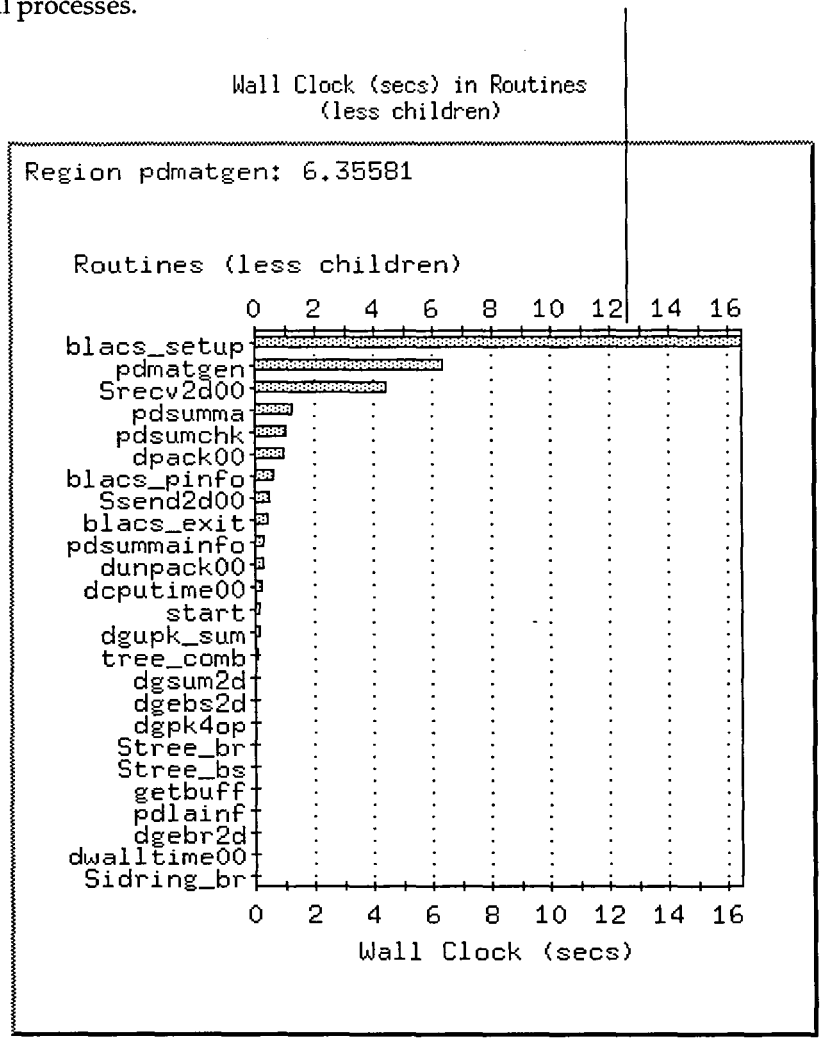


In this 3D Profile graph of a PVM application, the numbers on the thread axis represent individual processes.

Each bar in the graph represents the total wall clock time spent in a routine region for a single process.

In the 2D Profile window, the profiling data in the 2D Profile graph represents the sum of the data for each region across all processes, except for wall clock time. For wall clock time, the bars in the graph represent the maximum amount of wall clock time spent in each routine across all processes, as shown in the following example:

In this 2D Profile graph of a PVM application, each bar in the 2D profile graph represents the maximum amount of wall clock time spent in the associated routine for all processes.



Related Windows

2D Profile window
3D Profile window
Analysis Control window
Analysis Report window
Call Graph window
Executable Manager window
Profile Selection dialog

Related Commands

`add path`
`analyze`
`collect`
`continue`
`cxpa`
`deselect`
`merge`
`path`
`rerun`
`run`
`select`
`set events`
`set pdf`
`set subcomplex`
`set visibility`
`source`
`stop`
`quit`

Related Concepts

Compiling
Introducing metrics
Learning CXpa quickly
Reports
Selecting and deselecting regions in line mode
Selecting regions in GUI mode
Selecting metrics in line mode
Selecting regions in GUI mode
Using cxoi to instrument object files and libraries
Using pre-instrumented executables

CXpa and the mpa utility

To profile applications run with the `mpa` (modify process attributes) utility on Exemplar or SPP Series systems, you must:

- Pre-instrument the executable for profiling with CXpa.
- Exit CXpa.
- Run the pre-instrumented executable with `mpa` to generate a PDF file.

The following example shows how to accomplish this using CXpa's line mode interface.

```
% /opt/cxpa/bin/cxpa -nw a.out

        CONVEX Performance Analyzer

Type 'help' for help.
Reading executable a.out...
Selecting profile a.out.pdf...
(CXpa) select routine all
(CXpa) collect wall_clock cpu
(CXpa) save executable as a.out.inst
(CXpa) instrumenting, enter <CTRL+C> to abort.
...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
(CXpa) instrumentation finished.
(CXpa) quit
% /bin/mpa <mpa_options> a.out.inst
% /opt/cxpa/bin/cxpa a.out.inst.15237.pdf
```

The commands in the above example demonstrate how to profile executables run with the `mpa` utility:

- The command `/opt/cxpa/bin/cxpa -nw a.out` runs CXpa from the shell in line mode (`-nw`) and specifies the executable file `a.out` for profiling.
- The command `select routine all` selects all routines in the program for profiling.
- The command `collect cpu wall_clock` enables collection of CPU and wall clock time.

- The command `save executable a.out.inst` writes the instrumentation specified by the `collect` and `select` commands to a copy of the executable file name that is named `a.out.inst`.
- The `quit` command exits CXpa.
- The command `/bin/mpa` *<mpa_options>* `a.out.inst` runs the instrumented executable file `a.out.inst` from the shell using `mpa` to modify the program's process attributes with the specified options. As the program runs, profiling data is collected in the file a.out.inst.*<pid>*.pdf (the program's UNIX process ID, *pid*, is inserted into the name of the PDF file).
- The command `/opt/cxpa/bin/cxpa a.out.inst.15237.pdf` runs CXpa in GUI mode from the shell and specifies the PDF file a.out.inst.15237.pdf for analysis.

Refer to the man page for the `mpa` utility for information about using the `mpa` utility and its options. Refer to the "Using pre-instrumented executables" online help topic or section of this book for detailed information about pre-instrumenting executables.

Related Topics

Using pre-instrumented executables

Using batch mode

This section describes how to use CXpa in batch mode from the command line or from a shell script.

Using batch mode from the command line

To use CXpa in batch mode from the command line, specify a command file for input when you start CXpa by using the `-x` option and redirect input and output:

`cxpa -x` *<cmdfile>* **`a.out <`** *<input_file>* **`>&`** *<output_file>*

The above syntax shows how to tell CXpa to execute a command file at startup, read input to your program from a file, and redirect all output and messages to a file.

Command file input using the `-x` option

You can tell CXpa to execute a command file from the command line by using the `-x` option. When you use the `-x` option, CXpa executes in batch mode. For example:

```
% cxpa -x cmdfile a.out
```

CXpa executes the command file and quits when it encounters the `quit` command or the end of the file. A CXpa command file is a text file with that contains a list of CXpa commands. Each command must appear on a separate line. The # symbol denotes comments.

The following file listing shows a sample CXpa command file:

```
#This line is a comment.
select routine all
collect cpu wall_clock
run
analyze > cxpa.report
quit
```

Argument input using the `-e` option

You can specify arguments to pass to the program you are profiling on the CXpa command line with the `-e` option. These arguments are used when you execute your program in CXpa with the `run` command. This option is helpful for integrating CXpa with existing program execution shell scripts and is available in CXpa's line mode, batch mode, and GUI interfaces.

The following example shows how to use the `-e` option to specify program arguments:

```
% cxpa -x cmdfile -e a.out 12 35 14
```

Following the `-e` option, CXpa expects the name of the executable (optional) followed by program arguments. No other CXpa options may follow the `-e` option.

The following section explains how to use CXpa with a shell script that compiles and runs a program.

Using batch mode from a script

The following example shows a way to integrate CXpa into a script that compiles and runs a program. This example assumes the use of the Exemplar compilers.

```
#!/bin/csh -f
#
#Name: batch_script
#Run this script with cxpa if command line switch -cxpa is found
#

set PROFILER = ''
set PROFILER_COMP_FLAG = ''
foreach arg ($argv)
  if ($arg == '-profile') then
    set PROFILER = '/opt/cxpa/bin/cxpa -x cmd_file -e'
    set PROFILER_COMP_FLAG = '+pa'
    cat << EOF >! cmd_file
      select routine all
      run
      analyze
      quit
EOF
  endif
end

# compile a.out
#
/opt/ansic/bin/c89 $PROFILER_COMP_FLAG +O0 foo.c -o a.out

# Now run the executable
#

$PROFILER a.out arg1 arg2
```

If you include the `-profile` option when you invoke this script, it will compile the program with the `+pa` option, invoke CXpa with the resulting executable file, and execute a CXpa command file that performs a batch profiling session.

To profile with CXpa in batch mode using this script, use the following command line:

```
% batch_script -profile
```

Related Commands

`add path`
`analyze`
`collect`
`continue`
`cxpa`
`deselect`
`path`
`rerun`
`run`
`select`
`set events`
`set pdf`
`set subcomplex`
`set visibility`
`source`
`stop`
`quit`

Using online help

In GUI mode, you can access the Help window by selecting Help from a CXpa dialog or by choosing an option from any Help menu in the upper-right corner of any CXpa window.

In line mode, use the `help` command to access text-only online help. Refer to the reference page for the `help` command for information about accessing online help for commands in line mode.

To view the online release notice for the version of CXpa you are using, select Release Notice from the Help menu of the Executable Manager window or Analysis Control window.

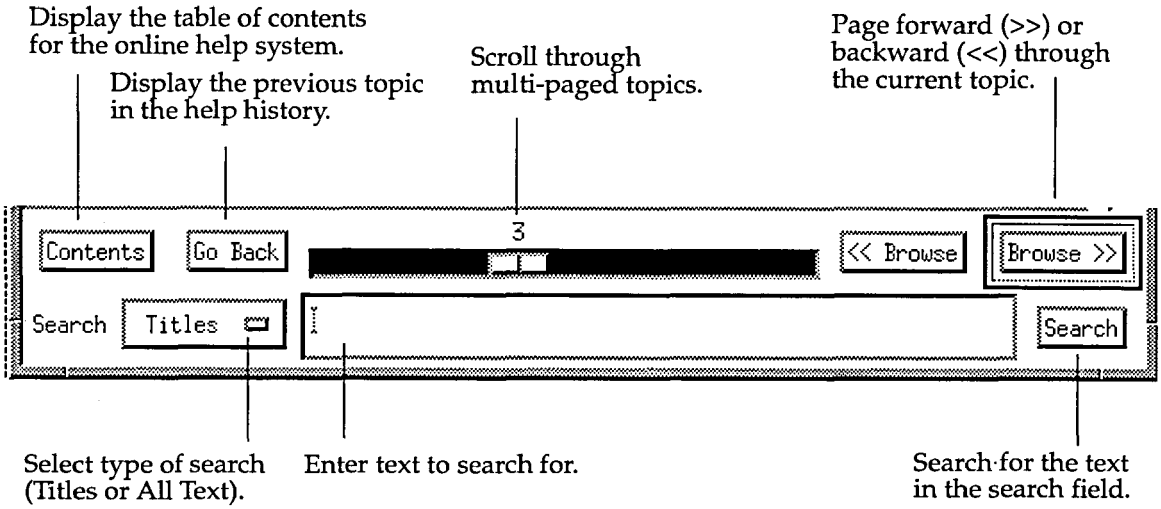
This section explains how to use the following features of the Help window:

- Using the buttons and scroll bars
- Using the menus
- Selecting a topic
- Searching for a topic
- Exiting from help

You can access this help page online by selecting the Using Help option of the Help menu in the upper-right corner of any CXpa window.

Using the buttons and scroll bars

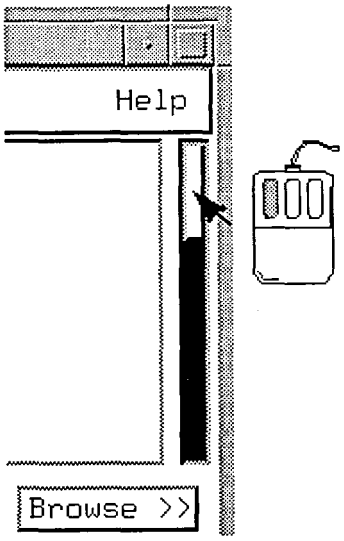
The buttons in the Help window are for navigating the online help system, as shown in the following figure.



The following list describes each button in detail:

- **Contents**—Displays the table of contents for the online help system. The table of contents provides an overview of and access to the rest of the help system.
- **Go Back**—Displays the previous topic stored in the help history. The help history contains all the topics viewed during the current CXpa session.
- **Slider**—Scrolls through the pages of the current topic. If the current topic does not consist of multiple pages, the slider is not displayed.
- **Browse**—Pages forward (>>) or backward (<<) through the current topic, one window length at a time. These buttons are deactivated if there is no next or previous page.
- **Titles/All Text**—Specifies a search of reference page titles or of the complete text of all pages.
- **Search field**—Provides a place to enter the text you wish to search for.
- **Search button**—Executes the search for the text entered in the Search field.

To scroll through the current page, point to the scroll bar with the mouse, and hold down the left mouse button while sliding the bar up and down to scroll the text. (The scroll bar does not appear if the entire page is displayed in the Help window.)



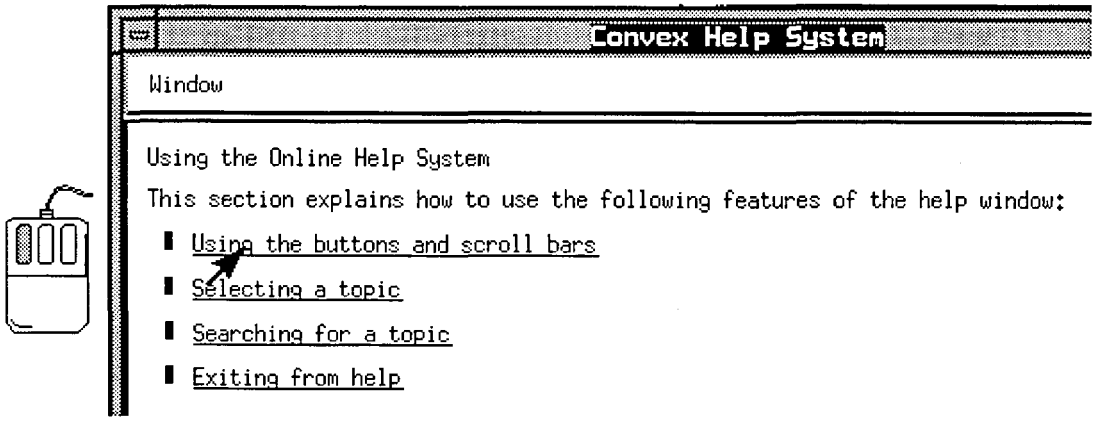
Using the menus

The two menus at the top of the window are described below:

- Window menu
  - **Print Text**—Prints an ASCII version of the current topic to your default printer using the `lpr` command. Your default printer is determined using the `PRINTER` environment variable.
  - **Close**—Closes the Help window and exits the help system.
- Help menu
  - **On Help**—Displays the "Using online help" topic.
  - **On Version**—Displays version information for the online help system.

Selecting a topic

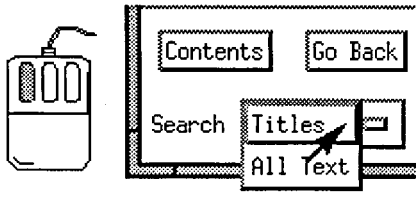
In the Help window, any underlined text is a Help topic name. Click on the underlined text with the left mouse button to view the help page for that topic.



Searching for a topic

You can search for a topic name by performing the following steps:

1. Select the type of search you want to perform. Searching the titles of the topics is generally quicker, while searching All Text is more thorough. Typically, you will want to search the titles first, and then search All Text if a Titles search proves unsuccessful.



2. Activate the Search Field by moving the cursor into the field area.
3. Type the character string that you want to search for. You can include spaces in the string, but you cannot use a regular expression.

   The topics you can search for include:
   - Report names
   - Window or dialog names
   - Topic names (for example, PDF, event, metric, line mode, and so on)
   - Message ID numbers (for example, A18)
   - CXpa commands

4. Choose Search or press **RETURN**.

The Help window displays a list of topic names that match the search string. To view one of the topics from the search list, select it with the mouse.

If only one topic name matches the search string, the Help window displays that topic directly.

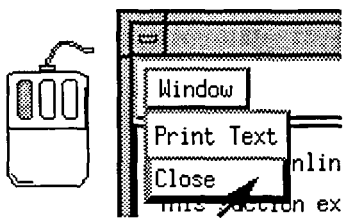
Printing a help topic

To print an ASCII text version of the current help topic in GUI mode, select Print Text from the Window menu in the Help window.

The lp or lpr command looks for the printer specified in your PRINTER environment variable. If this variable is not set, nothing is printed.

Exiting the help system

To exit the help system, select Close from the Window menu.



Choosing performance data to collect

4

This chapter explains how to configure CXpa to collect specific types of performance data (metrics) at specified regions of your program.

The information in this chapter includes:

- Introducing source code regions
- Selecting regions in GUI mode
- Selecting and deselecting regions in line mode
- Introducing metrics
- Selecting metrics in GUI mode
- Selecting metrics in line mode

Introducing source code regions

Before you run your program under CXpa, you must specify the source code regions of your program for which you want to collect profiling data (metrics). CXpa collects metrics only at the regions selected for profiling.

Depending on the options with which you compiled your source code, three types of source code regions can be selected for profiling:

- **Routines**—Routine regions are available for profiling if your source code is compiled with Exemplar compilers using the `+pa` option or if you used `cxoi` to instrument your program's object files and archive libraries.
- **Loops (all)**—Loop regions are only available for profiling if your source code contains loops and is compiled with Exemplar compilers using the `+pa` option at optimization level `+O2` or `+O3`.
- **Parallel loops**—Parallel loops are a subset of all loops that you can select for profiling. They are only available for profiling if your source code contains loops and is compiled with Exemplar compilers using the `+pa` option at optimization level `+O3 +Oparallel`.

Regions are initially selected or deselected for profiling depending on which CXpa interface you are using:

- In GUI mode, all routines in your program are initially selected.
- In line mode, no source code regions in your program are initially selected. You must first use the `select` command to select regions for profiling.

When you run your program, CXpa collects performance data at every selected source code region that is executed. You can control which regions of your program are selected for profiling using the Profile Selection dialog (in GUI mode) or the `select` and `deselect` commands (in line mode).

Selecting source code regions

CXpa provides functionality to select or deselect specific regions or types of regions for profiling. You can select or deselect:

- One type of source code region
  - In all routines
  - In specific routines
  - At specific lines (line mode only)

- All source code regions:
  - In specific routines
  - At specific lines (line mode only)
  - In all routines

You do not have to recompile your program to select or deselect regions for profiling.

For instructions on how to select regions in GUI mode, refer to the "Selecting regions in GUI mode" online help topic or section of this book.

For instructions on how to select regions in line mode, refer to the "Selecting and deselecting regions in line mode" online help topic or section of this book.

Guidelines for selecting regions to profile

Region selection should proceed from coarse-grained (all routines) to fine-grained (loops or parallel loops in specific routines) as code regions that exhibit performance problems are identified. This approach minimizes profiling intrusion (time delays that can be introduced due to the overhead of sampling hardware timers and storing profiling data while your program is running).

There are two key principles to remember:

- Minimize the number of data sampling points per program run.
- Collect only the metrics you are interested in per program run.

Use the following guidelines for selecting regions to profile:

- The first time you profile your program under CXpa, select all routines in your program for profiling at the routine region level (this is the default). This provides a complete picture of your program's performance. Using this information, you can then identify the routines that take the longest to execute.
- After you have identified the routines that consume the most wall clock time or CPU time, then select only those routines for profiling at the loop level and rerun your program under CXpa. With fewer source code regions selected, less time is spent in the timing routines CXpa uses to collect performance data.
- Profiling time increases with the number of regions selected for profiling. In general, selecting loop regions for profiling increases execution time more than selecting routine regions.
- When profiling parallel loops, select parallel loops for profiling. This provides a more accurate representation of the performance of parallel loops.

Refer to the "Profiling strategy" section of this book for more information about profiling intrusion and a step-by-step strategy for profiling.

NOTE: Selecting all source code regions in all routines for profiling increases profiling time, and significant profiling intrusion may be incurred.

Source code annotations for regions

CXpa displays special source code annotations that indicate the location of source code regions that can be selected for profiling and whether they are currently selected or deselected, as shown below.

Source code region type	Selected	Deselected
Routine	R	r
Loop	L	l
Parallel loop	P	p

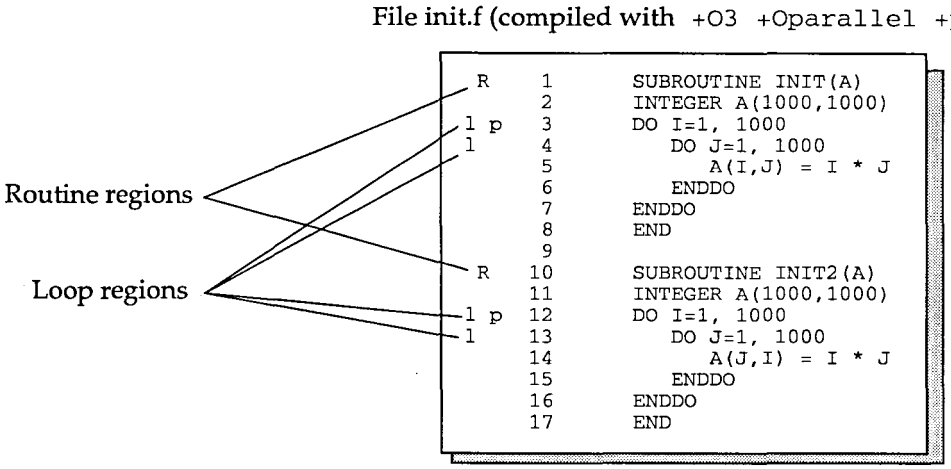
In GUI mode, you can display annotated source code by:

- Selecting Source Code from the Windows menu in any CXpa window
- Clicking on a bar of a 2D or 3D graph in the 2D or 3D Profile windows

In line mode, you can display annotated source code using the `list` or `list selectable` commands.

Introducing source code regions

The following figure shows annotated source code for sample routines with several types of selectable source code regions.



Each of the routines in the above example is currently selected for profiling. The loops beginning at lines 3, 4, 12, and 13 can be selected for profiling, but are currently deselected. The parallel loops at lines 3 and 12 can also be selected for profiling, but are currently deselected.

Related Topics

Compiling
Profiling strategy
Selecting regions in GUI mode
Selecting and deselecting regions in line mode

Related Windows

Executable Manager window
Profile Selection dialog

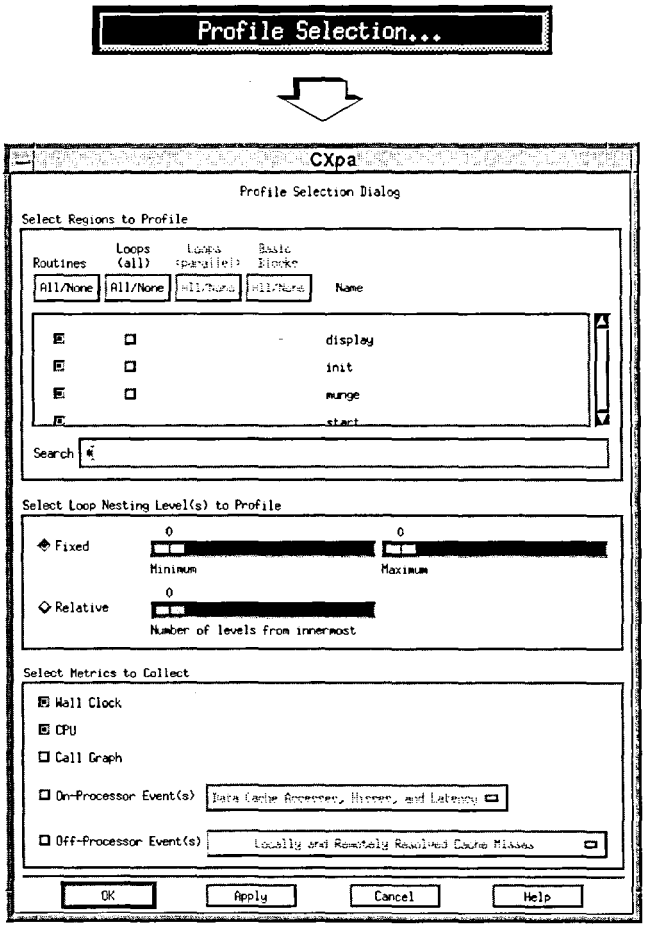
Related Commands

`deselect`
`list selectable`
`select`

Selecting regions in GUI mode

By default, all routine regions in your program that were instrumented for profiling with CXpa are initially selected for profiling in GUI mode. Use this default setting the first time you profile a program with CXpa.

To select a different set of regions for profiling, use the Profile Selection button in the Executable Manager window. The Profile Selection dialog appears.



Selecting regions in GUI mode

Using the Select Regions to Profile portion of the Profile Selection dialog, you can select or deselect the types of regions you want to profile for all routines in your program or for specific routines.

Refer to the "Profiling strategy" section of this book for more information about guidelines for selecting regions and metrics to profile.

NOTE: Selecting all source code regions in all routines for profiling is not recommended, due to increased profiling time and the amount of profiling intrusion that may be incurred.

NOTE: Profiling time increases with the number of regions and metrics selected for profiling. In general, selecting loop regions for profiling increases execution time more than selecting routine regions.

The following sections describe how to select source code regions using the Profile Selection dialog.

Selecting source code regions to profile

The upper panel of the Profile Selection dialog (Select Regions to Profile) is where you select the types of source code regions you want to profile and specify a set of routines that contain these regions to profile during a specific run of your program.

You can specify either all routines or a subset of routines. The metrics you select are only collected at these regions in the specified set of routines.

Depending on how you compiled your program, the following types of source code regions can be selected for profiling:

- Routines
- All loops (including parallel loops)
- Parallel loops only (parallel loops created by Exemplar compilers at optimization level `+O3 +Oparallel`)

Only region types that actually appear in your program can be selected. For example, if none of the routines in your program contain loops that were parallelized by the compiler, parallel loop regions are not selectable.

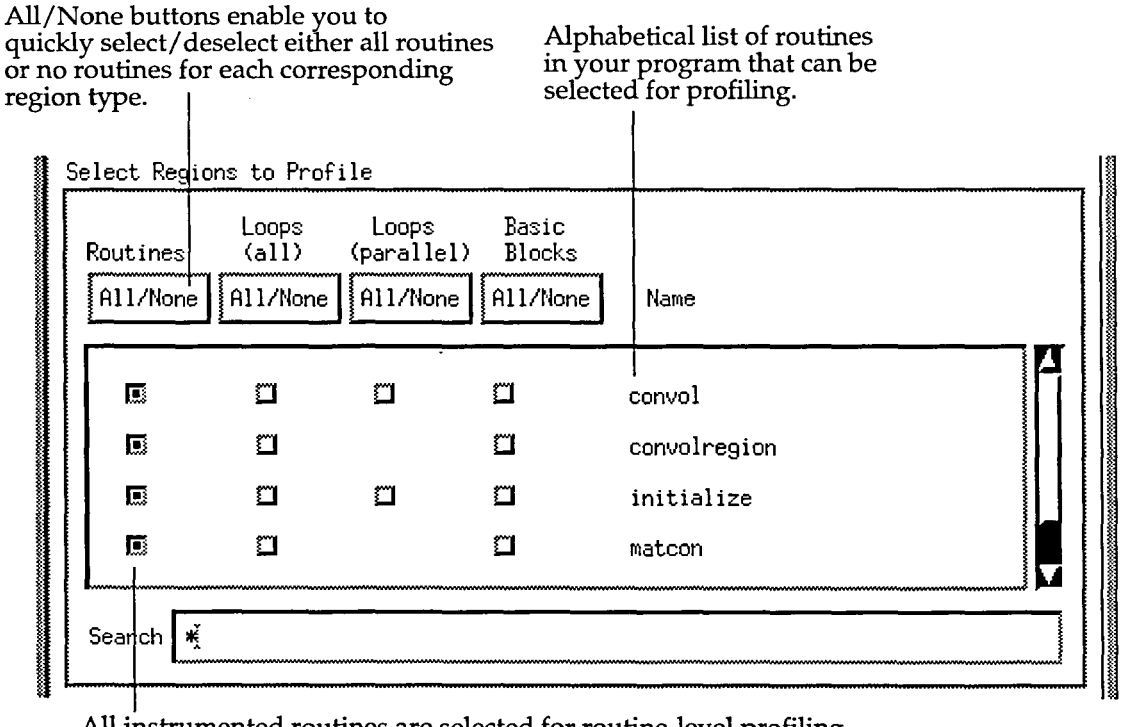
The routines in your program that can be selected for profiling are displayed in alphabetical order, and buttons are displayed opposite each routine name. If a button is not displayed for a particular region type for a routine, it means that no regions of that type can be profiled for that routine. For example, loops are not available for profiling unless the routine is compiled with an Exemplar compiler at optimization level `+O2` or `+O3` with the `+pa` option.

If your program contains a large number of routines, you can:

- Use the scrollbar to move through the routine list.
- Type the name of a routine in the Search field, then press **RETURN** to scroll the routine list so that the desired routine is displayed at the top of the list.

Default setting—Selecting all routines for routine-level profiling

The default region selection setting, which selects all available routine regions for routine-level profiling, is shown in the following figure. This is the recommended setting for the first time you run your program under CXpa. This will enable you to identify the routines that take the longest to execute. Because no loops are selected for profiling, loop nesting level settings are ignored.

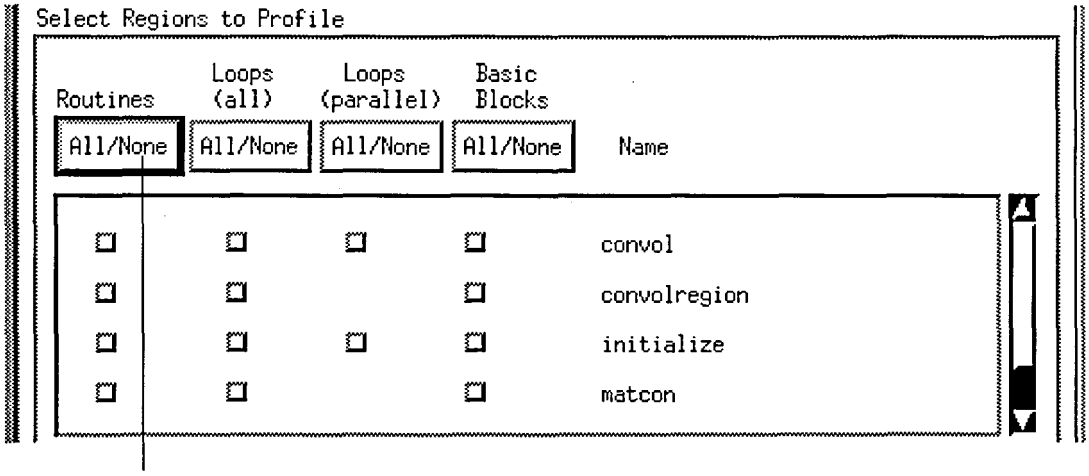


If you have changed the settings, but want to return the settings to this default, use the All/None buttons at the top of the region columns to select/deselect other types of source code regions so that routine regions are again selected for all routines.

Selecting types of source code regions in specific routines

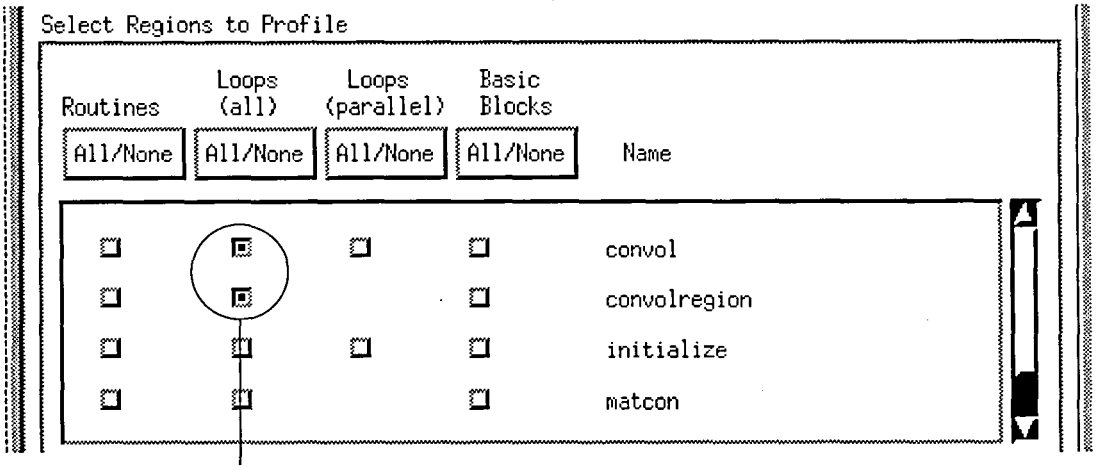
Once you have determined which routines take the longest time to execute, you can then profile loops within those routines to further isolate performance problems. To select regions in specific routines:

1. Use the All/None buttons at the top of each region column to deselect all region types and routines.



All source code regions in all routines are deselected for profiling.

2. Highlight the button corresponding to the desired region type opposite the routine you want to profile. The following figure shows how to select loops for profiling in routines `convol` and `convolregion` only.



All loops at the currently specified loop nesting level in routines `convol` and `convolregion` only are selected for profiling.

Although you can profile multiple types of source code regions for each routine, this is not recommended, because of the amount of profiling intrusion that may be introduced.

If you selected loop regions for profiling (as in the above example), the current loop nesting level setting applies to all loops. If no loops in a routine fall within the specified loop nesting level range, then no loops in that routine are selected for profiling.

The loop nesting level setting is displayed in the Select Loop Nesting Level(s) to Profile section of the Profile Selection dialog. The default loop nesting level selects only loops at nesting level 0 (outermost loops). Refer to the next section for information about selecting loop nesting levels.

Selecting loop nesting levels

If you have chosen to profile loops, you can optionally specify either a fixed range of loop nesting levels to profile or the number of loop nesting levels to profile relative to each loop nest's innermost level.

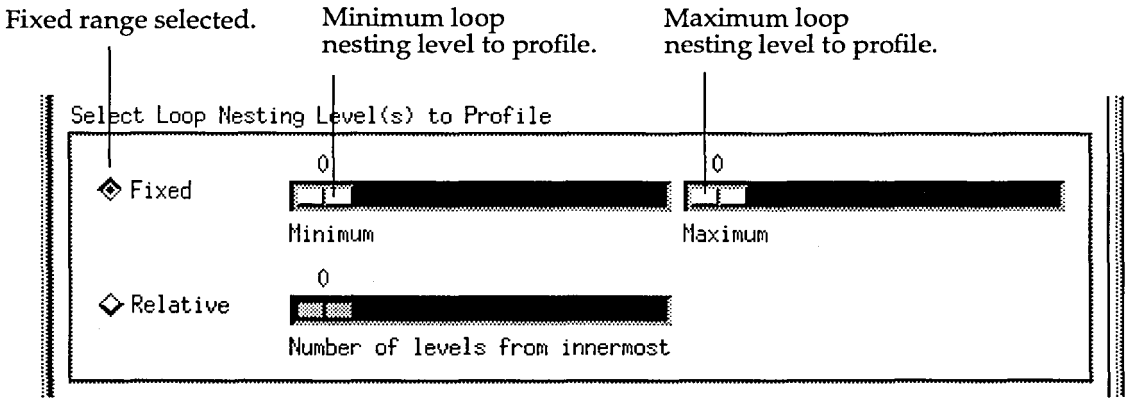
The loop nesting level setting applies to all loops selected for profiling and is only active when loop regions are selected for profiling.

CXpa automatically determines the number of loop nesting levels in your program and sets the maximum loop nesting levels and the maximum number of levels from the innermost loop appropriately. These nesting levels correspond to the loops that are created by the compiler, and may not correspond directly to your original source code due to optimizations performed.

Default loop nesting level range settings

The first time you profile loop regions in your program, use the default setting for loop nesting levels (shown in the following figure).

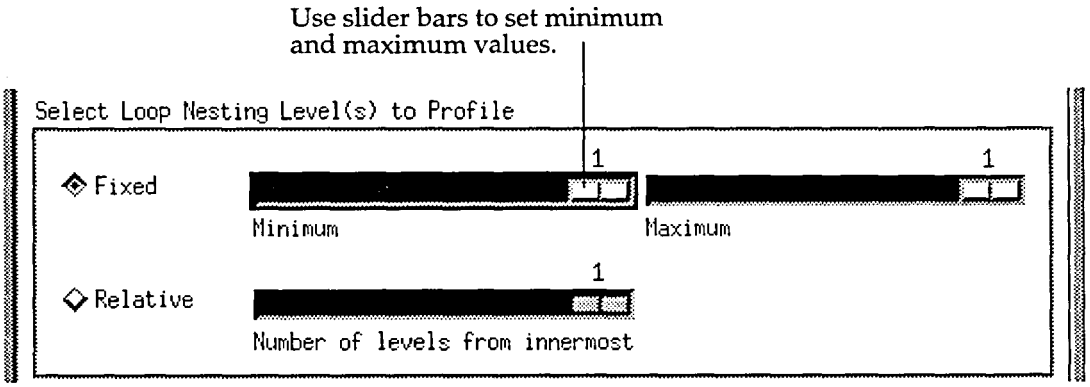
The default setting specifies a fixed loop nesting level range with a minimum of 0 and a maximum of 0. This means that all loops with a nesting level of 0 after optimization (outermost loops) are selected for profiling. This minimizes profiling intrusion.



Default loop nesting level settings—Selects loops at nesting level 0 only (outermost loops) for profiling.

Specifying a fixed loop nesting level range

On subsequent runs of your program, you can select different sections or slices of the loops within your program for profiling. When specifying a fixed range of loop nesting levels, you should generally set the minimum loop nesting level equal to the maximum loop nesting level, as shown in the following figure.



Sample fixed loop nesting level setting. Minimum and maximum nesting levels set to 1 selects only loops at nesting level 1 for profiling.

Specifying the number of relative loop nesting levels

Instead of choosing a fixed range of loop nesting levels for profiling, you can specify the number of loop nesting levels to profile relative to the innermost loop of each loop nest in your program.

- A relative setting of 0 means that only the loops at the innermost (deepest) level of each loop nest are selected for profiling.
- A relative setting of 1 means that only the loops at the two innermost nesting levels of each loop nest are selected for profiling.

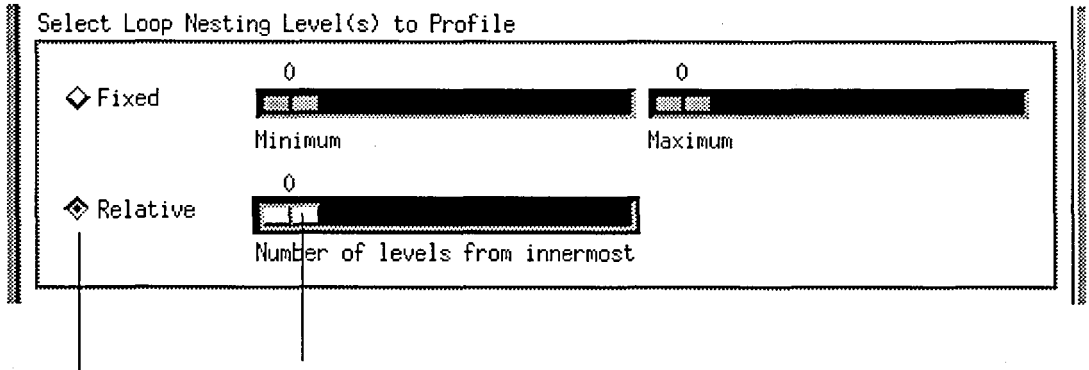
  For example, if the innermost nesting level of a loop nest is 4, and a relative setting of 1 is specified, the loops at nesting levels 3 and 4 of that loop nest are selected for profiling.
- A maximum setting (setting the slider bar as far to the right as it will go) is equivalent to selecting all loops at all loop nesting levels.

When a relative loop nesting level is specified, loops that are not part of a loop nest are also selected for profiling.

Selecting regions in GUI mode

To specify a relative setting for loop nesting levels:

1. Highlight the Relative button in the Select Loop Nesting Level(s) to Profile panel.
2. Use the slider bars to select the number of loop nesting levels to profile relative to the innermost loops in your program, as shown in the following figure:



Relative loop nesting level selected.

A relative loop nesting level setting of 0 selects all loops at the innermost nesting level of each loop nest for profiling, along with any loops that are not part of a loop nest.

Related Topics

Compiling
Introducing source code regions
Selecting metrics in GUI mode
Introducing metrics
Profiling strategy

Related Windows

Executable Manager window
Profile Selection dialog

Selecting and deselecting regions in line mode

In line mode, you can select or deselect any set of source code regions in your program with a variant of the `select` and `deselect` commands. You can:

- Select or deselect one type of source code region:
  - In all routines
  - In specific routines
  - At specific lines
- Select or deselect all source code regions:
  - In all routines
  - In specific routines
  - At specific lines

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

In line mode, no source code regions in your program are initially selected, so if you run your program without using the `select` command to select regions to profile, no metrics are collected.

NOTE: The loop nesting level setting affects the number of loops selected for profiling. The default loop nesting level setting only selects loops at nesting level 0 (outermost loops) for profiling.

Refer to the "Profile Selection dialog" or "`set visibility`" command online help topics or sections of this book for more information about loop nesting levels.

Selecting specific regions to profile provides greater control over profiling and can shorten profiling time. Refer to the "Profiling strategy" section of this book for more information about selecting regions and metrics and a step-by-step profiling strategy.

Selecting and deselecting regions in line mode

Selecting or deselecting one type of region in all routines

To select or deselect one type of source code region in all routines, use one or more of the following variants of the `select` or `deselect` commands:

Command	Description
`select routine all` `deselect routine all`	Selects or deselects all routines in your program that were instrumented for profiling.
`select loop all` `deselect loop all`	Selects or deselects all instrumented loops (including parallel loops created by Exemplar compilers) in all routines. The current loop nesting level setting is applied to the selection. Loop-level profiling is only available for routines compiled with Exemplar compilers with the `+pa` option at optimization level `+O2` or `+O3`.
`select pregion all` `deselect pregion all`	Selects or deselects all instrumented parallel loops created by Exemplar compilers in all routines. The current loop nesting level setting is applied to the selection. Parallel loops can only be profiled in routines compiled with Exemplar compilers with the `+pa` option at optimization level `+O3 +Oparallel`.

For example:

```
(CXpa) select routine all
```

The `select routine all` command selects all instrumented routines in your program for profiling. Routine and parallel loop region selections are not affected.

Use the command `select routine all` the first time you profile a program with CXpa. This identifies the routines that take the longest to execute and that may contain performance bottlenecks.

Selecting or deselecting one type of region in specific routines

To select or deselect one type of source code region in specific routines, use one or more of the following commands:

Command	Description
`select loop in` *<routine-list>*	Selects or deselects all instrumented loops in the specified routines. The current loop nesting level setting is applied to the selection. Loop-level profiling is only available for routines compiled with Exemplar compilers with the `+pa` option at optimization level `+O2` or `+O3`.
`deselect loop in` *<routine-list>*	
`select pregion in` *<routine-list>*	Selects or deselects all instrumented parallel loops in the specified routines. The current loop nesting level setting is applied to the selection. Parallel loops can only be profiled in routines compiled with Exemplar compilers with the `+pa` option at optimization level `+O3 +Oparallel`
`deselect pregion in` *<routine-list>*	

Each of these commands selects all regions of the indicated type in the specified routines. Separate multiple routines in the list with a space. If two routines have the same name, prefix them with a file name followed by a colon: *<file-name>*`:`*<routine-name>*.

For example:

```
(CXpa) select loop in INIT CALC
```

The above command selects all instrumented loops in the routines `INIT` and `CALC` that fall within the currently specified loop nesting level range. No other source code region selections are affected.

Selecting or deselecting a type of region at specific lines

To select or deselect one type of region at a specific source line (or source lines), use one or more of the following commands:

Command	Description
`select routine at` *<line-number-list>*	
`deselect routine at` *<line-number-list>*	Selects or deselects the instrumented routines at the specified line numbers.
`select loop at` *<line-number-list>*	
`deselect loop at` *<line-number-list>*	Selects or deselects the instrumented loops at the specified line numbers. Loop-level profiling is only available for routines compiled with Exemplar compilers with the `+pa` option at optimization level `+O2` or `+O3`. The current loop nesting level setting is applied to the selection.
`select pregion at` *<line-number-list>*	
`deselect pregion at` *<line-number-list>*	Selects or deselects the instrumented parallel loops at the specified line numbers.The current loop nesting level setting is applied to the selection. Parallel loops can only be profiled in routines compiled with Exemplar compilers with the `+pa` option at optimization level `+O3 +Oparallel`

The *<line-number-list>* specifies one or more line numbers that contain regions you want to select. Separate multiple line numbers in the list with a space. To select a region that is not in the current source file, prefix the line number with a file name followed by a colon: *<file-name>* : *<line-number>*. Use the `list` command to list source files and line numbers.

```
(CXpa) select loop at calc.f:3 15
```

The above command selects the instrumented loops at lines 3 and 15 of the file calc.f, assuming they fall within the currently specified loop nesting level range. No other source code region selections are affected.

Selecting and deselecting regions in line mode

Selecting or deselecting all regions in specific routines

After profiling all routines in your program, you may want to profile only the regions in specific routines whose performance you want to improve.

Use the following commands to select or deselect all regions in a routine or routines.

Command	Description
`select` *<routine-name>*	Selects or deselects all instrumented regions of any type in the specified routines. Separate multiple routine names in the list with a space. If two routines have the same name, prefix them with a file name followed by a colon: *<file-name>* : *<routine-name>*.
`deselect` *<routine-name>*	

For example:

```
(CXpa) select INIT CALC
```

The above command selects all instrumented regions in routines `INIT` and `CALC`. No other source code region selections are affected.

Selecting and deselecting regions in line mode

Selecting or deselecting all regions at specific lines

You can select or deselect all regions at a specific source line using the following command:

Command	Description
`select at` *<line-number-list>*	Selects or deselects all instrumented source code regions at the specified lines. The *<line-number-list>* specifies one or more line numbers that contain regions you want to select. Separate multiple line numbers in the list with a space. To select a region that is not in the current source file, prefix the line number with a file name followed by a colon: *<file-name>*`:`*<line-number>*. Use the `list` command to list source files and line numbers.
`deselect at` *<line-number-list>*	

For example:

```
(CXpa) select at calc.f:3 mult.f:5
```

The above command selects all instrumented source code regions at line 3 in the file calc.f and all instrumented source code regions at line 3 of the file mult.f. No other region selections are affected.

Selecting and deselecting regions in line mode

Selecting or deselecting all regions in all routines

Use the following commands to select or deselect all instrumented source code regions in your program for profiling:

Command	Description
`deselect routine all`	Deselects all instrumented regions in all routines of your program.
`select routine all`	Selects all instrumented regions in all routines of your program for profiling. This setting is not recommended. Selecting all regions in all routines in your program can significantly increase the amount of time it takes to profile large programs, especially for programs containing nested loops with large iteration counts.

Related Topics

Compiling
Introducing source code regions
Profiling strategy
Selecting metrics in line mode
Using cxoi to instrument object files and libraries

Related Commands

`deselect`
`list`
`list selectable`
`select`
`set visibility`

Selecting and deselecting regions in line mode

Introducing metrics

After selecting regions of your source code to profile, you can specify the types of performance metrics you want to collect for these regions. Collecting and comparing different metrics can help you discover performance bottlenecks such as:

- Routines and loops that consume the most wall clock or CPU time
- Loops that generate excessive cache misses
- Regions of code that spend a significant amount of their CPU time waiting on memory
- Lack of effective parallelism in a particular loop or routine
- Memory bank-contention or cache thrashing among threads in parallel regions
- Uneven distribution of work across threads in parallel regions

The first time you profile your program under CXpa, collect CPU time and wall clock time for all routines in your program. This enables you to determine which routines take the longest to execute or consume the most CPU resources.

The topics discussed in this section include:

- Metrics available on all architectures
- Event metrics:
  - Exemplar and SPP Series event terminology
  - Off-processor events (Exemplar S2000, Exemplar X2000, and SPP1600 Series)
  - On-processor events (SPP1600 Series and SPP1200 Series)

Metrics available on all architectures

The following metrics are available on all architectures:

- **CPU Time**—Time the processors work on the process.
- **Wall Clock Time**—Time to solution, including process idle time.
- **Dynamic call graph**—Wall clock time and CPU time (inclusive and exclusive), call counts, and metrics for each profiled routine, its parents, and its children.

- **CPU/Wall clock time**—The ratio of CPU time to wall clock time. This metric is computed during analysis if CPU and wall clock time metrics are collected.
  - For serial regions, if the CPU/Wall clock ratio is high (approaches 1.0), the region is compute-bound.
  - For parallel regions, this metric measures the *concurrency factor*, or the speed-up achieved through parallelization. Values that approach *n*, where *n* is the number of processors your program runs on, indicate good parallel concurrency.
  - For both parallel and serial regions, if the CPU/Wall clock ratio is low, this could indicate a performance bottleneck caused by one or more of the following:

    I/O calls—For example, read() or write() calls.

    System calls—For example, open() or close() calls.

    Memory accesses—For example, cache misses. You can compare event metrics and latency for these regions to find out if the bottleneck is due to memory accesses.

- **Execution counts**—Number of times a routine or basic block was executed or, for loops, the number of loop invocations.

- **Chunks**—For parallel loops created by the Exemplar compilers at optimization level `+O3 +Oparallel`, this indicates the number of chunks (or packets of loop iterations) that are assigned to execute on a particular thread.

  Chunk counts can be used to examine relative load balancing across threads in parallel loops. CXpa does not currently report the number of iterations in a chunk (chunk size).

Event metrics

For Exemplar and SPP Series systems, event counts and latency metrics can be collected for memory access events. Memory access events occur when required data or instructions must be retrieved from memory rather than from the processor cache. The type of event metrics that can be collected vary according to machine architecture.

Monitoring events such as cache miss counts and latency for "hot" routines on subsequent runs of your program can be useful for identifying second-order effects that contribute to poor performance such as ineffective cache utilization or contended access to data among processors on the same node.

To find bottlenecks, you can compare and contrast metrics for different events. For example, you may observe a large number of remote memory miss events at a region. However, latency metrics may reveal that even though a large number of misses are occurring in a given region, average latency time is short or total event latency time for that region is not significant.

Terms and definitions

The following definitions will help you understand and interpret the events that can be collected using CXpa on Exemplar and SPP Series systems. Terms are listed in alphabetical order.

Average clock cycles per instruction

This metric, available on SPP1200 and SPP1600 Series systems only, measures the efficiency of instruction scheduling for the region being profiled. It is computed during analysis if instruction counts and clock cycles are collected. On SPP1200 and SPP1600 Series systems, the maximum number of instructions that can be executed in a single clock cycle is 2. This means that the theoretical peak value for the average clock cycles metric on this architecture is 0.5.

If the value for average clock cycles is high, and it is associated with a frequently called routine that performs only a small amount of work (for example, a routine that simply assigns a new value to a variable and returns), inlining the routine could improve the instruction scheduling by eliminating the routine call overhead.

Code sections that contain many patterns of consecutive loads and stores followed by immediate use of requested operands can also result in high average clock cycles.

In some cases, the instruction scheduling can be improved by compiling the routine at a higher optimization level or, if programming at the assembly level, changing the order of instructions. For Exemplar systems, refer to the *Hewlett-Packard PA-RISC 2.0 Architecture and Instruction Set Reference Manual* for information on how to achieve optimal instruction scheduling; for SPP Series systems, refer to the *Hewlett-Packard PA-RISC 1.1 Architecture and Instruction Set Reference Manual* (Manual Part No. 09740-90039).

Average MIPS rate

The average MIPS (millions of instructions per second) rate is calculated during analysis if instruction counts, clock cycles, and wall clock time are collected. This metric is only available on SPP1200 and SPP1600 Series systems.

The formula CXpa uses to calculate the average MIPS rate is as follows:

$$Average\_MIPS\_rate = \frac{Number\_of\_instructions\_completed}{Wall\_clock\_time\_in\_sec}$$

The theoretical peak MIPS rate for Hewlett-Packard PA-RISC 7100/7200 processors with a 10-ns clock cycle at a clock rate of 100 MHz is 200 MIPS. There is a direct correlation between average clock cycles and average MIPS rates; decreasing average clock cycles increases average MIPS rates.

CTI (Coherent Toroidal Interconnect) rings

The ring interconnect that connects all the hypernodes of a multihypernode Exemplar or SPP Series system in a ring topology. While the CTI ring is derived from the IEEE SCI (Scalable Coherent Interface) standard, complete compatibility is sacrificed to provide lower latencies.

Data cache miss

A data cache miss occurs if data to be loaded does not reside in the processor's data cache.

Data cache hit

A data cache hit occurs if data to be loaded resides in the processor's data cache.

Data cache hit rate

This metric, available on SPP1200 and SPP1600 Series systems only, measures the percentage of total data cache accesses that were data cache hits. If the data cache hit rate is low, this indicates cache thrashing.

The data cache hit rate is calculated as follows:

$$data\_cache\_hit\_rate = \frac{total\_data\_cache\_accesses - data\_cache\_misses}{total\_data\_cache\_accesses} \times 100$$

Data TLB misses

Data translation lookaside buffer (TLB) misses. This metric represents the number of times the address translation from virtual to physical memory for data to be referenced was not found in the TLB. The TLB is a cache of virtual-to-physical memory address translations for the most recently referenced page table entries. On SPP1600 Series systems, the TLB contains 120 entries; on Exemplar S2000/X2000 systems, it contains 92 entries.

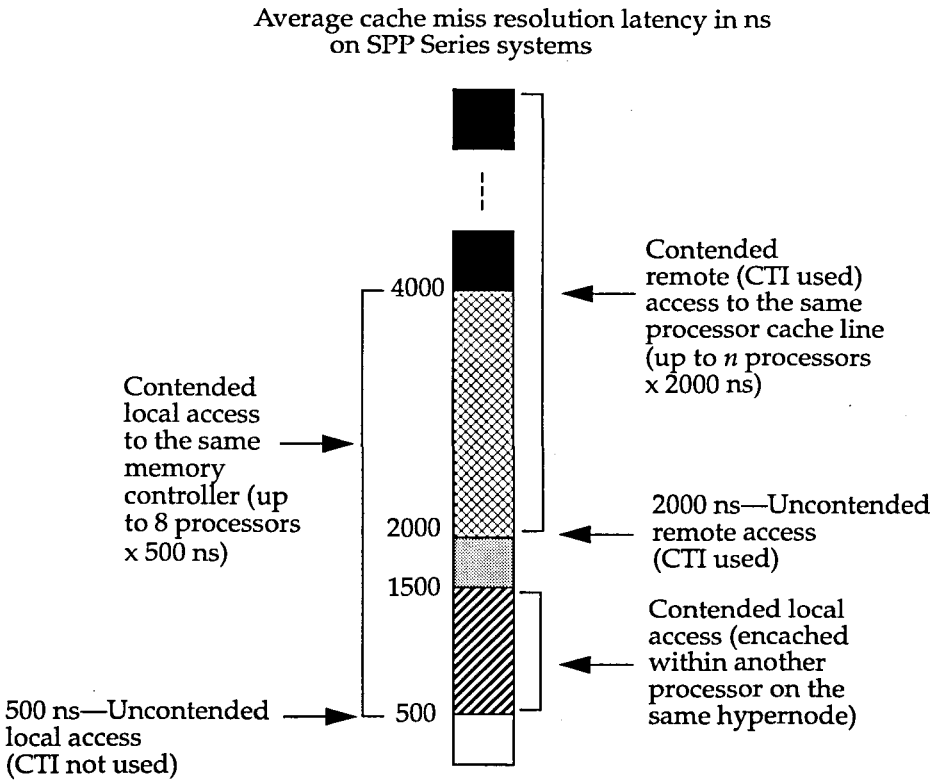
Event counts

The number of times an event, such as a data or instruction cache miss, occurred.

Event latency/counts

Average latency time per event for a region. For example, if total data cache miss counts are collected, then the event latency/counts metric represents the average amount of time it took to resolve a data cache miss.

Average cache miss resolution latency metrics can be useful when examining regions with large numbers of cache misses or latency and can be characterized on SPP Series machines as shown in the following figure:



You can use this information to interpret 2D and 3D profile graphs of average cache miss resolution.

Event latency/CPU

Ratio of event latency to CPU time for a region, expressed as a percentage. When this percentage is high, it means that the program spent the majority of its CPU time in the region reaching for data or instructions that were not encached rather than computing with encached data or instructions.

Hypernode

In Exemplar and SPP Series systems, a set of up to 16 processors and physical memory organized as a symmetric multiprocessor (SMP) running a single image of the operating system microkernel. An SPP system consists of one or more hypernodes, with a high speed CTI ring connecting the hypernodes.

Instruction cache miss

An instruction cache miss occurs if data to be loaded does not reside in the processor's instruction cache.

Instruction TLB misses

Instruction translation lookaside buffer (TLB) misses. This metric represents the number of times the address translation from virtual to physical memory for an instruction was not found in the TLB. The TLB is a cache of 120 virtual-to-physical memory address translations for the most recently referenced page table entries.

Latency

As reported by CXpa, the amount of time spent accessing memory to locate data or instructions not found in the processor's data or instruction cache.

Locally resolved cache misses

Cache misses that are resolved by using the hypernode crossbar to access memory found on the same hypernode as the processor; the CTI rings are not used.

Read miss

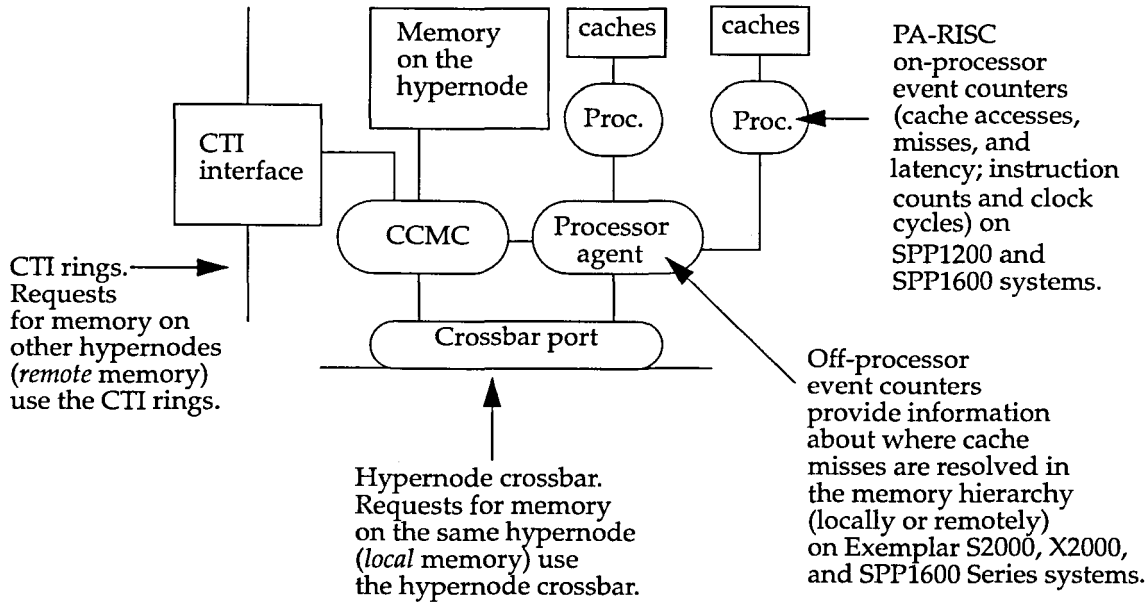
An instruction or data cache miss caused by a load.

On-processor and off-processor events

On-processor events are monitored by event counters located on the Hewlett-Packard PA-RISC processors used in SPP1200 and SPP1600 systems. These event counters monitor events from the processor's point of view only.

Off-processor events are monitored by event counters on the processor agent chip. These event counters can be configured to monitor the number of data cache miss events that are resolved locally and/or remotely (remote miss events imply use of the CTI rings; local miss events do not use the CTI rings). On SPP1600 Series systems, these counters can also be configured to track whether those events occurred due to reads (loads) or writes (stores).

The following diagram shows the location of event counters on Exemplar SPP Series systems. Refer to *Exemplar S-Class and X-Class Architecture*, *Exemplar SPP1000-Series Architecture*, or the *Exemplar Programming Guide* for more information.



Processor agent chip

The processor agent chip is the gate array on Exemplar and SPP Series systems that provides a high-speed interface between the pairs of PA-RISC processors in a functional block on a hypernode and the hypernode crossbar.

Performance counters (referred to as *off-processor* counters) located on the processor agent chip used in Exemplar S2000, Exemplar X2000, and SPP1600 Series systems provide information about locally and remotely resolved data cache misses and latency time.

Remotely resolved cache misses

Cache misses that were resolved by using the CTI rings to access memory on another hypernode. Accessing memory on other hypernodes using the CTI rings takes significantly longer than accessing memory on the same hypernode via the hypernode crossbar.

Write miss

An instruction or data cache miss caused by a store or by a load and clear.

Off-processor events (Exemplar S2000, Exemplar X2000, and SPP1600 Series systems)

On Exemplar S2000, Exemplar X2000, and SPP1600 Series systems, the following off-processor events can be collected:

Locally resolved data cache misses and latency

> Configures off-processor event counters to collect the number of times that data not found in the processor cache was found in local memory (memory allocated to that processor's hypernode). Data cache miss latency is also collected.

Remotely resolved data cache misses and latency

> Configures off-processor event counters to collect the number of times that data not found in the processor cache was found in remote memory (memory allocated to another hypernode). Data cache miss latency is also collected.

NOTE: Collecting remote miss events on single-node Exemplar or SPP Series systems yields zeros.

Locally and remotely resolved data cache misses and latency

> Collects the number of local and remote memory misses and latency, combined.

On SPP1600 Series systems, you can specify whether the above events are collected during read accesses (loads), write accesses (loads), or both. The default is both.

On-processor events (SPP1200 and SPP1600 Series systems)

On SPP1200 and SPP1600 Series systems, the following on-processor events can be collected:

Data cache access counts, miss counts, and latency

Configures on-processor event counters to collect the total number of data cache accesses, data cache misses, and cache miss latency. Average latency and data cache hit rates are computed during analysis.

Instruction cache miss counts and latency

Configures on-processor event counters to collect the total number of instruction cache misses and latency. Average instruction cache miss latency is computed during analysis.

Instruction counts and clock cycles

Configures the on-processor event counters to collect the number of completed instructions and clock cycles. The average number of clock cycles per instruction is computed during analysis. The average MIPS rate is computed during analysis if wall clock time is also collected.

Data and instruction TLB misses

Configures on-processor event counters to collect data and instruction TLB misses. The TLB is a cache of virtual-to-physical memory address translations for the most recently referenced page table entries. This metric represents the number of times the virtual-to-physical memory address translations for data or instructions to be referenced were not found in the TLB.

Related Topics

Dynamic Call Graph report
Loop reports
Profiling strategy
Routine reports
Selecting metrics in GUI mode
Glossary
Parallel Region reports
Reports
Selecting metrics in line mode

Related Windows

2D Profile window
3D Profile window
Analysis Report window
Call Graph window
Executable Manager window
Profile Selection dialog

Related Commands

`analyze`
`collect`
`select`
`set events`

Introducing metrics

Selecting metrics in GUI mode

Once you have selected the source code regions in your program you want to profile, select the types of metrics you want to collect at those regions when you run your program. By default, CXpa collects CPU time, wall clock time, and execution counts for the regions in your program you have selected for profiling.

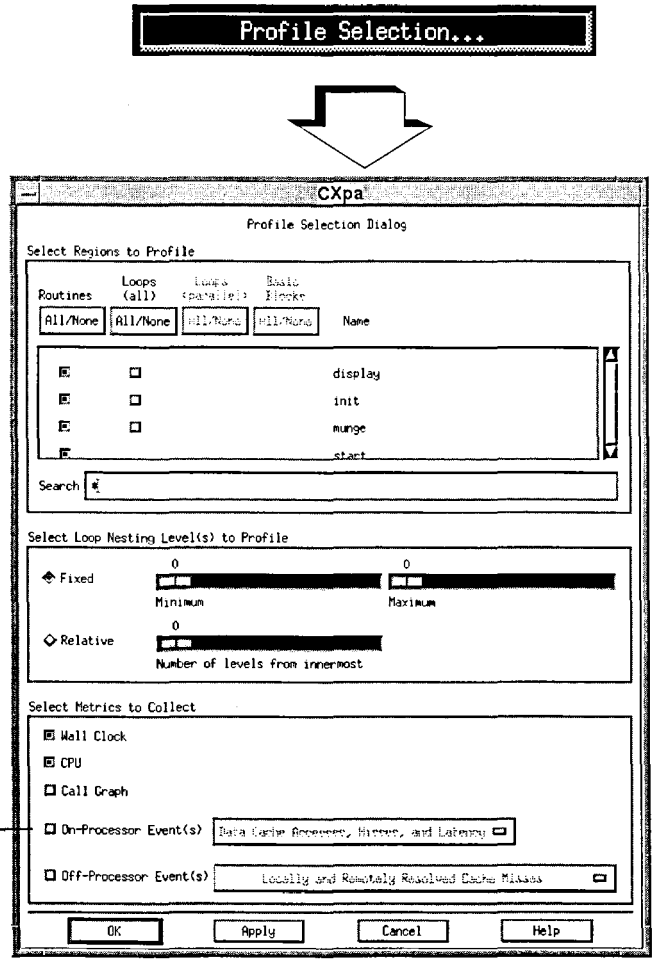
CXpa collects these metrics at the regions of your program that are selected for profiling. You can choose to collect:

- Wall clock time (default)
- CPU time (default)
- Events
- Dynamic call graph

This section describes the procedure for choosing metrics to collect using CXpa's X/Motif graphical user interface.

To specify the type of metrics to collect during profiling, perform the following steps:

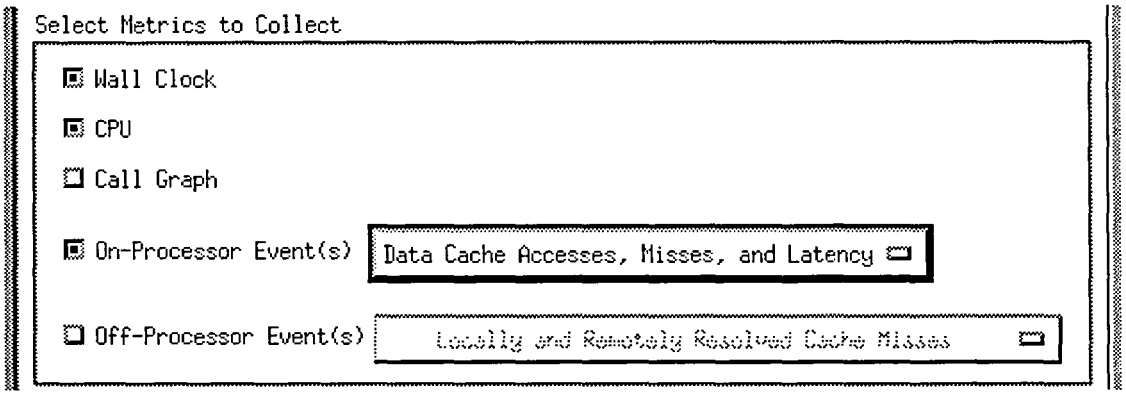
1. Use the Profile Selection button in the Executable Manager window to bring up the Profile Selection dialog.



By default, CXpa collects CPU time and wall clock time for all routines.

On/Off-processor event selection menus vary according to architecture and are desensitized until event collection is enabled.

2. Select the source code regions of your program you want to profile as described in "Selecting regions in GUI mode."
3. Specify the metrics you want to collect using the buttons in the Select Metrics to Collect section of the Profile Selection dialog.

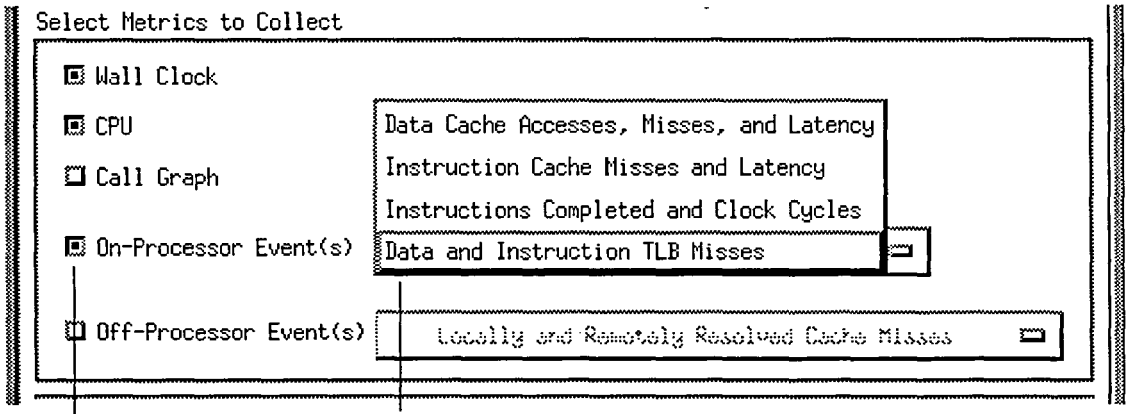


If you want to collect Events, you must specify the type of event or events you want to collect as described in steps 4 and 5. Otherwise, continue with step 6.

4. Select On-Processor or Off-Processor Events in the Select Metrics to Collect section of the Profile Selection dialog to enable event collection.

   Available On/Off-Processor Event(s) menus and menu selections vary according to architecture and are desensitized until event collection is enabled.

5. Select an event type or types from the On-Processor or Off-Processor Event(s) option menus, as shown in the following figure.



Event collection is enabled.

Click on Event(s) option menus to display a list of hardware events that can be collected on your system. These vary according to machine architecture.

Selecting metrics in GUI mode

The number and type of events you can select vary according to machine architecture.

Refer to the "Introducing metrics" online help topic or section in this book for a discussion of available event metrics for Exemplar S2000, Exemplar X2000, SPP1200 Series, and SPP1600 Series systems.

6. Choose OK.

Related Topics

Introducing metrics
Profiling strategy
Selecting regions in GUI mode

Related Windows

Executable Manager window
Profile Selection dialog

Selecting metrics in line mode

Once you have selected the regions in your program you want to profile, you must specify the types of metrics you want to collect at those regions.

In line mode, you specify the types of metrics you want to collect using the `collect` command and, if you have chosen to collect events, use the `set events` command.

CPU time, wall clock time, dynamic call graph, and execution count metrics can be collected on all architectures. Other metrics that you can collect (including events and latency time) differ according to machine architecture.

This section describes the procedure for selecting types of metrics to collect when running CXpa in line mode. Refer to the following online help topics or sections of this book for more information:

- "Introducing metrics"—List of available metrics on each architecture, metric descriptions, and event terminology.
- Reference page for the `collect` command—Complete syntax and examples.
- Reference page for the `set events` command—Complete syntax and examples.

Choosing metrics to collect at the command line

Perform the following steps to select the types of metrics you want to collect:

1. Select the regions in your program that you want to profile using a form of the `select` command. For example:

   ```
   (CXpa) select routine all
   ```

   The above command tells CXpa to select all routine regions in your program for profiling.

NOTE: If you do not select one or more source code regions in your program for profiling with the `select` command, CXpa will not collect any metrics.

2. Enter a form of the `collect` command at the CXpa prompt. For example:

   ```
   (CXpa) collect cpu wall_clock call_graph events
   ```

The above command tells CXpa that you want to collect CPU time, wall clock time, dynamic call graph information, and events at the regions of your program selected for profiling. Refer to the reference page for the `collect` command for more information and a list of valid parameters.

If you chose to collect events, you must use the `set events` command to specify the type of events you want to collect.

3. If you have specified event collection with the `events` parameter of the `collect` command, enter the command `set events` *<event-type>* at the CXpa prompt where *<event-type>* specifies the type of event you want to collect.

   The type and number of events you can collect differ according to machine architecture. Refer to the next section, "Event types," for a list of valid event types and parameters for specifying events for Exemplar and SPP Series architectures. For example:

   ```
   (CXpa) set events local_misses
   ```

   The above command tells CXpa that you want to count the number of times that data missed in the processor cache was found in memory on that processor's hypernode (for the profiled regions of your program). By default, events are collected during read and write operations. The `local_misses` parameter is valid for Exemplar S2000, Exemplar X2000, and SPP1600 Series systems.

NOTE: Each use of the `set events` command overwrites its previous setting.

4. Enter the `run` command at the CXpa prompt to collect the metrics you have specified for the selected regions of your program:

   ```
   (CXpa) run
   ```

Event types

The type and number of events that can be collected per program run differ according to machine architecture:

- On Exemplar S2000 and X2000 systems, you can monitor one type of off-processor event per program run.
- On SPP1200 Series systems, you can monitor one set of on-processor events per program run.
- On SPP1600 Series systems, you can monitor one type of off-processor event and one set of on-processor events per program run.

Valid parameters to the `set events` command for specifying events on each architecture are listed in the following sections.

Off-processor events (Exemplar S2000, Exemplar X2000, and SPP1600 Series systems)

On Exemplar S2000, Exemplar X2000, and SPP1600 Series systems, use one of the following parameters of the `set events` command to select the type of off-processor event you want to collect. You can select only one of the following to collect per program run:

- **`local_misses`**—Number of times that the profiled regions of your program had to access local memory (memory on that processor's hypernode) because of a processor data cache miss.
- **`remote_misses`**—Number of times the profiled regions of your program had to access remote memory (memory on another hypernode) because of a processor data cache miss. This implies use of the CTI (Coherent Toroidal Interconnect) rings. This event is only meaningful for multihypernode systems; on a single-hypernode system, collecting remote miss events yields zeroes.
- **`local_and_remote_misses`**—Number of locally and remotely resolved data cache misses combined for the profiled regions of your program.

On SPP1600 Series systems, you can use one or both of the following parameters of the `set events` command to specify whether the type of off-processor event you have selected is collected during read operations only, write operations only, or both:

- **`read_only`**—Collects cache misses that occur during read operations. A read miss is a data cache miss caused by a load.
- **`write_only`**—Collects cache misses that occur during write operations. A write miss is a data cache miss caused by a store or a data cache miss caused by a load and clear.

The default is `read_only write_only` (both).

On-processor events (SPP1200 and SPP1600 Series systems)

SPP1200 and SPP1600 Series systems use HP PA-RISC 7200 processors that have on-processor event counters. Valid parameters to the `set events` command for specifying on-processor event types are listed below. You can select only one of the following to collect per program run:

- **`data_cache`**—Configures on-processor event counters to collect data cache access counts, miss counts, and latency.
- **`instruction_cache`**—Configures on-processor event counters to collect instruction cache miss counts and latency.
- **`instruction_counts`**—Configures on-processor event counters to collect completed instruction counts and clock cycles.
- **`tlb_misses`**—Configures on-processor event counters to collect data and instruction TLB (translation lookaside buffer) misses.

Related Topics

Introducing metrics
Profiling strategy
Selecting and deselecting regions in line mode

Related Commands

`collect` `select`
`set events`

Part 2 Reference pages

Part 2 of this book contains reference pages for:

- CXpa's performance reports
- The windows in CXpa's X/Motif graphical user interface
- The commands for CXpa's line mode interface
- CXpa messages

You can also access this information through the CXpa online help system.

Reports

5

This chapter covers the following topics:

- Overview of the text reports that CXpa displays for the profiled regions of your program
- Instructions for creating and viewing reports
- Report filtering options
- A description of the information contained in the CXpa report header
- Detailed examples of each type of report
- Descriptions of all report fields

Report types

CXpa can display textual performance reports for the following types of source code regions:

- Routines
- All loops
- Parallel loops only

The metrics available in performance reports vary according to machine type, source code regions selected for profiling, and the options used when compiling your program. Performance reports that are available for profiled regions are listed below:

- **Call Counts report**—Execution counts for routines.
- **Computation report**—CPU time and % total CPU time.
- **Time to solution report**—Wall clock time and CPU/wall clock time.
- **Dynamic Call Graph report** —For each routine, the information displayed includes inclusive wall clock and CPU time (including time spent in child routines) and call counts for the routine, its parents, and its children.
- **Events**—Available reports and metrics differ according to

machine architecture and the type of event collected during the run of the program that generated the performance data (PDF) file:

- Off-processor event reports (Exemplar S2000, Exemplar X2000, and SPP1600 Series systems only)—Locally resolved cache misses; remotely resolved cache misses; locally and remotely resolved cache misses.
- On-processor event reports (SPP1200 and SPP1600 Series systems only)—Data cache misses and latency; data cache accesses; instruction cache misses and latency; data and instruction TLB misses; instructions completed and clock cycles.

For a detailed discussion of each type of report, refer to the "Routine reports," "Loop reports," "Parallel Region reports," and "Dynamic Call Graph report" online help topics or sections in this book.

For a list of fields, abbreviations, and annotations that can appear in CXpa reports, refer to the "Report fields" online help topic or section of this book.

Displaying report data for individual threads

This section describes how you can filter report data to display information for either processes (the default) or individual threads.

Line mode

In line mode, you can filter report data with the `set visibility` command:

- `set visibility thread`—Display performance data on a per-thread basis for all reports.
- `set visibility process`—Display performance data on a per-process basis for all reports. This is the default.

To view the current CXpa thread/process visibility settings in line mode, use the `info` command.

GUI mode

To filter report data in GUI mode:

1. Open an Analysis Report window.
2. Select Filter Report from the Options menu in the Analysis Report window. CXpa opens a Filter Report dialog.

3. Select the level of detail to be displayed in CXpa reports.
   - Choose Thread to display performance data on a per-thread basis for all reports.
   - Choose Process to display performance data on a per-process basis for all reports. This is the default.
4. Choose OK.

Viewing reports

Textual performance reports are available in GUI mode and line mode.

GUI mode

In GUI mode, you can select the types of reports you want to view from the Analysis Report window. You can create an Analysis Report window using one of the following methods:

- Select Report from the option menu at the lower right corner of the Executable Manager window.
- Use the Create Report button in the Analysis Control window.
- Select Report from the Windows menu in the Executable Manager window or the Analysis Control window.

Once you have created an Analysis Report window, select the type of source code region for which you want to create reports, then select and/or deselect appropriate metrics.

You can also create customized reports by specifying a subset of routines that contain regions of the currently selected type by selecting Subset from the Select Regions option menu in the Analysis Report window. Refer to the "Region Subset Selection dialog" online help topic or section of this book for more information.

Line mode

In line mode, use a form of the `analyze` command to display performance reports (for example, `analyze`, `analyze routine`, `analyze loop`, `analyze call_graph`, or `analyze pregion`).

CXpa displays reports in line mode using the pager specified with your `PAGER` environment variable. You can also redirect output from this command to a file.

Report header

When you generate a report, CXpa prefaces the report with a header that contains the following information:

- **Executable**—Executable name for the program being profiled.
- **Profile Data**—Performance data file (PDF) name.
- **Process State**—State of the process when the PDF was created or closed. The states include: `running`, `paused`, `terminated`, `exited`, and `not started`.
- **Metric Totals**—Totals for metrics collected across all threads of the process (for example, total wall clock time or CPU time).
- **Architecture**—Architecture where the PDF was created and related system information.

Related Topics

Dynamic Call Graph report
Introducing metrics
Loop reports
Parallel Region reports
Report fields
Routine reports

Related Windows

Analysis Report window
Call Graph window
Filter Report dialog
Region Subset Selection dialog

Dynamic Call Graph report

Description

For each profiled routine in your program, the Dynamic Call Graph displays inclusive or exclusive wall clock and CPU time and call counts for the routine, its parents, and its children.

To collect the profiling data required to generate a Dynamic Call Graph report:

- Prepare all routines in the program for routine-level profiling by compiling them with the `+pa` option of the Exemplar compilers and/or using the `/opt/cxpa/bin/cxoi` utility to instrument the object files and archive libraries.
- Make sure all routines are selected for profiling in the Profile Selection dialog (in GUI mode) or enter the following command (in line mode):

  `(CXpa) select routine all`

- Enable CPU time, wall clock time, and Call Graph metric collection in the Profile Selection dialog (in GUI mode) or specify the `call_graph`, `cpu`, and `wall_clock` parameters of the `collect` command (in line mode). For example:

  (CXpa) `collect cpu wall_clock call_graph`

NOTE: When generating a Dynamic Call Graph report, make sure that all routines in your program are instrumented and selected for profiling. Otherwise, the results displayed in the call graph report may be misleading.

For example, if an instrumented routine named `parent()` calls an uninstrumented routine `child()`, and `child()` then calls an instrumented routine `sub1()`, the call graph would incorrectly show that `parent()` called `sub1()`, and all of the time spent in the uninstrumented routine `child()` would be attributed to `sub1()`.

Report

The fields and columns in the Dynamic Call graph report display the following information:

- `Wall (with children)`—Inclusive wall clock time (includes time spent in called routines).
- `CPU (with children)`—Inclusive CPU time (includes time spent in called routines).

- `------`(and so on)—Dashed lines represent report section boundaries. The sections are displayed in order, from highest to lowest, ranked by inclusive wall clock time for the section's primary routine.
- `>`—Indicates the primary routine in each section.
  - The routines listed above the primary routine in each section are the callers of that routine (that is, that routine's parents).
  - The routines listed below the primary routine in each section were called by that routine (that is, that routine's callees, or children).
- `Calls in`—For callers of the primary routine in each section, the number of times the primary routine was called; for the primary routine in each section, the total number of calls made to that routine.
- `Calls out`—For callees of the primary routine in each section, the number of times the routine was called by the primary routine; for the primary routine, the total number of times that it was called.
- `PS`—Profiling status. A period (.) in this field indicates that the time displayed does not reflect a measured time, but rather a `gprof`-style "best-guess" time inferred from available profiling data
- `Routine names`—Names of all profiled routines in your program.

NOTE: All CXpa reports express time in seconds unless the time is annotated with the letter "m" for milliseconds.

Refer to the "Report fields" online help topic or section of this book for information about fields, column headings, abbreviations, and annotations that can appear in Call Graph reports.

Refer to the "Call Graph window" online help topic or section of this book for information about viewing call graph information in GUI mode using CXpa's interactive Call Graph window.

The following sample report shows the first two sections from a Dynamic Call Graph report:

```
================================================================================
                        Routine Performance Analysis
================================================================================
                                Dynamic Call Graph
  Wall (with   CPU (with    calls      calls       PS  routine
  children)    children)      in         out          names
--------------------------------------------------------------------------------
>     10.144       6.858          1        12    start
       5.338       4.996                    3       kernel
       2.542       0.440                    4       report
       2.247       1.421                    3       tick
       1.417m      0.208m                   2       second
--------------------------------------------------------------------------------
       5.338       4.996          3                 start
>      5.338       5.022          3        81    kernel
       2.580       1.968                   75       test
       0.721m      0.720m                   3       result
       0.027m      0.027m                   3       space
--------------------------------------------------------------------------------
```

... lines of output omitted.

The primary routine in the first section of the dynamic call graph shown in the above example is `start`, which is the main routine of the program. As shown in the `Calls out of` column, `start` made a total of 12 calls to routines `kernel` (3 calls), `tick` (3 calls), `report` (4 calls), and `second` (2 calls). As shown in the `Wall (with children)` and `CPU (with children)` columns, the largest amount of CPU and wall clock time spent in routines called by `start` is attributed to the `kernel` routine.

To display dynamic call graph information for individual threads in call graph reports:

- In line mode, execute the following commands:
  - `(CXpa)` **`set visibility threads`**
  - `(CXpa)` **`analyze call_graph`**
- In GUI mode:
  - Select Filter Report from the Options menu in the Analysis Report window. CXpa opens a Filter Report dialog.
  - Select Threads, then choose Apply.

Dynamic Call Graph report

Context

To display a Call Graph report in GUI mode, open the Analysis Report window and choose Routines as the Region Type and Call Graph from the Metrics list.

To display a dynamic call graph report in line mode, use the command `analyze call_graph`.

Related Topics

Introducing metrics
Loop reports
Parallel Region reports
Report fields
Reports

Related Windows

Analysis Report window
Call Graph window
Filter Report dialog
Region Subset Selection dialog

Related Commands

`analyze`
`collect`
`select`
`set visibility`

Loop reports

Description

Loop performance analysis reports display metrics for all profiled loop regions in your program.

Loop reports are only available if:

- The routines containing the loops you wish to profile are compiled with an Exemplar compiler using the `+pa` option at optimization level `+O2` or `+O3`. If you selected parallel loops only for profiling, the routines must be compiled at optimization level `+O3 +Oparallel`.
- Your program contains loops, and they were selected for profiling.
- At least one profiled loop region was executed.

NOTE: The loop nesting level setting affects the number of loops selected for profiling. The default loop nesting level setting (a fixed loop nesting level range with a minimum of 0 and a maximum of 0) only selects loops at nesting level 0 (outermost loops) for profiling.

Refer to the "Profile Selection dialog" or "`set visibility`" command online help topics or sections of this book for more information about loop nesting level settings.

Depending on the architecture on which you created the PDF and the type of metrics collected, the following reports can be displayed for each routine containing profiled loop regions that were executed:

- **Computation**—CPU time, including and excluding time spent in inner loops.
- **Time to solution**—Wall clock time and CPU/wall clock time.
- **Events**—Available event reports and metrics differ according to machine architecture and the type of event collected during the run of the program that generated the PDF file:
  - Off-processor event reports (Exemplar S2000, Exemplar X2000, and SPP1600 Series systems only)—Locally resolved cache misses, remotely resolved cache misses, or locally and remotely resolved cache misses.
  - On-processor event reports (SPP1200 and SPP1600 Series systems only)—Data cache misses and latency; data cache accesses; instruction cache misses and latency; instructions completed and clock cycles; and data and instruction TLB (translation lookaside buffer) misses.

All loop reports display the following information:

- Line number corresponding to the source code for the loop—Line numbers for optimized loops annotated with a lowercase letter indicate that the loop was split into two or more loops during optimization.
- Number of times the loop was executed.
- Resulting nesting level of the loop after optimization.
- The history of the transformations applied by the compiler (shown in the Optimization column) for optimized loops. This provides the most accurate picture of loop performance for your program.

  Refer to the "Report fields" online help topic or section of this book for a listing of the abbreviations shown in the Optimization column and their meaning.

  For a complete discussion of automatic optimizations performed by Exemplar compilers and manual optimization techniques for Exemplar and SPP Series systems, refer to the *Exemplar Programming Guide.*
- Inclusive metric values—Includes values for inner loops (plus inner).
- Exclusive metric values—Does not include values for inner loops (less inner).

For parallel loops, the data in loop performance analysis reports is summed across all threads of the loop. To view information for individual threads in parallel loops, select the Loops (parallel only) option on the Region Type selection panel in the Analysis Report window or use the `analyze pregion` command to create parallel region reports.

Refer to the "Report fields" section online help topic or section of this book for information about fields (columns), abbreviations, and annotations that can appear in loop reports.

Refer to the "Reports" or "`set visibility`" online help topics or sections of this book for information about filtering options for reports.

NOTE: All CXpa reports express time in seconds unless the time is annotated with the letter "m" for milliseconds.

Reports

For loop regions, three types of reports are available: Computation, Time to Solution, and Events. These are described in the following sections, along with sample reports.

Computation

Loop computation reports are displayed for each routine containing profiled loop regions that were executed. The metrics in the Computation report for loops can be used to examine the amount of CPU time spent executing each loop:

- Excluding time spent in inner loops
- Including time spent in inner loops

A sample loop Computation report is shown in the following example:

```
================================================================================
                          Loop Performance Analysis
                                  For: init
================================================================================

Optimized Loops:
                                        Computation
                          Times           (less inner)          (plus inner)
 Line     NL Optimization  Exec          CPU Time              CPU Time             PS
-------  -- ----------- --------  ----------            ----------             --
   128    0              1000000     219.794               356.863
   129a   1 pU:5         8980000      68.728                68.728
   129b   1              8962040      68.340                68.340
```

Time to solution

The metrics in the Time to Solution report for loop regions can be used to examine the following (assuming you have collected wall clock time):

- Total wall clock time spent in loops.
- Ratio of CPU time to wall clock time (concurrency factor).

For all loops, if the CPU/wall clock ratio is low, it indicates that there may be some type of performance bottleneck caused by one or more of the following:

- I/O calls (for example, read() or write() calls).
- System calls (for example, open() or close() calls).
- Memory accesses (for example, cache misses). You can compare event metrics and latency for these loops to find out if the bottleneck is due to memory accesses.

For serial loops, if the CPU/wall clock ratio is high (approaches 1.0), the region is compute-bound.

For parallel loops, the CPU/wall clock ratio is the concurrency factor for the parallel loop. (Parallel loops are annotated with a "P" in the Optimization column.) Values for CPU/wall clock time that approach *n*, where *n* is the number of processors used by your program, indicate good parallel concurrency.

For example, as you increase the number of processors or the amount of work (data set size), the concurrency factor should increase proportionately. This would indicate that the parallel loop region is scaling well in parallel.

Time to Solution reports for loops are displayed for each routine containing profiled loop regions that were executed. The following example shows a Time to Solution report. The high CPU/Wall ratio for the loops in this region (7.76 to 7.64) indicates near peak parallel concurrency This program was run on an SPP1200 subcomplex with 8 processors.

```
================================================================================
                          Loop Performance Analysis
                              For: convolregion
================================================================================
Optimized Loops:
                                        Time to Solution
                              Times       (less inner)          (plus inner)
 Line     NL Optimization     Exec     Wall Clock CPU/Wall  Wall Clock CPU/Wall  PS
-------  -- -----------  --------  ---------- --------  ---------- --------  --
   128    0              1000000      16.370     1.00       27.864     7.76
   129a   1 pU:5         8980000       5.717     1.00        5.717     7.76
   129b   1              8962040       5.778     1.00        5.778     7.64
```

Events

If you have chosen to collect events, CXpa displays event reports for loops. The types of event reports displayed vary according to machine architecture and the type of event or events collected for the program run that generated the PDF file.

- Refer to the "Introducing metrics" online help topic or section of this book for more details on the types of event metrics that can be collected on a specific architecture.
- Refer to *Exemplar Architecture: S-Class and X-Class Servers* or *Exemplar SPP1000-Series Architecture* for more information about these architectures.

Cache misses and latency

Cache miss and latency reports vary according to the architecture and the exact type of cache miss event collected for the run of the program that generated the PDF file being analyzed.

On Exemplar S2000, Exemplar X2000, and SPP1600 Series systems, you can configure off-processor performance counters to collect cache miss counts and latency time for:

- Total cache misses
- Cache misses that were resolved locally (on the same hypernode as the processor—CTI not used)
- Cache misses that were resolved remotely (on a different hypernode—CTI used)

The cache miss report heading indicates the type of event collected for that run of your program, and can be one of the following:

- Locally resolved cache misses and latency
- Remotely resolved cache misses and latency
- Locally and remotely resolved cache misses and latency

On SPP 1200 and SPP1600 Series machines, you can configure on-processor performance counters to collect data cache accesses, misses, and latency, or instruction cache misses and latency. The cache misses and latency report heading indicates the type of event collected, and can be one of the following (data cache accesses are displayed in a separate report):

- Data cache misses and latency
- Instruction cache misses and latency

Cache misses and latency reports for loops enable you to examine:

- Total number of cache miss events that occurred in a loop.
- Latency—The total wall clock time spent accessing memory to locate data or instructions not found in the processor's cache.
- % CPU—Ratio of latency time to CPU time, expressed as a percentage.

  The % CPU value indicates how much of your computational work is being performed vs. the memory latency you are incurring in a source region for the active thread. If the percentage is low, it means that the CPU found all of the data it needed to do its job in the cache (close at hand). If the percentage is high, it means the CPU had to spend a lot of time asking the memory system to retrieve the data it needed relative to the amount of work it had to do.

This percentage must be looked at in light of the amount of CPU work available to make a significant observation about program performance in light of cache effects. For example, if the event latency/CPU (% CPU) is 75%, and the overall CPU time is 25 secs for the routine, then it is probably safe to say that cache contention is occurring. You could then try to search out the offending memory access pattern in the region and correct it. If the event latency/CPU (% CPU) is 75%, but the overall CPU time is only 25 milliseconds, then the data is probably not significant.

- Differences in latency and cache miss counts among threads can indicate data access patterns for threads that cause memory bank contention among threads or cache thrashing.

The following sample cache miss report for loops was generated on an Exemplar S2000 system. For this program run, locally resolved cache misses were collected for both read and write operations. Locally resolved cache miss counts indicate the number of cache misses that are resolved by using the hypernode crossbar to access memory found on the same hypernode as the processor. The CTI (Coherent Toroidal Interconnect) is not used.

```
================================================================================
                          Loop Performance Analysis
                              For: convolregion
================================================================================
Optimized Loops:
            Locally and Remotely Resolved Cache Misses and Latency
                          Times                    (less inner)
 Line     NL Optimization  Exec       Number of        Latency       % CPU     PS
-------  -- -----------  --------   ---------------  ------------- --------  --
   128    0                160000            398597          0.226     0.13%
   129a   1 pU:5          7192000           3746200          2.081     3.16%
   129b   1               7015760            294035          0.174     0.33%

                          Times                    (plus inner)
 Line     NL Optimization  Exec       Number of        Latency       % CPU     PS
-------  -- -----------  --------   ---------------  ------------- --------  --
   128    0                160000           4438832          2.481     0.85%
   129a   1 pU:5          7192000           3746200          2.081     3.16%
   129b   1               7015760            294035          0.174     0.33%

================================================================================
```

Data cache accesses (SPP1200 and SPP1600 Series only)

On SPP1200 and SPP1600 Series systems, you can configure on-processor event counters to collect data cache misses, accesses, and latency. A separate Data Cache Accesses report provides the following information:

- Total number of data cache accesses
- Data cache hit rate (`% Hits` column)—The data cache hit rate is calculated as follows:

$$data\_cache\_hit\_rate = \frac{total\_data\_cache\_accesses - data\_cache\_misses}{total\_data\_cache\_accesses} \times 100$$

If the data cache hit rate (`% Hits`) is low, this indicates cache thrashing.

A sample Data Cache Accesses report for loops is shown below:

```
================================================================================
                          Loop Performance Analysis
                             For: convolregion
================================================================================
Optimized Loops:
                                Data Cache Accesses
                          Times                    (less inner)
 Line    NL Optimization  Exec       Number of                    % Hits    PS
-------  -- ----------- --------   ---------------                --------  --
   128    0              1000000      8845854107                    98.9%
   129a   1 pU:5         8980000      1926684484                    97.7%
   129b   1              8962040      1859999803                    97.5%

                          Times                    (plus inner)
 Line    NL Optimization  Exec       Number of                    % Hits    PS
-------  -- ----------- --------   ---------------                --------  --
   128    0              1000000     12632538394                    98.5%
   129a   1 pU:5         8980000      1926684484                    97.7%
   129b   1              8962040      1859999803                    97.5%
```

Instructions completed and clock cycles (SPP1200 and SPP1600 Series only)

On SPP1200 and SPP1600 Series systems, you can configure on-processor event counters to collect instructions completed and clock cycles. A separate Instructions Completed and Clock Cycles report provides the following information:

- Number of instructions completed
- Average clock cycles (cycles per instruction)
- Average MIPS (millions of instructions per second) rate, calculated only if wall clock time is also collected

The average clock cycles metric measures the efficiency of instruction scheduling for the region being profiled. It is computed during analysis if instruction counts and clock cycles are collected. On SPP1200 systems, the maximum number of instructions that can be executed in a single clock cycle is 2. This means that the theoretical peak value for the average clock cycles metric on this architecture is 0.5.

If the value for average clock cycles is high, and it is associated with a frequently called routine that performs only a small amount of work (for example, a routine that simply assigns a new value to a variable and returns), inlining the routine could improve the instruction scheduling by eliminating the routine call overhead.

Code sections that contain many patterns of consecutive loads and stores followed by immediate use of requested operands can also result in high average clock cycles.

In some cases, the instruction scheduling can be improved by compiling the routine at a higher optimization level or, if programming at the assembly level, changing the order of instructions. Refer to the *Hewlett-Packard PA-RISC 1.1 Architecture and Instruction Set Reference Manual* (Manual Part No. 09740-90039) for information on how to achieve optimal instruction scheduling.

The average MIPS (millions of instructions per second) rate is calculated during analysis if wall clock time is also collected. The formula CXpa uses to calculate the average MIPS rate is as follows:

$$Average\_MIPS\_rate = \frac{Number\_of\_instructions\_completed}{Wall\_clock\_time\_in\_sec}$$

The theoretical peak MIPS rate for Hewlett-Packard PA-RISC 7100/7200 processors with a 10-ns clock cycle at a clock rate of 100 MHz is 200 MIPS. There is a direct correlation between average clock cycles and average MIPS rates; decreasing average clock cycles will result in increased average MIPS rates.

A sample Instructions Completed and Clock Cycles report for loops is shown below:

```
================================================================================
                          Loop Performance Analysis
                              For: convolregion
================================================================================
Optimized Loops:
                  Instructions Completed and Clock Cycles
                                                 (less inner)
                          Times                        Avg Clock   Avg MIPS
 Line    NL Optimization  Exec       Number of          Cycles      Rate     PS
------- -- ----------- --------   ---------------  ------------- --------  --
   128    0             1000000       14263931087            1.6       63
   129a   1 pU:5        8980000        3838782979            1.9       54
   129b   1             8962040        3624657999            2.0       51

                                                 (plus inner)
                          Times                        Avg Clock   Avg MIPS
 Line    NL Optimization  Exec       Number of          Cycles      Rate     PS
------- -- ----------- --------   ---------------  ------------- --------  --
   128    0             1000000       21727372065            1.7       59
   129a   1 pU:5        8980000        3838782979            1.9       54
   129b   1             8962040        3624657999            2.0       51

================================================================================
```

Data and instruction TLB misses (SPP1200 and SPP1600 Series only)

On SPP1200 and SPP1600 Series machines, you can configure on-processor event counters to collect data and instruction TLB (translation lookaside buffer) misses for profiled regions.

The TLB is a hardware structure in each CPU in an SPP Series system that contains information necessary to translate a virtual memory reference to a physical page and to validate memory accesses. The TLB contains 120 entries that represent address translations from virtual to physical memory for the most recently referenced page table entries.

TLB miss metrics represent the number of times the address translation from virtual to physical memory for data or an instruction to be referenced was not found in the TLB and could not be resolved through hardware handling.

When a TLB miss occurs, the operating system must then perform the address translation and store it in the TLB so that it can determine if the data is resident in memory:

- If the number of data TLB misses is high, this can indicate poor data locality. To correct the problem, try laying out frequently referenced data structures in an application closer together and not crossing page boundaries. In addition, focus on making the most of each data structure referenced by processing it completely before proceeding. This supports locality of references and maximizes use of the TLB.
- If the number of instruction TLB misses is high, try changing the link order so that routines that exhibit high TLB miss counts are placed next to each other on the application's link line. A more permanent source alternative is to cut-and-paste source code for routines with high TLB miss counts into one source file and relink it into your application.

Refer to the *Exemplar SPP1000-Series Architecture* reference for more information about the TLB. Refer to the *Exemplar Programming Guide* for more information on maximizing data locality.

The following example shows data and instruction TLB miss reports for loops:

```
================================================================================
                          Loop Performance Analysis
                              For: convolregion
================================================================================

Optimized Loops:
                                    Data TLB Misses
                          Times                        (less inner)
 Line     NL Optimization  Exec        Number of                              PS
-------  -- -----------  --------   ---------------                           --
   128    0                10000                0
   129a   1 pU:5          223000                0
   129b   1               212112                0

                          Times                        (plus inner)
 Line     NL Optimization  Exec        Number of                              PS
-------  -- -----------  --------   ---------------                           --
   128    0                10000                0
   129a   1 pU:5          223000                0
   129b   1               212112                0

Optimized Loops:
                                  Instruction TLB Misses
                          Times                        (less inner)
 Line     NL Optimization  Exec        Number of                              PS
-------  -- -----------  --------   ---------------                           --
   128    0                10000              501
   129a   1 pU:5          223000              174
   129b   1               212112              124

                          Times                        (plus inner)
 Line     NL Optimization  Exec        Number of                              PS
-------  -- -----------  --------   ---------------                           --
   128    0                10000              799
   129a   1 pU:5          223000              174
   129b   1               212112              124

================================================================================
```

Context

To display Loop reports in GUI mode from the Analysis Report window, select Loops (all) as the region level.

To display Loop reports in line mode, use the `analyze loop` command.

Related Commands	`analyze`	`set visibility`
Related Topics	Dynamic Call Graph report Parallel Region reports Reports	Introducing metrics Report fields Routine reports
Related Windows	Analysis Report window Region Subset Selection dialog	Call Graph window

Parallel Region reports

Description

Parallel region performance analysis reports display metrics for profiled parallel loop regions in your program. Parallel loops are created by the Exemplar compilers at optimization level `+O3 +Oparallel`.

Parallel region reports are only available if:

- The source files containing the parallel loops you wish to profile are compiled with an Exemplar compiler with the +pa option at optimization level `+O3 +Oparallel`.
- Your program contains parallel loops, and they were selected for profiling.
- At least one profiled parallel loop region was executed.

NOTE: The loop nesting level setting affects the number of loops selected for profiling. The default loop nesting level setting (a fixed loop nesting level range with a minimum of 0 and a maximum of 0) only selects loops at nesting level 0 (outermost loops) for profiling.

Refer to the "Profile Selection dialog" or "`set visibility`" command online help topics or sections of this book for more information about loop nesting level settings.

Depending on the architecture on which you created the PDF (performance data file) and the type of metrics collected, the following reports can be displayed for each routine executed in your program that contains profiled parallel loops:

- **Time to solution**—Includes wall clock time, CPU/wall clock time, and chunk counts.
- **Events**—Available event reports and metrics differ according to machine architecture and the type of event collected during the run of the program that generated the PDF file:
  - Off-processor event reports (Exemplar S2000, Exemplar X2000, and SPP1600 Series systems)—Locally resolved cache misses, remotely resolved cache misses, or locally and remotely resolved cache misses.
  - On-processor event reports (SPP1200 and SPP1600 Series systems)—Data cache misses and latency; data cache accesses; instruction cache misses and latency; instructions completed and clock cycles; and data and instruction TLB misses.

Each parallel loop region report contains two sections:

- `Optimized Loops (cumulative, including spawn/join overhead)`—These metrics are cumulative across all threads executing in the parallel region and include spawn and join overhead. These values are graphed in the 2D Profile window for parallel loops.
- `Optimized Loops (by thread, excluding spawn/join overhead)`—These metrics are calculated on a per-thread basis for all threads executing in the parallel region and do not include spawn and join overhead. These values are graphed in the 3D Profile window for parallel loops.

Refer to the "Report fields" online help topic or section of this book for information about fields (columns), abbreviations, and annotations that can appear in parallel region reports.

Refer to the "Reports" or "`set visibility`" online help topics or sections of this book for information about filtering options for reports.

NOTE: All CXpa reports express time in seconds unless the time is annotated with the letter "m" for milliseconds.

Reports

For parallel regions, two types of reports are available: Time to Solution and Events. The output of these reports are described in the following sections, along with sample reports.

Time to solution

The metrics in the Time to Solution report for parallel loops can be used to examine the following for each parallel loop:

- Total wall clock time for each parallel loop or each thread of a parallel loop.
- Distribution of work across threads—By looking at CPU time, wall clock time, and/or chunk counts, you can spot parallel loop regions where the load does not appear to be balanced across threads. You can also examine event metrics for those regions to see whether performance bottlenecks are due to cache effects.

  A *chunk* is a unit of work executed on a single thread in a parallel loop. This unit of work corresponds to a packet of iterations of a parallelized loop assigned to execute on a single thread. CXpa currently does not track the number of iterations in a chunk (chunk size).

- Ratio of CPU time to wall clock time.

  If the CPU/wall clock ratio is high (approaches 1.0 for serial regions or single threads or, for parallel regions, approaches *n*, where *n* is the number of processors used by the program), the region is compute-bound.

  If the CPU/wall clock ratio is low, it indicates that there may be some type of performance bottleneck caused by one or more of the following:

  - I/O calls (for example, read() or write() calls).
  - System calls (for example, open() or close() calls).
  - Memory accesses (for example, cache misses). You can compare event metrics and latency for these loops to find out if the bottleneck is due to memory accesses.

  For parallel loops, the CPU/wall clock ratio is the concurrency factor for the parallel loop. (Parallel loops are annotated with a "P" in the Optimization column.) Values that approach *n*, where *n* is the number of processors being used by your program, indicate good use of resources within a parallel region.

  For example, as you increase the number of processors or the amount of work (data set size), the concurrency factor should increase proportionately. This would indicate that the region is scaling well in parallel.

A sample Time to Solution report for a parallel loop region is shown below. The cumulative CPU/wall clock ratio for all threads indicates good parallel concurrency, and a comparison of metrics for individual threads indicates an even distribution of work among threads for this loop.

```
================================================================================
                    Parallel Loop Performance Analysis
                                For: convol
================================================================================

Optimized Loops (cumulative, including spawn/join overhead):

                                Time to Solution
 Line      Times Exec                 CPU          Wall Clock    CPU/Wall  PS
-------  ------------           ---------------  -------------  --------  --
    76a             1                   29.964          6.587      4.55

Optimized Loops (by thread, excluding spawn/join overhead):

                                Time to Solution
 Line      TID  Chunks                 CPU          Wall Clock    CPU/Wall  PS
-------  ---  --------           ---------------  -------------  --------  --
    76a     0         1                 3.734          6.212      0.60
            1         1                 3.701          5.418      0.68
            2         1                 3.716          5.591      0.66
            3         1                 3.723          5.741      0.65
            4         1                 3.762          6.586      0.57
            5         1                 3.727          6.370      0.59
            6         1                 3.728          6.397      0.58
            7         1                 3.698          5.454      0.68
================================================================================
```

Events

If you have chosen to collect events, CXpa displays event reports for parallel loop regions. The types of event reports displayed vary according to machine architecture and the type of event or events collected for the program run that generated the PDF file.

- Refer to *Exemplar Architecture: S-Class and X-Class Servers* or *Exemplar SPP1000-Series Architecture* for more information about these architectures.
- Refer to the "Introducing metrics" online help topic or section of this book for more details on the types of event metrics that can be collected on a specific architecture.

Cache misses and latency

Cache misses and latency reports vary according to the architecture and the exact type of cache miss event collected for the run of the program that generated the PDF file being analyzed.

On Exemplar S2000, Exemplar X2000, and SPP1600 Series systems, you can configure off-processor performance counters to collect cache miss counts and latency time for total cache misses, cache misses that were resolved locally (on the same hypernode as the processor—CTI not used), or cache misses that were resolved remotely (on a different hypernode—CTI used). The cache miss report heading indicates the type of event collected for that run of your program, and can be one of the following:

- Locally resolved cache misses and latency
- Remotely resolved cache misses and latency
- Locally and remotely resolved cache misses and latency

On SPP1200 and SPP1600 Series systems, you can configure on-processor performance counters to collect data cache accesses, misses, and latency or instruction cache misses and latency. The cache misses and latency report heading indicates the type of event collected, and can be one of the following (data cache accesses are displayed in a separate report):

- Data cache misses and latency
- Instruction cache misses and latency

Cache misses and latency reports for loops enable you to examine:

- Total number of cache miss events that occurred in a loop.
- Latency—The total wall clock time spent accessing memory to locate data not found in the cache.
- % CPU—Ratio of latency time to CPU time, expressed as a percentage.

  The % CPU value indicates how much of your computational work is being performed vs. the memory latency you are incurring in a source region for the active thread. If the percentage is low, it means that the CPU found all of the data it needed to do its job in the cache (close at hand). If the percentage is high, it means the CPU had to spend a lot of time asking the memory system to retrieve the data it needed relative to the amount of work it had to do.

This percentage must be looked at in light of the amount of CPU work available to make a significant observation about program performance in light of cache effects. For example, if the event latency/CPU (% CPU) is 75%, and the overall CPU time is 25 secs for the routine, then it is probably safe to say that cache contention is occurring. You could then try to search out the offending memory access pattern in the region and correct it. If the event latency/CPU (% CPU) is 75%, but the overall CPU time is only 25 milliseconds, then the data is probably not significant.

- Balance of work distributed among threads—Chunk counts directly indicate how well work is distributed across threads.
- Differences in latency and cache miss counts among threads can indicate data access patterns for threads that cause memory bank contention among threads or cache thrashing.

The following sample cache miss report for loops was generated on an SPP1000 Series system. For this program run, locally resolved cache misses were collected for both read and write operations. Locally resolved cache miss counts indicate the number of cache misses that are resolved by using the hypernode crossbar to access memory found on the same hypernode as the processor (the CTI is not used).

```
================================================================================
                        Parallel Loop Performance Analysis
                                    For: munge
================================================================================

Optimized Loops (cumulative, including spawn/join overhead):

                          Local Cache Miss Events
 Line      Times Exec                  Number of       Latency       % CPU     PS
-------  ------------              ---------------  -------------  --------  --
    36a             1                      394419          0.196     14.2%

Optimized Loops (by thread, excluding spawn/join overhead):
                          Local Cache Miss Events
 Line     Thread Id     Number of       Latency        Chunks       % CPU     PS
-------  ----------  -------------  -------------  -------------  --------  --
    36a             0         394419          0.196              1     56.6%
                    1         394504          0.192              1     56.2%
                    2         391414          0.194              1     56.4%
                    3         389436          0.201              1     57.2%
================================================================================
```

Data cache accesses (SPP1200 and SPP1600 Series only)

On SPP1200 and SPP1600 Series systems, you can configure on-processor event counters to collect data cache misses, accesses, and latency. A separate Data Cache Accesses report provides the following information:

- Total number of data cache accesses
- Data cache hit rate (`% Hits` column)—The data cache hit rate is calculated as follows:

$$data\_cache\_hit\_rate = \frac{total\_data\_cache\_accesses - data\_cache\_misses}{total\_data\_cache\_accesses} \times 100$$

If the data cache hit rate (`% Hits`) is low, this indicates cache thrashing.

A sample Data Cache Accesses report for a parallel loop is shown below:

```
================================================================================
                      Parallel Loop Performance Analysis
                                  For: convol
================================================================================
Optimized Loops (cumulative, including spawn/join overhead):

                              Data Cache Accesses
 Line      Times Exec                Number of                  % Hits    PS
-------  ------------           ---------------                --------  --
    76a             1                794645275                    98.5%

Optimized Loops (by thread, excluding spawn/join overhead):

                              Data Cache Accesses
 Line     TID  Chunks                Number of                  % Hits    PS
-------  ---  --------           ---------------                --------  --
    76a    0         1                99815818                    98.5%
           1         1                99019296                    98.5%
           2         1                99020036                    98.5%
           3         1                99032853                    98.5%
           4         1                99046744                    98.5%
           5         1                99026766                    98.5%
           6         1                99032982                    98.5%
           7         1                98690604                    98.5%
================================================================================
```

Instructions completed and clock cycles (SPP1200 and SPP1600 Series only)

On SPP1200 and SPP1600 Series systems, you can configure on-processor event counters to collect instructions completed and clock cycles. A separate Instructions Completed and Clock Cycles report provides the following information:

- Number of instructions completed
- Average clock cycles (cycles per instruction)
- Average MIPS (millions of instructions per second) rate, which is computed only if wall clock time is also collected

On SPP1200 and SPP1600 Series systems, the maximum number of instructions that can be executed in a single clock cycle is 2. This means that the theoretical best value for the average clock cycles metric on these architectures is 0.5.

Code sections that contain many patterns of consecutive loads and stores followed by immediate use of requested operands can also result in high average clock cycles.

In some cases, the instruction scheduling can be improved by compiling the routine at a higher optimization level or, if programming at the assembly level, changing the order of instructions. Refer to the *Hewlett-Packard PA-RISC 1.1 Architecture and Instruction Set Reference Manual* (Manual Part No. 09740-90039) for information on how to achieve optimal instruction scheduling.

The average MIPS (millions of instructions per second) rate is computed during analysis if wall clock time is also collected. The formula CXpa uses to calculate the average MIPS rate is as follows:

$$Average\_MIPS\_rate = \frac{Number\_of\_instructions\_completed}{Wall\_clock\_time\_in\_sec}$$

The theoretical peak MIPS rate for Hewlett-Packard PA-RISC 7100/7200 processors with a 10-ns clock cycle at a clock rate of 100 MHz is 200 MIPS. There is a direct correlation between average clock cycles and average MIPS rates; decreasing average clock cycles will result in increased average MIPS rates.

A sample instructions and clock cycles report for a parallel loop is shown below:

```
================================================================================
                      Parallel Loop Performance Analysis
                                  For: convol
================================================================================
Optimized Loops (cumulative, including spawn/join overhead):

                    Instructions Completed and Clock Cycles
                                                        Avg Clock    Avg MIPS
 Line      Times Exec                  Number of          Cycles       Rate     PS
-------  ------------              ---------------  -------------  --------  --
    76a             1                2119724437            1.4        72

Optimized Loops (by thread, excluding spawn/join overhead):

                    Instructions Completed and Clock Cycles
                                                        Avg Clock    Avg MIPS
 Line     TID  Chunks                  Number of          Cycles       Rate     PS
-------  ---  --------             ---------------  -------------  --------  --
    76a    0         1                  251029382            1.4        45
           1         1                  250018696            1.4        34
           2         1                  250125491            1.4        44
           3         1                  250087228            1.4        47
           4         1                  250059551            1.4        42
           5         1                  250077437            1.4        43
           6         1                  250021793            1.4        34
           7         1                  248867823            1.4        46

================================================================================
```

Data and instruction TLB misses (SPP1200 and SPP1600 Series only)

On SPP1200 and SPP1600 Series machines, you can configure on-processor event counters to collect data and instruction TLB (translation lookaside buffer) misses for profiled regions.

The TLB is a hardware structure in each CPU in an SPP Series system that contains information necessary to translate a virtual memory reference to a physical page and to validate memory accesses. The TLB contains 120 entries that represent address translations from virtual to physical memory for the most recently referenced page table entries.

TLB miss metrics represent the number of times the address translation from virtual to physical memory for data or an instruction to be referenced was not found in the TLB and could not be resolved through hardware handling. When a TLB miss occurs, the operating system must perform the address translation and store it in the TLB so that it can determine if the data is resident in memory:

- If the number of data TLB misses is high, this can indicate poor data locality. To correct the problem, try laying out frequently referenced data structures in an application closer together and not crossing page boundaries. In addition, focus on making the most of each data structure referenced by processing it completely before proceeding. This supports locality of references and maximizes use of the TLB.
- If the number of instruction TLB misses is high, try changing the link order so that routines that exhibit high TLB miss counts are placed next to each other on the application's link line. A more permanent source alternative is to cut-and-paste source code for routines with high TLB miss counts into one source file and relink it into your application.

Refer to the *Exemplar SPP1000-Series Architecture* reference for more information.

The following example contains data and instruction TLB miss reports for parallel loops:

```
================================================================================
                    Parallel Loop Performance Analysis
                              For: initialize
================================================================================
Optimized Loops (cumulative, including spawn/join overhead):

                                Data TLB Misses
 Line      Times Exec                Number of                              PS
-------  ------------            ---------------                            --
   152a             1                        31

Optimized Loops (by thread, excluding spawn/join overhead):

                                Data TLB Misses
 Line     TID  Chunks                Number of                              PS
-------  --- --------            ---------------                            --
   152a    0         1                        7
           1         1                        3
           2         1                        4
           3         1                        5
           4         1                        6
           5         1                        2
           6         1                        1
           7         1                        2

Optimized Loops (cumulative, including spawn/join overhead):

                             Instruction TLB Misses
 Line      Times Exec                Number of                              PS
-------  ------------            ---------------                            --
   152a             1                        83

Optimized Loops (by thread, excluding spawn/join overhead):

                             Instruction TLB Misses
 Line     TID  Chunks                Number of                              PS
-------  --- --------            ---------------                            --
   152a    0         1                       13
           1         1                        7
           2         1                       11
           3         1                        9
           4         1                       13
           5         1                        4
           6         1                        4
           7         1                        8
================================================================================
```

Parallel Region reports

Context

To view parallel loop region reports from the Analysis Report window in GUI mode, select Loops (parallel) as the region type.

To view parallel region reports in line mode, use the `analyze pregion` command.

Related Topics

Introducing metrics
Loop reports
Reports
Dynamic Call Graph report
Report fields
Routine reports

Related Windows

Analysis Report window
Region Subset Selection dialog
Call Graph window

Related Commands

`analyze`
`set visibility`

Report fields

This section lists and describes column headings, abbreviations, annotations, and other terms that appear in CXpa reports. These are listed in alphabetical order.

NOTE: All CXpa reports express time in seconds unless the time is annotated with the letter "m" for milliseconds.

- `% CPU`—Ratio of cache miss latency to CPU time, expressed as a percentage.
- `% Hits`—Data cache hit rate. SPP1200/SPP1600 Data Cache Accesses report only.
- `Avg clock cycles`—Average number of clock cycles per instruction. SPP1200/SPP1600 Instructions Completed and Clock cycles report only.
- `Avg MIPS rate`—Average number of MIPS (millions of instructions per second). SPP1200/SPP1600 Instructions Completed and Clock cycles report only.
- `Calls in`—For callers of the primary routine in each section of the Dynamic Call Graph report, the number of times the primary routine was called; for the primary routine in each section, the total number of calls made to that routine.
- `Calls out`—For callees of the primary routine in each section of the Dynamic Call Graph Report, the number of times the routine was called by the primary routine; for the primary routine, the total number of times that it was called.
- `Chunks`—Lists the total number of chunks executed by a single thread in the parallel region. A *chunk* is a unit of work executed on a single thread in a parallel loop. This unit of work corresponds to a packet of iterations of a parallel loop assigned to execute on a single thread. CXpa currently does not track the number of iterations in a chunk (chunk size).
- `CPU Time`—CPU time spent executing the region (accumulated time for all threads).
- `CPU/Wall`—For serial code, the ratio of CPU time to wall clock time measures CPU utilization. For parallel regions, this metric indicates the concurrency achieved through parallelism.

- `Data TLB Misses`—Number of times the address translation from virtual to physical memory for data to be referenced was not found in the translation lookaside buffer (TLB).
- `Instruction TLB Misses`—Number of times the address translation from virtual to physical memory for an instruction was not found in the translation lookaside buffer (TLB).
- `Latency`—The amount of time spent accessing memory to locate data or instructions not found in the processor's cache.
- `(less children)`—Does not include values for called routines. However, if an instrumented routine calls an uninstrumented routine, CXpa will not be able to separate the time spent in the uninstrumented child routine from the time spent in the instrumented parent.
- `(less inner)`—Excluding time spent in inner loops.
- `Line`—Source line number of the region.
  - Routines—Starting source file line number of the routine.
  - Serial and parallel loops—Line numbers for optimized loops annotated with a lowercase letter indicate that the loop was split into two or more loops during optimization. For parallel loops, the line number is the starting line number that maps back to the original source code.
- `Locally Resolved Cache Misses`—Cache misses that are resolved by using the hypernode crossbar to access memory found on the same hypernode as the processor; the CTI rings are not used.
- `m`—Time is expressed in milliseconds.
- `N/A`—Metric was not collected.
- `NL`—Nesting level of the loop after optimization.
- `Number of`—Number of times that the selected event occurred.

- `Optimization`—Abbreviation for the transformation(s) that the Exemplar compilers perform on loops. These abbreviations include:
  - `B:`*n*—Loop blocking; *n* is the blocking factor, which is the number of iterations that were blocked together.
  - `D`—Distributed.
  - `Ds`—Dynamic selection.
  - `Hs`—Hoisted.
  - `I`—Interchanged.
  - `P`—Parallel.
  - `PS`—Parallel strip-mined.
  - `pU:`*n*—The loop was partially unrolled. *n* is the loop unrolling factor, which is the number of iterations that were unrolled.

    For more information about automatic optimizations performed by the compiler and manual optimization techniques for Exemplar and SPP Series systems, refer to the *Exemplar Programming Guide.*
- `Optimized Loops (cumulative, including spawn/join overhead)`—These metrics are cumulative across all threads executing in the parallel region and include spawn and join overhead. These values are graphed in the 2D Profile window for parallel loops.
- `Optimized Loops (by thread, excluding spawn/join overhead)`—These metrics are calculated on a per-thread basis for all threads executing in the parallel region and do not include spawn and join overhead. These values are graphed in the 3D Profile window for parallel loops.
- `plus children`—Includes values for called routines.
- `plus inner`—Including time spent in inner loops.
- `PS`—Displays the profiling status. If this column is blank, then the region executed normally. Other possible profiling statuses are as follows:

Profiling status	Meaning
`e`	The program exited at this point.
`g`	This region could not be timed due to the granularity of the clock supported on the architecture.

m	Invalid time management was detected for this region due to an unprofilable code construct, an unprofilable command such as an `exec` or `fork`, or incorrect instrumentation in your program or in a library routine. To work around incorrect instrumentation: • Deselect regions in your routines that are listed with a profiling status of `m`. • Choose not to profile library routines if a library routine is listed with a profiling status of `m`.
p	The program was paused at this point, and the timing information is incomplete.
t	The program terminated at this point.
u	This region was uninstrumentable because it was too small to gather timing data or contains an unrecognized construct.
x	CPU time (excluding called routines and/or inner loops) and ratio(s) cannot be computed accurately.
y	Wall clock time (excluding called routines and/or inner loops) and ratio(s) cannot be computed accurately.
z	Latency time (excluding called routines and/or inner loops) and ratio(s) cannot be computed accurately.
. (period)	Dynamic call graph report only. The time displayed does not reflect a measured time, but rather a `gprof`-style "best-guess" time inferred from available profiling data.

- `read only`—Cache miss event collection was specified for misses that occurred during reads (loads) only.
- `Remotely Resolved Cache Misses`—Cache misses that were resolved by using the CTI rings to access memory on another hypernode.
- `TID`—Kernel thread ID number for each thread executing the parallel region.
- `Times Exec`—Number of times the routine was executed or, for loops, the number of loop invocations.

- `Wall Clock`—Time to solution for all threads executing the region. Wall clock time includes time when threads are idle.
- `write only`—Cache miss event collection was specified for misses that occurred during writes (stores) only.

Related Topics

Dynamic Call Graph report
Introducing metrics
Loop reports
Parallel Region reports
Reports
Routine reports

Related Windows

Analysis Report window
Call Graph window
Region Subset Selection dialog

Routine reports

Description

Routine performance analysis reports display metrics for profiled routines in your program.

Routine reports are only available if the source files containing regions you wish to profile at the routine level are instrumented for profiling at the routine level. To instrument routines:

- Compile the source files with the `+pa` option of the Exemplar compilers

  or

- Instrument the object files and or archive libraries containing the routines with the `/opt/cxpa/bin/cxoi` utility

NOTE: If an instrumented routine calls an uninstrumented routine, CXpa cannot separate the time spent or metrics collected in the uninstrumented child routine from the time spent in the instrumented parent.

Depending on the architecture on which you created the PDF (performance data file) and the type of metrics collected, the following reports can be displayed for each routine executed in your program that contains profiled routines:

- **Call counts**—Routine execution counts.
- **Computation**—CPU time and the percentage of total CPU time (% column), including and excluding time spent in called routines.
- **Time to Solution**—Wall clock time, percentage of total wall clock time, (% column), and CPU/wall clock time.
- **Events**—Available event reports and metrics differ according to machine architecture and the type of event collected during the run of the program that generated the PDF file. These reports include:
  - Off-processor event reports (Exemplar S2000, Exemplar X2000, and SPP1600 Series systems only)—Locally resolved cache misses, remotely resolved cache misses, or locally and remotely resolved cache misses.
  - On-processor event reports (SPP1200 and SPP1600 Series systems only)—Data cache misses and latency; data cache accesses; instruction cache misses and latency; instructions completed and clock cycles; and data and instruction TLB (translation lookaside buffer) misses.

For all routine reports, metrics are displayed:

- Excluding values for called routines (less children).
- Including values for called routines (plus children).

NOTE: All CXpa reports express time in seconds unless the time is annotated with the letter "m" for milliseconds.

Refer to the "Report fields" online help topic or section of this book for information about fields, column headings, abbreviations, and annotations that can appear in Routine reports.

Reports

For routines, four types of reports are available: Call Counts, Computation, Time to Solution, and Events. These reports are described in the following sections, along with sample reports.

Call counts

The Call Counts report displays the number of times each profiled routine in your program was executed (called). A sample Call Counts report is shown below:

```
================================================================================
                        Routine Performance Analysis
================================================================================
                                  Call Counts
   Times Exec     PS                          Routine Name
---------------- -- ------------------------------------------------------------
         1000000     convolregion
               1     initialize
               1     convol
               1 e   start
               1     matcon

(e) incomplete, contains exit.
================================================================================
```

Computation

The metrics in the Computation report for routines can be used to examine:

- Total CPU time spent in each profiled routine
- Percentage of program's total CPU time for each profiled routine

Analyzing this information at the routine level can help you identify which routines require the greatest portion of total execution time. You can then profile and analyze these routines in greater detail.

A sample Computation report for routines is shown below:

```
================================================================================
                        Routine Performance Analysis
================================================================================
                                    Computation
        (less children)       (plus children)
  CPU Time      %               CPU Time      %                 PS Routine Name
------------  -----             ------------  -----             -- ------------
      22.219   53.4%                  22.219   53.4%                convolregion
       2.573    6.2%                  41.276   99.3%                convol
       0.040    0.1%                   0.306    0.7%                initialize
       2.422m   0.0%                  41.585  100.0%             e  start
       0.765m   0.0%                  41.583  100.0%                matcon

(e) incomplete, contains exit.
================================================================================
```

Time to solution

The metrics in the Time to Solution report for routines can be used to examine the following:

- Total wall clock time spent in each profiled routine
- Percentage of total wall clock time spent in each profiled routine
- The ratio of CPU time to wall clock time for each profiled routine

If the CPU/wall clock ratio is low, it indicates that there may be some type of performance bottleneck caused by one or more of the following:

- I/O calls (for example, read() or write() calls).
- System calls (for example, open() or close() calls).
- Memory accesses (for example, cache misses). You can compare event metrics and latency for these loops to find out if the bottleneck is due to memory accesses.

For serial regions, if the CPU/wall clock ratio is high (approaches 1.0), the region is compute-bound.

For parallel regions, the CPU/wall clock ratio is the concurrency factor for the parallel region. Values for CPU/wall clock time that approach *n*, where *n* is the number of processors used by your program, indicate good parallel concurrency.

For example, as you increase the number of processors or the amount of work (data set size), the concurrency factor should increase proportionately. This would indicate that the region is scaling well in parallel.

A sample Time to Solution report for routines is shown below. The program that generated the PDF contained regions that executed in parallel on 8 CPUs. Routine `convolregion` (which perform the core computation and executes in parallel) exhibits good parallel concurrency.

```
================================================================================
                          CXpa Version 3.2 Profile

Executable      : /src/demos/convol/convol.-O3.exe
Profile Data    : /src/demos/convol/convol.-O3.exe.pdf
Process State   : exited
CPU Time        : 258.553  secs
Wall Clock Time : 55.352  secs
Architecture    : CONVEX SPP-1200 (8 cpus)
================================================================================
                        Routine Performance Analysis
================================================================================

                                Time to Solution
        (less children)                 (plus children)
 Wall Clock     %    CPU/Wall   Wall Clock     %    CPU/Wall   PS Routine Name
------------  -----  --------- ------------  -----  ---------  -- ------------
      47.888   86.5%      5.34        47.888   86.5%      5.34     convolregion
       6.615   11.9%      0.05        48.238   87.1%      5.36     convol
       6.584   11.9%      0.00        55.344  100.0%      4.67  e  start
       0.513    0.9%      0.01         0.513    0.9%      0.11     initialize
       9.270m   0.0%      0.09        48.760   88.1%      5.30     matcon

(e) incomplete, contains exit.
================================================================================
```

Events

If you have chosen to collect events, CXpa displays event reports for routines. The types of event reports displayed vary according to machine architecture and the type of event or events collected for the program run that generated the PDF file.

- Refer to the *Exemplar Architecture: S-Class and X-Class Servers* and the *Exemplar SPP1000-Series Architecture* reference manuals for more information about these architectures.

- Refer to the "Introducing metrics" online help topic or section of this book for more details on the types of event metrics that can be collected on a specific architecture.

Cache misses and latency

Cache misses and latency reports vary according to the architecture and the exact type of cache miss event collected for the run of the program that generated the PDF file being analyzed.

On Exemplar S2000, Exemplar X2000, and SPP1600 Series systems, you can configure off-processor performance counters to collect cache miss counts and latency time for:

- Total cache misses
- Cache misses that were resolved locally (on the same hypernode as the processor—CTI not used)
- Cache misses that were resolved remotely (on a different hypernode—CTI used)

The cache miss report heading indicates the type of event collected for that run of your program, and can be one of the following:

- Locally resolved cache misses and latency
- Remotely resolved cache misses and latency
- Locally and remotely resolved cache misses and latency

On SPP1600 Series systems, you can also specify whether the above events were collected read operations, write operations, or both. The default is reads and writes.

On SPP1200 and SPP1600 Series machines, you can configure on-processor performance counters to collect data cache accesses, misses, and latency or instruction cache misses and latency. The cache misses and latency report heading indicates the type of event collected, and can be one of the following (data cache accesses are displayed in a separate report):

- Data cache misses and latency
- Instruction cache misses and latency

Cache misses and latency reports for routines enable you to examine:

- Total number of cache miss events that occurred in a routine or routines.
- Latency—The total wall clock time spent accessing memory to locate data or instructions not found in the processor's cache.
- % CPU—Ratio of latency time to CPU time, expressed as a percentage.

The % CPU value indicates how much of your computational work is being performed vs. the memory latency you are incurring in a source region for the active thread. If the percentage is low, it means that the CPU found all of the data it needed to do its job in the cache (close at hand). If the percentage is high, it means the CPU had to spend a lot of time asking the memory system to retrieve the data it needed relative to the amount of work it had to do.

This percentage must be looked at in light of the amount of CPU work available to make a significant observation about program performance in light of cache effects. For example, if the event latency/CPU (% CPU) is 75%, and the overall CPU time is 25 secs for the routine, then it is probably safe to say that cache contention is occurring. You could then try to search out the offending memory access pattern in the region and correct it. If the event latency/CPU (% CPU) is 75%, but the overall CPU time is only 25 milliseconds, then the data is probably not significant.

- Differences in latency and cache miss counts indicate data access patterns for routines that cause memory bank contention among threads or cache thrashing.

The following sample cache miss report for routines was generated on an Exemplar S2000 system. For this program run, locally resolved cache misses were collected for both read and write operations. Locally resolved cache miss counts indicate the number of cache misses that are resolved by using the hypernode crossbar to access memory found on the same hypernode as the processor; the CTI (Coherent Toroidal Interconnect) rings are not used.

```
================================================================================
                        Routine Performance Analysis
================================================================================

                 Locally Resolved Cache Misses and Latency
        (less children)                    (plus children)              Routine
  Number of     Latency    % CPU    Number of     Latency    % CPU  PS  Name
- ------------ ----------- -----  ------------ ----------- -----  --  -------
       395101       0.221  16.6%        395101       0.221  16.6%      init
       394777       0.196  14.2%        394777       0.196  14.2%      munge
         1802       0.926m 47.8%        794315       0.420  15.4% e    start
         2635       1.312m 12.5%          2635       1.312m 12.5%      display

(e) incomplete, contains exit.

================================================================================
```

Data cache accesses (SPP1200 and SPP1600 Series only)

On SPP1200 and SPP1600 Series systems, you can configure on-processor event counters to collect data cache misses, accesses, and latency. A separate data cache accesses report provides the following information:

- Total number of data cache accesses
- Data cache hit rate (`% Hits` column)—The data cache hit rate is calculated as follows:

$$data\_cache\_hit\_rate = \frac{total\_data\_cache\_accesses - data\_cache\_misses}{total\_data\_cache\_accesses} \times 100$$

If the data cache hit rate (`% Hits`) is low, this indicates cache thrashing.

A sample Data Cache Accesses report for routines is shown below.

```
================================================================================
                        Routine Performance Analysis
================================================================================
                              Data Cache Accesses
        (less children)                    (plus children)              Routine
   Number of              % Hits     Number of              % Hits  PS  Name
---------------           ------  ---------------           ------  --  -------
       12834277            99.6%       12834277            99.6%      convolregion
        6109895             98.9%       18944172             99.4%     convol
        3438050             99.9%        3438050             99.9%     initial
          22505             92.4%       22409251             99.4%  e  start
           4524             92.1%       22386746             99.4%     matcon

(e) incomplete, contains exit.
```

Instructions completed and clock cycles (SPP1200 and SPP1600 Series only)

On SPP 1200 and SPP1600 Series systems, you can configure on-processor event counters to collect instructions completed and clock cycles. A separate Instructions Completed and Clock Cycles report provides the following information:

- Number of instructions completed
- Average clock cycles (cycles per instruction)
- Average MIPS (millions of instructions per second) rate, which is only calculated if wall clock time is also collected

The average clock cycles metric measures the efficiency of instruction scheduling. On SPP 1200 and SPP1600 Series systems, the maximum number of instructions that can be executed in a single clock cycle is 2. This means that the theoretical peak value for the average clock cycles metric on this architecture is 0.5.

If the value for average clock cycles is high, and it is associated with a frequently called routine that performs only a small amount of work (for example, a routine that simply assigns a new value to a variable and returns), inlining the routine could improve the instruction scheduling by eliminating the routine call overhead.

Code sections that contain many patterns of consecutive loads and stores followed by immediate use of requested operands can also result in high average clock cycles.

In some cases, the instruction scheduling can be improved by compiling the routine at a higher optimization level or, if programming at the assembly level, changing the order of instructions. Refer to the *Hewlett-Packard PA-RISC 1.1 Architecture and Instruction Set Reference Manual* (Manual Part No. 09740-90039) for information on how to achieve optimal instruction scheduling.

The average MIPS (millions of instructions per second) rate is calculated during analysis if wall clock time is also collected. The formula CXpa uses to calculate the average MIPS rate is as follows:

$$Average\_MIPS\_rate = \frac{Number\_of\_instructions\_completed}{Wall\_clock\_time\_in\_sec}$$

The theoretical peak MIPS rate for Hewlett-Packard PA-RISC 7100/7200 processors with a 10-ns clock cycle at a clock rate of 100 MHz is 200 MIPS. There is a direct correlation between average clock cycles and average MIPS rates; decreasing average clock cycles will result in increased average MIPS rates.

A sample Instructions Completed and Clock Cycles report for routines is shown below:

```
================================================================================
                        Routine Performance Analysis
================================================================================
                    Instructions Completed and Clock Cycles
          (less children)                      (plus children)
                  Avg Clock Avg MIPS                   Avg Clock Avg MIPS Routine
      Number of      Cycles  Rate        Number of      Cycles  Rate PS  Name
-------------------- ------ ----- -------------------- ------ ----- -- -------
            32912416    1.2   103             32912416    1.2   103    convolregion
             8126678    1.9    62              41039094    1.3    91    convol
              513145    2.1     7                513145    2.1     7    initial
               57797    5.1     0              41621472    1.3     4 e  start
               11436    5.6     3              41563675    1.3    90    matcon

(e) incomplete, contains exit.
================================================================================
```

Data and instruction TLB misses (SPP1200 and SPP1600 Series only)

On SPP1200 and SPP1600 Series machines, you can configure on-processor event counters to collect data and instruction TLB (translation lookaside buffer) misses for profiled regions.

The TLB is a hardware structure in each CPU in an SPP Series system that contains information necessary to translate a virtual memory reference to a physical page and to validate memory accesses. The TLB contains 120 entries that represent address translations from virtual to physical memory for the most recently referenced page table entries.

TLB miss metrics represent the number of times the address translation from virtual to physical memory for data or an instruction to be referenced was not found in the TLB and could not be resolved through hardware handling. When a TLB miss occurs, the operating system must perform the address translation and store it in the TLB so that it can determine if the data is resident in memory:

- If the number of data TLB misses is high, this can indicate poor data locality. To correct the problem, try laying out frequently referenced data structures in an application closer together and not crossing page boundaries. In addition, focus on making the most of each data structure referenced by processing it completely before proceeding. This supports locality of references and maximizes use of the TLB.

- If the number of instruction TLB misses is high, try changing the link order so that routines that exhibit high TLB miss counts are placed next to each other on the application's link line. A more permanent source alternative is to cut-and-paste source code for routines with high TLB miss counts into one source file and relink it into your application.

Refer to the *Exemplar SPP1000-Series Architecture* reference for more information on the TLB. Refer to the *Exemplar Programming Guide* for more information about programming on SPP Series systems.

A sample data and instruction TLB misses report for routines is shown below:

```
================================================================================
                        Routine Performance Analysis
================================================================================

                                Data TLB Misses
    (less children)                   (plus children)                Routine
      Number of                         Number of          PS          Name
-------------------------       -------------------------  --  --------------
                        9                               9      init
                        5                              15   e  start
                        1                               1      display
                        0                               0      munge

(e) incomplete, contains exit.

                             Instruction TLB Misses
    (less children)                   (plus children)                Routine
      Number of                         Number of          PS          Name
-------------------------       -------------------------  --  --------------
                       19                              43   e  start
                       13                              13      display
                       10                              10      init
                        1                               1      munge

(e) incomplete, contains exit.
```

Context

To display Routine reports in GUI mode, open the Analysis Report window and choose Routines as the region type.

To display Routine reports in line mode, use the `analyze routine` command.

Related Topics	Dynamic Call Graph report Loop reports Report fields	Introducing metrics Parallel Region reports Reports
Related Windows	Analysis Report window	Region Subset Selection dialog
Related Commands	`analyze`	

Windows

6

This chapter contains reference pages for all CXpa windows and dialogs. These pages are in alphabetical order by title. Each help page is divided into the following sections:

- **Description**—Explains the purpose and functionality of the window or dialog.
- **Fields**—Describes the purpose of each field that appears in the window or dialog.
- **Buttons**—Describes the function of each button that appears in the window or dialog.
- **Context**—Describes the action that makes this window or dialog appear.
- **Related Windows, Topics, and Commands**—Lists sections of this document that contain related information.

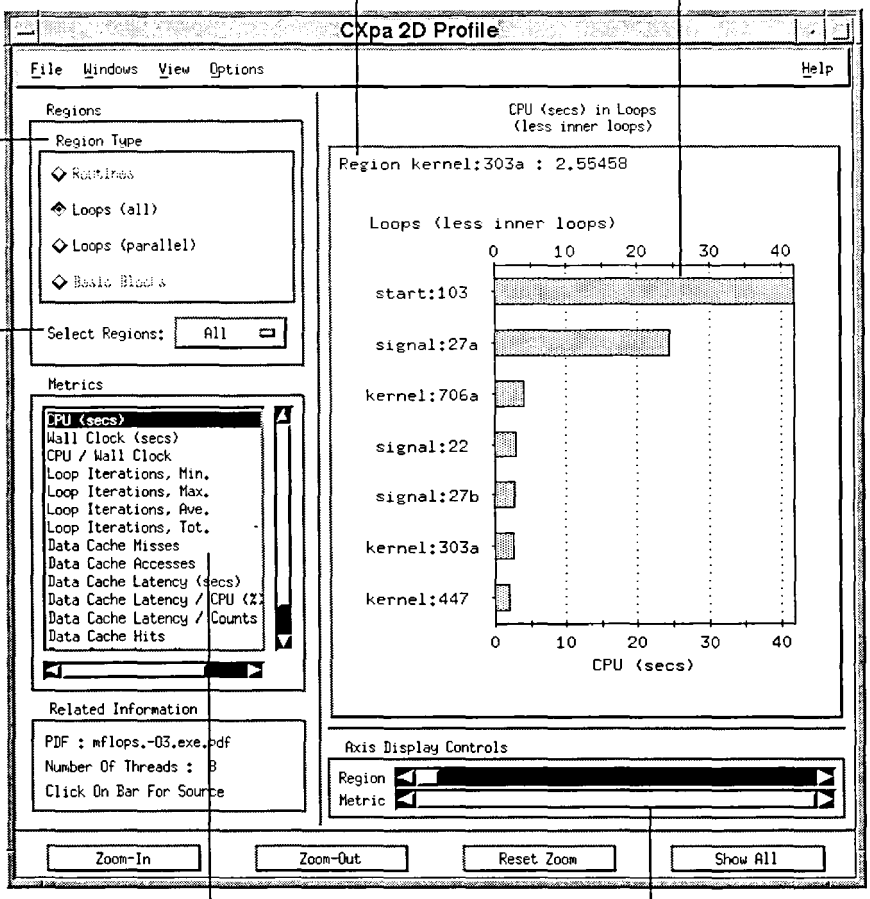
2D Profile window

Interactive data value display updates as you move the mouse over the graph, enabling you to view exact data values for regions in the graph.

Click with the left mouse button on any bar in the graph to view source code associated with a region.

Select the type of region to graph.

Select all regions of the selected type or a subset to graph.



Select the type of metric to graph.
The list is updated as different source code regions are selected, depending on the metrics available in the current PDF file.

Independently scroll Region and Metric axes.

2D Profile window

Description

The 2D Profile window displays a customizable two-dimensional graph of performance data for the selected regions of your program. This graph displays data for the entire process. To display performance data for individual threads, use the 3D Profile window.

In the 2D Profile window you can:

- Select different regions and metrics to graph.
- Click with the left mouse button on any bar in the graph to display associated source code.
- View exact data values for regions in the graph by moving the mouse over regions in the graph (data values are displayed in the upper left corner of the graph).
- Save 2D profile graphs in PostScript, xwd, or ASCII formats (select Save Profile from the File menu).
- Sort regions in 2D profile graphs by fixed metric, alphabetically, by load order, or by data values for the current metric (from lowest to highest or from highest to lowest). The default sort order is by data values for the current metric, from highest to lowest.

  To access sort options, select Sort from the View menu.
- View or specify the range of metric values displayed or the number of regions displayed at one time in the 2D graph using the Zoom dialog or buttons along the bottom of the window. This can be useful when there are a large number of data items to graph and you want to focus on a subset of the data.

  To access the Zoom dialog, select Zoom from the View menu.
- Use the scrollbars in the Axis Display Controls panel to scroll the Region and Metric axes independently.

By default, CXpa graphs metric values for all selected regions or for the maximum number of regions for which labels can legibly be displayed, whichever is less. If all selected regions are not displayed, use the Region Axis Display Control scrollbar to navigate the graph or use one of the other zoom options to change the graph display.

Menus

Name	Meaning
File	Contains items for saving the graph in a PostScript, ASCII, or xwd file format and closing the window.

Windows	Contains items for creating additional CXpa windows for viewing 2D or 3D profile graphs, performance reports, call graphs, or source files.
View	Contains items for sorting (Sort) and changing the range of values or source code regions displayed in the graph (Zoom).
Options	Contains an item (Filter Profile) that enables you to include or exclude values for called routines and/or inner loops in the graph.
Help	Contains items for invoking the CXpa help system.

Regions

Contains buttons that change the type of source code region to graph on the Region axis.

Name	Meaning
Routines	Graph selected metric for routines.
Loops (all)	Graph selected metric for loops.
Loops (parallel)	Graph selected metric for parallel loops only.
Select Regions	Option menu containing two items: **All**—Graph profiling data for all regions of the type specified above. **Subset**—Opens a Region Subset Selection dialog where you can specify a customized list of routines to graph profiling data for.

Metrics

Contains a scrolled list of metrics available in the current PDF where you can select the type of metric graphed on the Metric axis.

The available metric choices depend on the regions selected for profiling and the metrics that were collected for the run of your program that produced the PDF file.

Refer to the "Introducing metrics" online help topic or section of this book for metric descriptions and a list of metrics available by architecture.

Related Information

Displays the name of the PDF, the number of threads, and directions for displaying source code associated with any bar in the 2D profile graph.

2D Profile window

Axis Display Controls

Use the scrollbars in the Axis Display Controls panel to scroll the Region and Metrics axes independently.

Buttons

Name	Meaning
Reset Zoom	Restores the default graph display. By default, CXpa graphs metric values for all selected regions or for the maximum number of regions for which labels can legibly be displayed, whichever is less. If all selected regions are not displayed, use the Region Axis Display Control scrollbar to navigate the graph or use one of the other zoom options to change the graph display.
Zoom-In	Reduces the number of regions displayed in the 2D graph at one time by half.
Zoom-Out	Doubles the number of regions displayed in the 2D graph at one time.
Show All	Displays the entire graph. If the number of regions is so large that the region labels cannot be displayed legibly, labels are not displayed. If this happens, use the Reset Zoom or Zoom-In button to restore the labels and use scrollbars to navigate the graph.

Context

Use one of the following methods to open a 2D Profile window:

- Use the Create 2D Profile button in the Analysis Control window or the 2D Profile button in the Executable Manager window.
- Select 2D Profile from the Windows menu in any CXpa window.

Related Topics

Introducing metrics

Related Windows

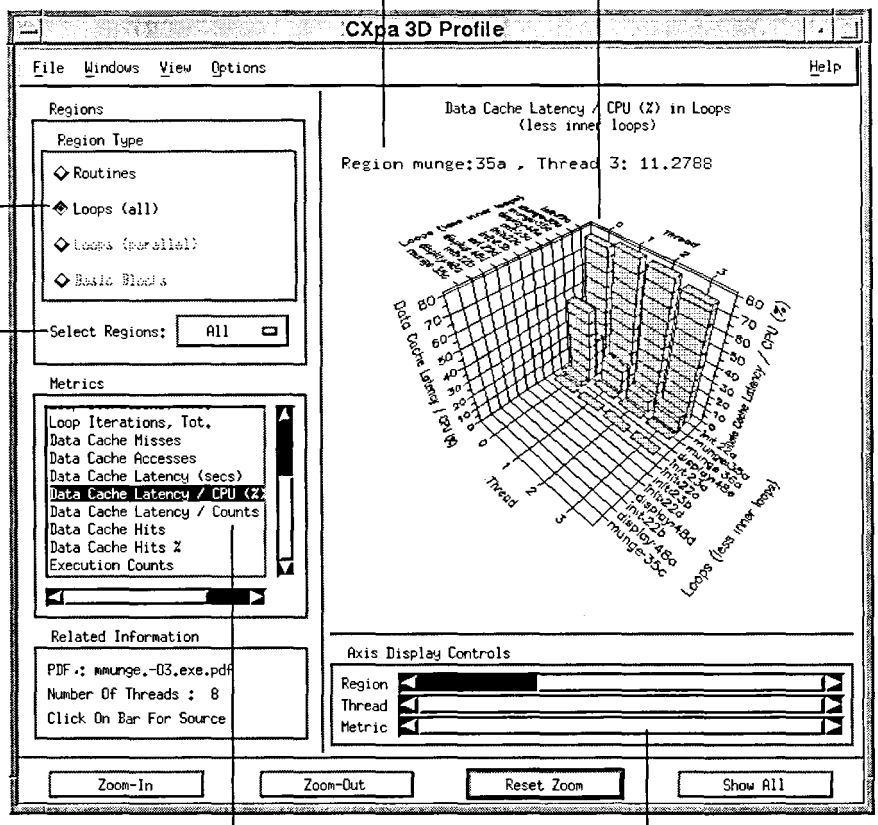
3D Profile window
Analysis Report window
Executable Manager window
Profile Selection dialog
Save Profile dialog
Source Code window
Analysis Control window
Call Graph window
Filter Profile dialog
Region Subset Selection dialog
Sort dialog
Zoom dialog

2D Profile window

3D Profile window

Interactive data value display updates as you move the mouse over the graph, enabling you to view exact data values for regions in the graph.

Click with the left mouse button on any bar in the graph to view source code associated with a region.



Select the type of region to graph.

Select all regions of the selected type or a subset to graph.

Select a metric to graph on the Metric axis. The list is updated as different source code regions are selected, depending on the metrics available in the current PDF file.

Independently scroll Region, Thread, and Metric axes.

3D Profile window

Description

The 3D Profile window displays a three-dimensional graph of parallel performance data (which vary according to architecture) for the profiled regions of your program. Performance data is graphed on a per-thread basis.

In the 3D profile window, you can:

- Select different regions and metrics to graph.
- Click with the left mouse button on any bar in the graph to display source code associated with the corresponding region.
- Save 3D profile graphs in PostScript, xwd, or ASCII file formats (select Save Profile from the File menu).
- View or specify the range of metric values displayed or the number of regions or threads displayed at one time in the 3D graph using the Zoom dialog or buttons along the bottom of the window. This can be useful when there are a large number of data items to graph and you want to focus on a subset of the data.

  To access the Zoom dialog, select Zoom from the View menu.
- View exact data values for regions in the graph by moving the mouse over regions in the graph. Data values are displayed in the upper left corner of the graph.
- Use the scrollbars in the Axis Display Controls panel to scroll the Region, Thread, and Metric axes independently.
- Sort regions in 3D profile graphs by fixed metric, alphabetically, by load order, or by data values for the currently selected metric (from lowest to highest or from highest to lowest). The default sort order is by data values for the current metric, from highest to lowest.

  To access sort options, select the Sort option from the View menu.
- Rotate the graph by placing the mouse pointer over the graph and holding down the middle mouse button.
  - To restrict the rotation to a single axis, press the **x**, **y**, or **z** key after pressing the mouse button, but before moving the mouse to rotate the graph.
  - To return the graph to its original position, use the Reset Zoom button.

By default, CXpa graphs metric values for all selected regions or for the maximum number of regions for which labels can legibly be displayed, whichever is less. If all selected regions or threads are not displayed, use the Region or Thread Axis Display Control scrollbars to navigate the graph or use one of the other zoom options to change the graph display.

Menus

Name	Meaning
File	Contains items for saving the graph in a PostScript, ASCII, or xwd file format and closing the window.
Windows	Contains items for creating additional CXpa windows for viewing 2D and 3D profile graphs, performance reports, and source files.
View	Contains items for sorting the data (Sort) and changing the range of values, number of source code regions, or number of threads displayed in the 3D graph (Zoom).
Options	Contains an item for including or excluding values for called routines and/or inner loops in the 3D graph.
Help	Contains options for invoking the CXpa help system.

Regions

Contains buttons that change the type of program region to graph on the Region axis.

Region level option	Action
Routines	Graph selected metric for routines.
Loops (all)	Graph selected metric for loops.
Loops (parallel)	Graph selected metric for parallel loops only.
Select Regions	Option menu containing two items: **All**—Graph profiling data for all regions of the type specified above. **Subset**—Opens a Region Subset Selection dialog where you can specify a customized list of routines to graph profiling data for.

Metrics

Contains a scrolled list of metrics available in the current PDF where you can select the type of metric graphed on the Metric axis.

The available metric choices depend on the regions selected for profiling and the metrics that were collected for the run of your program that produced the PDF file.

3D Profile window

Refer to the "Introducing metrics" online help topic or section of this book for metric descriptions and a list of metrics available by architecture.

Related Information

Displays the name of the PDF, the number of threads, and directions for displaying source code associated with any bar in the 3D profile graph.

Axis Display Controls

Use the scrollbars in the Axis Display Controls panel to scroll the Region, Thread, and Metrics axes independently.

Buttons

Name	Meaning
Reset Zoom	Restores the default graph display. By default, CXpa graphs metric values for all selected regions or for the maximum number of regions for which labels can legibly be displayed, whichever is less. If all selected regions or threads are not displayed, use the Region or Thread Axis Display Control scrollbars to navigate the graph or use one of the other zoom options to change the graph display.
Zoom-In	Reduces the number of regions displayed in the 3D graph at one time by half. The number of threads displayed remains constant.
Zoom-Out	Doubles the number of regions displayed in the 3D graph at one time. The number of threads displayed remains constant.
Show All	Displays the entire graph. If the number of regions or threads is so large that the region labels cannot be displayed legibly, labels are not displayed. If this happens, use the Reset Zoom or Zoom-In button to restore the labels and use the scrollbars to navigate the graph.

Context

Use one of the following methods to open a 3D Profile window:

- Press the Create 3D Profile button in the Analysis Control window or the 3D Profile button in the Executable Manager window.
- Select 3D Profile from the Windows menu of any CXpa window.

3D Profile window

Related Topics

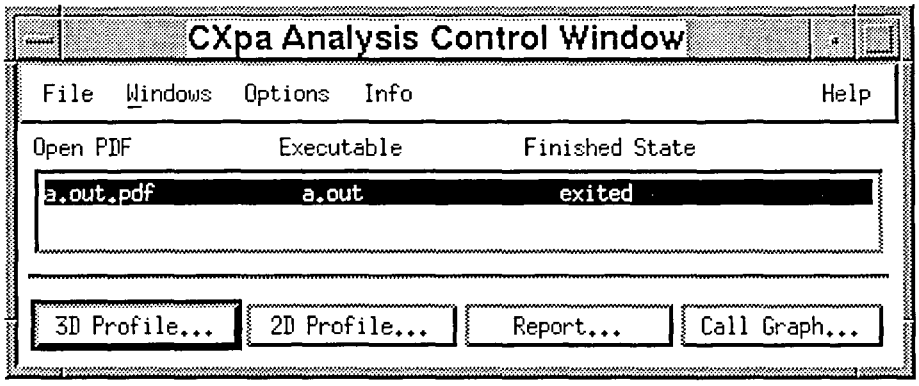
Introducing metrics

Related Windows

2D Profile window
Analysis Control window
Analysis Report window
Call Graph window
Executable Manager window
Filter Profile dialog
Profile Selection dialog
Region Subset Selection dialog
Save Profile dialog
Sort dialog
Zoom dialog

3D Profile window

Analysis Control window



Description

The Analysis Control window appears when you invoke CXpa with the names of one or more existing PDF files only (without specifying the name of an executable file). For example:

```
% /opt/cxpa/bin/cxpa *.pdf &
```

In this analysis-only mode of CXpa, you can view previously collected performance data, but you cannot instrument or run your application (to instrument and/or run your program under CXpa, invoke CXpa with the name of an executable file).

The current PDF is always highlighted. To analyze the data in the current PDF, use the 3D Profile, 2D Profile, Report or Call Graph button at the bottom of the window.

From the Analysis Control window you can:

- Create and view a dynamic call graph, performance reports or 2D and 3D profile graphs from the information in any number of PDFs, including PDFs created on different architectures or from different executables.
- Open multiple call graphs, profile graphs, and reports per PDF to compare performance analysis information.

- Merge data from multiple PDF files generated by the same executable into a single PDF file for analysis. This feature is useful for analyzing PDF files generated from MPI or PVM applications or for comparing performance data across multiple runs of an application with different data sets.
- View source files for your application by selecting Source Code from the Windows menu.

You can use the X application resource `Cxpa*defaultWindow` to specify automatic creation of a graph or report window when you invoke CXpa with a PDF file only. Valid values for `Cxpa*defaultWindow` are `All`, `None`, `2DProfile`, `CallGraph`, `3DProfile`, `Report`, or `Source`. The default is `None`.

Selecting a different PDF file to analyze

To select a different PDF file to analyze, select a PDF from the Open PDF list. The name of the PDF file is highlighted, indicating that it is the currently selected PDF. You can then use the 3D Profile, 2D Profile, Report, or Call Graph button to analyze the data in the current PDF.

Opening additional PDF files for analysis

To open a new PDF and make it the current PDF:

1. Select Open PDF from the File menu. This opens an Open PDF dialog for specifying the name of the new PDF file.
2. Specify the name of an existing PDF file by selecting it from the Files list in the Open PDF dialog or by explicitly typing it in the PDF file field.

   If you do not specify a relative or full path, CXpa looks for the PDF file in the current directory.
3. Choose OK. The name of the specified PDF file is displayed in the Open PDF list in the Analysis Control window, and it is highlighted, indicating that it is the currently selected PDF.

   You can now use the 3D Profile, 2D Profile, Report, or Call Graph button to analyze the new PDF.

Invoking CXpa with multiple PDF files for analysis

You can invoke CXpa from the shell in GUI mode with the names of multiple PDF files. For example:

```
% /opt/cxpa/bin/cxpa  *.pdf &
```

When you do so, each of the specified PDF files is automatically listed in the Open PDF column in the Analysis Control window, and the last PDF file specified on the command line is selected for analysis. To select or deselect a PDF file, highlight its name in the list.

Merging PDF files

You can combine profiling data from multiple PDF files generated from the same executable into a single PDF file for analysis. Use the merge feature when analyzing CXpa profiling data generated from MPI or PVM applications.

The PDF files to be merged must be generated from the same executable (called the base executable). In GUI mode, CXpa uses the executable file associated with the first PDF file specified in the Open PDF list in the Analysis Control window as the base executable.

To merge multiple PDF files:

1. Make sure that all PDF files to be merged are listed in the Open PDF list.

   The easiest way to do this is to invoke CXpa from the shell with a list of the PDF files to be merged.

2. Select Merge from the File menu to open a Merge PDFs dialog.

3. Specify the name of the PDF file that will contain the merged PDF data in the Merged PDF file field.

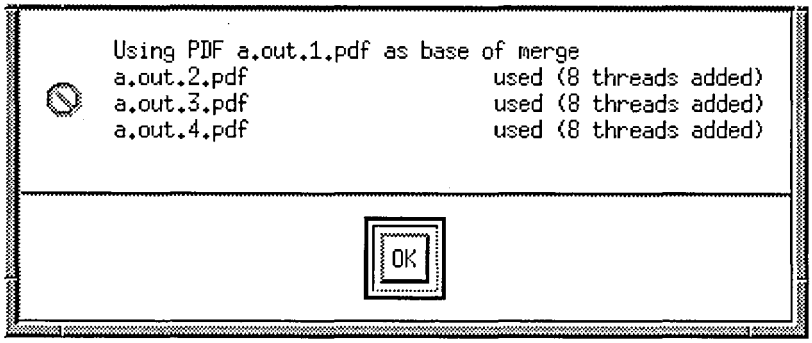
4. Choose OK.

   CXpa merges all PDF files listed in the Open PDF list in the Analysis Control window into a single PDF file, using the file name specified in the Merged PDF file field, then closes the Merge PDFs dialog.

   If an invalid PDF is encountered, an error message is displayed, and CXpa continues to merge the data from valid PDF files.

   When the merge is complete, CXpa displays a dialog that lists the name of the base PDF file associated with the new, merged PDF and the name of each PDF file merged. The base PDF file specifies the executable file associated with the merged PDF.

For example:



The name of the new PDF file is displayed at the bottom of the Open PDF list in the Analysis Control window, and it becomes the active PDF.

5. To analyze the profiling data in the merged PDF file, use one of the buttons at the bottom of the Analysis Control window (2D Profile, Call Graph, 3D Profile, or Report).

Refer to the "Profiling MPI and PVM applications with CXpa," online help topic or section of this book for more information about merging PDF files generated by message-passing applications.

Menus

Name	Meaning
File	Contains items for opening an existing PDF, merging data from multiple PDF files into a single PDF file, and exiting CXpa.
Windows	Contains items for creating additional windows for viewing 2D and 3D profile graphs, performance reports, dynamic call graphs, or source files.
Options	Contains an item for changing CXpa's source code search path.
Info	Contains an item for displaying information on the current PDF or your current session.
Help	Contains items for invoking the CXpa help system.

Analysis Control window

Fields

Name	Meaning
Open PDF	Lists the names of available PDFs. The current PDF is always highlighted.
Executable	Lists the name of the executable that created the PDF.
Finished State	Lists the state of the executable (for example, exited, stopped, terminated, and so on).

Buttons

Name	Action
3D Profile	Opens a window containing a 3D profile graph.
2D Profile	Opens a window containing a 2D profile graph.
Report	Displays a window containing textual performance reports.
Call Graph	Opens a window containing an interactive dynamic call graph display.

Context

The Analysis Control window appears when you invoke CXpa with the name of a PDF file only (without specifying an executable file).

Related Topics

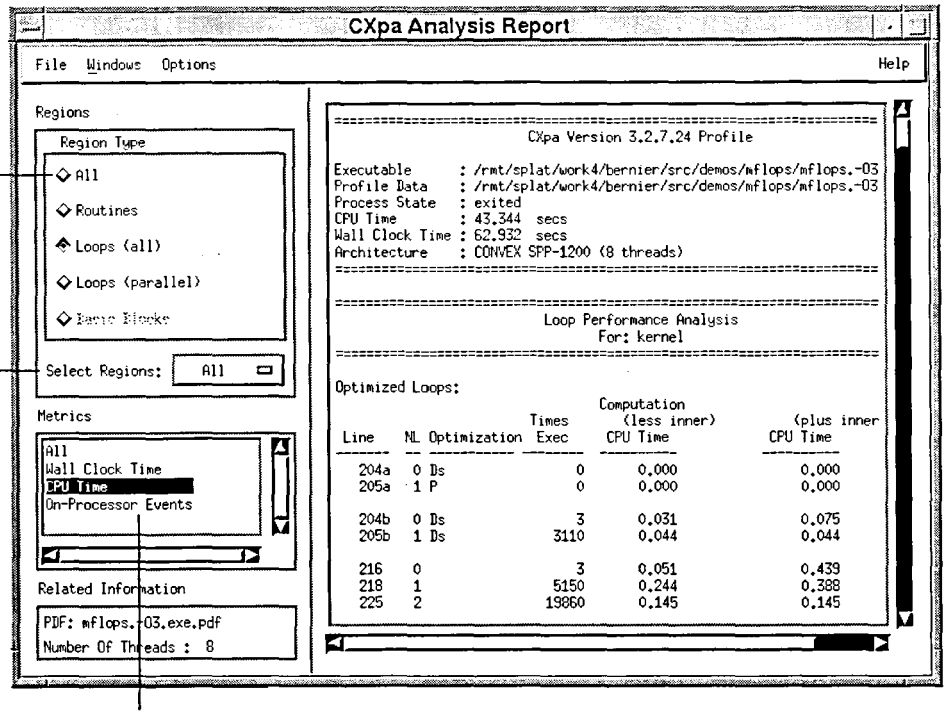
Analyzing PDFs only
Performance data files
Profiling MPI and PVM applications with CXpa
Using pre-instrumented executables

Related Windows

2D Profile window
Analysis Control window
Call Graph window
Open PDF dialog
Info Session dialog
Source Code window
3D Profile window
Analysis Report window
Merge PDFs dialog
Info PDF dialog
Source Code Selection dialog
Source Search Path dialog

Analysis Control window

Analysis Report window



Select the type of region for which you you want to view profile reports.

Select all routines or a subset of routines containing regions of the type selected above.

Select the type of metrics to view in profile reports. The list is updated as different source code regions are selected, depending on the metrics available in the current PDF file.

Description

The Analysis Report window displays textual performance reports generated from the data in the active PDF. You can create multiple Analysis Report windows, allowing you to compare information among several PDFs or to examine different types of analysis reports for a single PDF.

By default, when you first bring up the Analysis Report window, all available reports and metrics are displayed for all profiled regions of your program.

Analysis Report window

To display specific metrics for a specific type of source code region:

1. Select the region type using one of the buttons in the Region Type area of the window.
2. Choose a metric from the Metrics list.

You can create a customized report by specifying a subset of routines that contain regions of the currently selected type (refer to the "Region Subset Selection dialog" online help topic or section in this book for more information).

To specify whether you want to display performance data in reports on a per-process basis (the default) or on a per-thread basis, select Filter Report from the Options menu. Threaded data in reports is displayed on a per-routine basis for each thread, with the number of the thread displayed in the report heading for each routine.

For more information on the types of reports available at each region level and the metrics displayed in each type of report, refer to the following online help topics or sections in this book: "Reports," "Dynamic Call Graph report," "Loop reports," "Parallel Region reports," "Report fields," and "Routine reports."

Menus

Name	Options
File	Contains items for saving the report in the window to a file and closing the window.
Windows	Contains items for creating additional windows for viewing 2D and 3D profile graphs, performance reports, call graphs, or source files.
Options	Contains an item for filtering performance report data on a per-process basis (the default) or on a per-thread basis.
Help	Contains various items for invoking the CXpa Help system.

Regions

Contains options to change the type of source code region for which reports are displayed.

Region Type option	Action
All	Report selected metrics for all profiled regions.
Routines	Report selected metrics for routines.
Loops (all)	Report selected metrics for all profiled loops.
Loops (parallel)	Report selected metrics parallel loops only.
Select Regions	Contains an option menu with two items: **All**—Report profiling data for all regions of the type specified above. **Subset**—Open a Region Subset Selection dialog where you can specify a subset of routines that contain regions of the currently selected type to include profiling data for in reports.

Metrics

The Metrics panel contains a scrolled list of metrics available in the current PDF where you can select the type of metrics you want to view in CXpa profile reports.

The available metric choices depend on the regions selected for profiling and the metrics that were collected for the run of your program that produced the PDF file.

Refer to the "Introducing metrics" online help topic or section of this book for metric descriptions and a list of metrics available by architecture.

Related Information

Label	Meaning
PDF	Displays the name of the file in which the current performance data is stored.
Number of Threads	Displays the number of threads in your program.

Context

To open an Analysis Report window:

- Use the Report button on the Executable Manager window or the Create Report button on the Analysis Control window.
- Select Report from the Windows menu in any CXpa window.

Analysis Report window

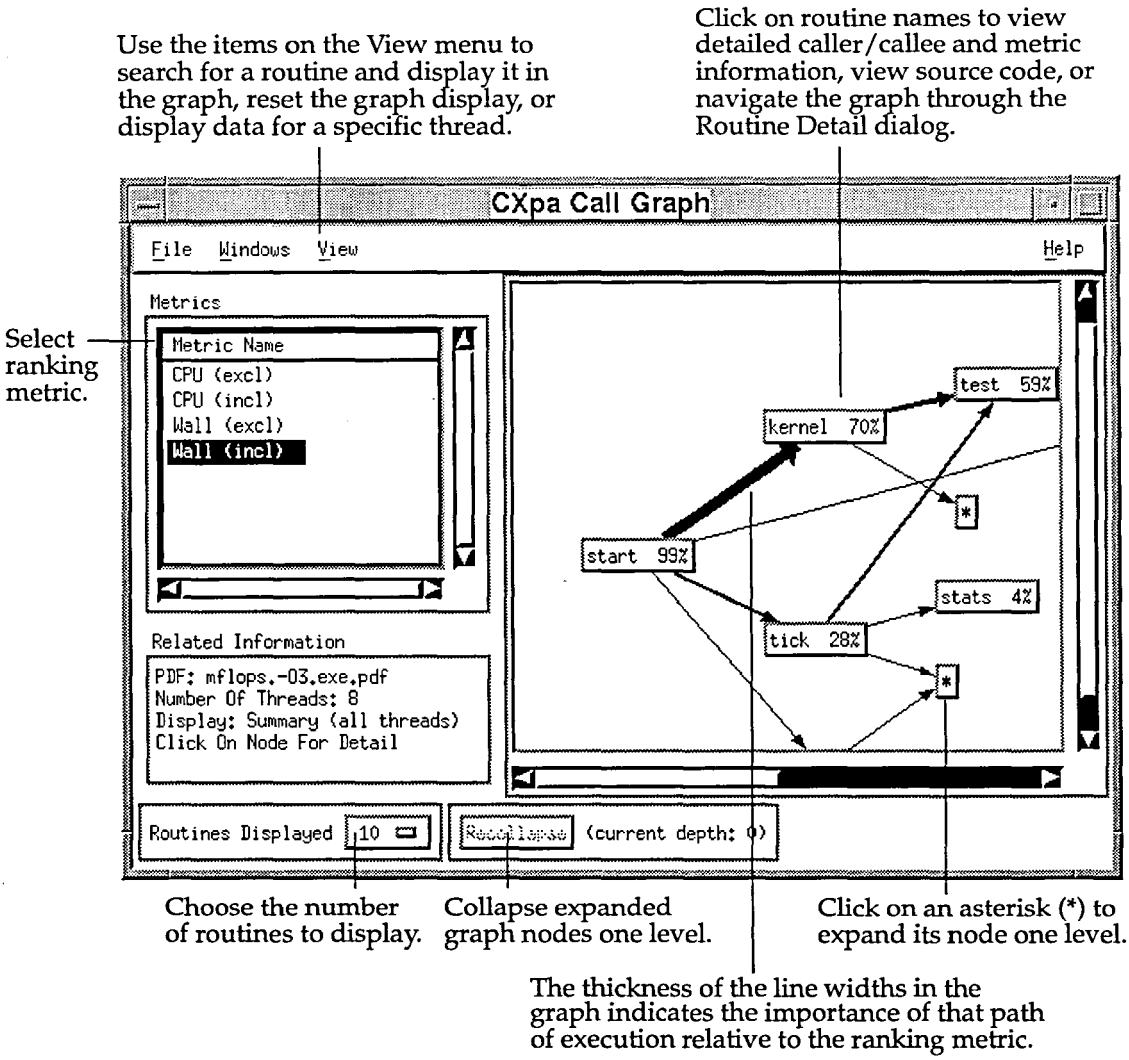
Related Windows

2D Profile window
Analysis Control window
Region Subset Selection dialog
Filter Report dialog
Save Report dialog
3D Profile window
Call Graph window
Executable Manager window
Profile Selection dialog

Related Topics

Dynamic Call Graph report
Loop reports
Report fields
Routine reports
Introducing metrics
Parallel Region reports
Reports

Call Graph window



Description

The Call Graph window displays a dynamic call graph of the routines in your program.

To collect the profiling data needed to display a dynamic call graph:

- Prepare all routines in the program for routine-level profiling by compiling them with the `+pa` option of the Exemplar compilers and/or using the `/opt/cxpa/bin/cxoi` utility to instrument the object files and archive libraries.
- Make sure all routines are selected for profiling in the Profile Selection dialog (in GUI mode) or enter the following command (in line mode):

  ```
  (CXpa) select routine all
  ```

- Enable CPU time, wall clock time, and Call Graph metric collection in the Profile Selection dialog (in GUI mode) or specify the `call_graph`, `cpu`, and `wall_clock` parameters of the `collect` command (in line mode). You can collect any other additional metrics, but these three must be collected to obtain a dynamic call graph. For example:

  ```
  (CXpa) collect cpu wall_clock call_graph
  ```

NOTE: When generating a dynamic call graph, make sure that all routines in your program are instrumented and selected for profiling. Otherwise, the results displayed in the call graph may be misleading.

For example, if an instrumented routine named `parent()` calls an uninstrumented routine `child()`, and `child()` then calls an instrumented routine `sub1()`, the call graph would incorrectly show that `parent()` called `sub1()`, and all of the time spent in the uninstrumented routine `child()` would be attributed to `sub1()`.

When a call graph is first opened, the top 10 routines, ranked by inclusive wall clock time (that is, including time spent in called routines) are displayed.

The percentage of the program total for the selected metric that is attributed to each of the top 10 routines is displayed to the right of the routine name.

The rest of the routines are collapsed into nodes that are indicated by asterisks (*) in the graph. The width of lines in the graph indicate the most important path of execution for the selected metric.

Collapsed nodes can be expanded and viewed by clicking on asterisks with the mouse. When a collapsed node is expanded, the top *n* routines in the collapsed node (still ranked by the currently selected metric) are displayed. The number of routines displayed is controlled by the Routines Displayed option menu at the bottom of the window. The default is 10 routines.

Changing the number of routines to display

To change the number of routines displayed in the graph:

1. Position the mouse over the Routines Displayed option menu at the bottom of the Call Graph window. Click and hold down the left mouse button and drag the mouse to make a selection.
2. When you release the mouse, the call graph display updates to show the chosen number of routines, ranked by the currently selected metric.

To recollapse an expanded node one level, use the Recollapse button in the lower right-hand corner of the Call Graph window. The current depth is displayed to the right of the Recollapse button.

Selecting a different metric to rank routines

You can change the metric used to rank the routines. Available metric choices are wall clock time, CPU time (inclusive or exclusive), and any other metric collected during the program run that generated the current PDF file.

To change the ranking metric, select a metric from the Metrics list in the upper left corner of the Call Graph window. The selected metric is highlighted.

The call graph display updates to display currently selected number of routines, ranked by the selected metric.

Viewing detailed information and source code for routines

To view detailed information and source code for a routine in the Call Graph window, click on the name of the desired routine in the graph to open a Routine Detail dialog. From this dialog you can:

- View values for all metrics collected for the selected routine.
- View lists of callers and callees and their call counts for the selected routine.
- Use the Show in Source button to open a Source Code window displaying source code for the selected routine.
- Select different routines to display in the Call Graph window.

Searching for a routine

To search for a routine and display it in the Call Graph window:

1. Select Find Routine from the View menu to open a Find Routine dialog.

2. Select the routine to display in the Call Graph window by highlighting its name in the list.

   If the program contains a large number of routines, you can specify a string in the Find field to search for, then press Find to locate the first routine name containing that string. CXpa scrolls the routine list so that the matching routine name is placed at the top of the list (or, if it is near the end of the list, it is visible in the list).

   Press Find again to locate the next routine that contains the specified string and scroll the list, and so on. The search will wrap to the beginning of the list. Be sure to click on the routine name to select it after executing the search.

3. Choose OK.

   CXpa highlights the routine name in the Call Graph window and, if necessary, scrolls the window so that the currently selected routine is showing. If the selected routine is in a node that is currently collapsed, its node is expanded until the routine is displayed.

   The information associated with the selected routine is displayed in the Routine Detail dialog. If a Routine Detail dialog is not currently open, CXpa opens one.

Displaying information for specific threads

By default, the dynamic call graph displays summary information for all threads of the process. To display information for a specific thread:

1. Select Thread from the View menu to open a Thread Selection dialog that contains a scrolling list of thread ID numbers.
2. Highlight the thread ID number in the list to select it. You can select one thread ID number or All Threads.
3. Choose OK to close the dialog and update the call graph display.

Menus

Name	Meaning
File	Contains an item for closing the window.
Windows	Contains items for creating additional CXpa windows for viewing 2D and 3D profile graphs, performance reports, and source files.
View	Contains items for resetting the call graph display, searching for a specific routine to display, and displaying data for a specific thread in the call graph.

Help	Contains items for invoking the CXpa help system.

Buttons and Fields

Name	Description/Action
Metric	Contains a scrolling list of metric names to select as the ranking metric for routines displayed in the Call Graph window. The selected metric is highlighted.
Related Information	Displays the name of the current PDF file, the total number of threads in the program, and whether data is displayed for all threads or for a specific thread.
Routines Displayed	Option menu for specifying the number of routines to be displayed. The default is the top 10 routines, ranked by inclusive wall clock time. Available choices are 10, 15, 20 or 100.
Recollapse	Collapse all expanded Call Graph nodes one level. This button is desensitized when all nodes are collapsed (depth 0).

Context

To open a Call Graph window:

- Use the Create Call Graph button in the Analysis Control window or the Call Graph button in the Executable Manager window.
- Select Call Graph from the Windows menu of any CXpa window.

Related Topics

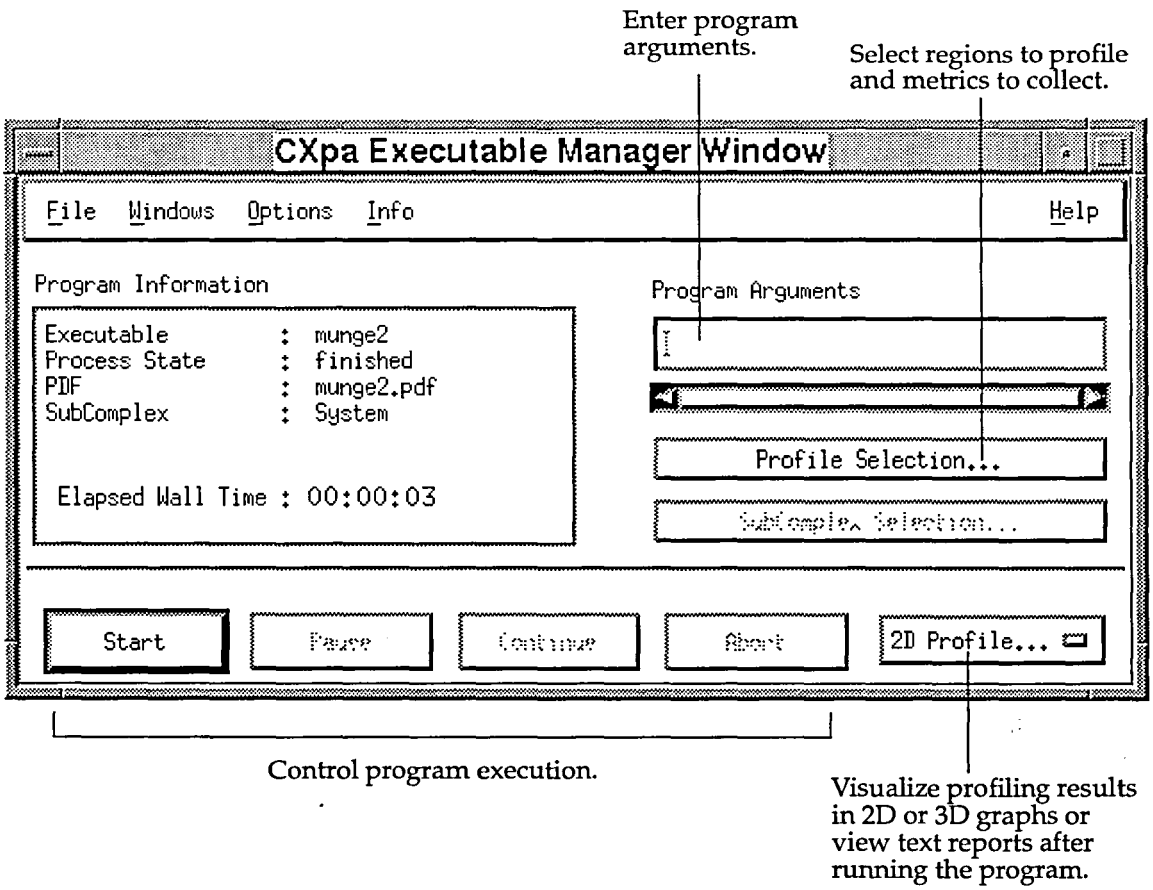
Dynamic Call Graph report

Related Windows

Find Routine dialog
Routine Detail dialog
Thread Selection dialog

Call Graph window

Executable Manager window



Description

The Executable Manager window appears when you invoke CXpa with an executable. From this window, you can profile a program that has been prepared for profiling with CXpa.

From the Executable Manager window you can:

- Select source code regions for profiling and metrics to collect.
- Specify arguments to the program you are profiling.

- Run your program to collect performance data.
- Display 2D or 3D profile graphs, call graphs, or textual performance reports generated from the performance data you collected.
- View source files for your program.
- Specify a new PDF name to prevent overwriting an existing PDF.
- Overwrite any existing PDF by specifying the name of that PDF.

From the Executable Manager window, you cannot display performance reports and graphs from PDFs created in a previous session, generated from a different executable, or generated on a different architecture.

To display performance reports and profiles from previously created PDFs, including PDFs created on other architectures or with different executables, invoke CXpa with the name of one or more existing PDF files only (without specifying an executable file name).

To obtain the most accurate profiling data, do not pause your program during profiling. For best results, run the program to completion in a standalone environment (on a "quiet" or dedicated subcomplex).

Profiling a program

Assuming your program has been prepared for profiling with CXpa, perform the following steps to instrument and and run your program to collect profiling data:

1. If you want to run your process on a different subcomplex, use the Subcomplex Selection button to open a Subcomplex Selection dialog. Select the name of the subcomplex on which you want your process to run.
2. Enter any arguments to the program you are profiling in the Program Arguments field.

   By default, CXpa collects CPU time, wall clock time, and execution counts for all routines in your program. Use these defaults the first time you profile your program under CXpa.

   To use these defaults, skip to step 4. To select different source code regions (such as loops) for profiling and/or collect different metrics, continue with step 3.
3. Use the Profile Selection button to open a Profile Selection dialog. To select different metrics and regions for profiling:
   - Select the routines and loops you want to profile by using the buttons in the Select Regions to Profile section of the dialog.
   - To change the default set of metrics, use the buttons in the Select Metrics to Collect section of the dialog to select/deselect metrics.

- If you wish to collect event metrics (such as cache misses), select Events as one of your metrics choices and use the Event Counter(s) option menu to select the type of event that you want to collect.
- Choose OK to apply the region and metrics settings and close the Profile Selection dialog.

The next step describes how to run your program under CXpa's control. You can select Save or Save As from the File menu to write the region and metric selections you just made to the executable file or to a copy, exit CXpa, and then run the executable outside CXpa to generate performance data files for later analysis with CXpa. Refer to the "Using pre-instrumented executables" online help topic or section of this book for more information on how to do this.

4. Use the Start button to run the program and collect performance data. An xterm window appears that displays your program's input and output. By default, performance data is stored in a file named *<executable>*.pdf.

   When your program completes execution, the Process State field shows a state of `finished`.

   You can use the Pause button at the bottom of the Executable Manager window to suspend your program's execution during profiling.

   Once your program is paused, you can:

   - View an intermediate Call Graph, 2D or 3D profile graph, or text report, as shown in step 5.
   - Abort your program.
   - Continue your program's execution.

5. To display performance information, select 2D Profile, 3D Profile, Report, or Call Graph from the option menu at the bottom-right corner of the window or from the Windows menu.

6. To quit CXpa, choose Exit from the File menu.

Menus

Name	Meaning
File	Contains items for opening a new PDF, saving region and metric selections to the executable file being profiled or to a copy, and exiting CXpa.
Windows	Contains items for creating additional windows for viewing 2D and 3D profile graphs, performance reports, call graphs, or source files.

Executable Manager window

Options	Contains an item for changing the path CXpa uses to locate program source files.
Info	Lists items that display information about the PDF, the executable being profiled, and your current CXpa session.
Help	Contains items for invoking the CXpa help system.

Fields

Name	Meaning
Executable	Lists the executable you are profiling.
Process State	Lists the process state for the program you are profiling. These states include: `Running`, `Paused`, `Finished`, and `Terminated`.
PDF	Lists the name of the performance data file (PDF) where performance data will be stored.
SubComplex	Lists the name of the subcomplex on which your executable will run.
Elapsed Wall Time	Presents the actual time that has passed since you pressed the Start button.
Program Arguments	Allows you to specify command-line arguments to the program you are profiling and file redirection operations.

Buttons

Name	Action
Profile Selection	Displays a Profile Selection dialog for selecting regions to profile and metrics to collect.
SubComplex Selection	Displays the Subcomplex Selection dialog for choosing the subcomplex that your executable will run on.
Start	Runs the executable and initiates profile data collection.
Pause	Pauses the executable and profile data collection.
Continue	Continues a paused program and resumes profile data collection.
Abort	Terminates the program and stops profile data collection.

2D Profile	Displays an option menu for creating additional analysis windows. The choices are 2D Profile, 3D Profile, Report, and Call Graph. The default is 2D Profile.

Context

The Executable Manager window appears when you start CXpa with the name of an executable file. If you do not specify any options, CXpa looks for a file named a.out in the current directory.

Related Windows

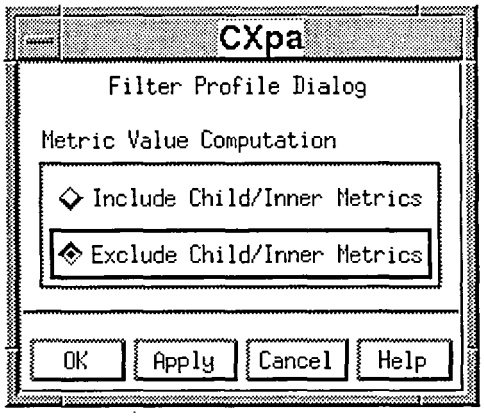
2D Profile window
Analysis Control window
Call Graph window
Info PDF dialog
Info Session dialog
Source Code window
Subcomplex Selection dialog
3D Profile window
Analysis Report window
Info Executable dialog
New PDF dialog
Profile Selection dialog
Source Search Path dialog

Related Topics

Analyzing PDFs only
Learning CXpa quickly
Introducing metrics
Introducing source code regions
Performance data files
Reports
Selecting metrics in GUI mode
Selecting regions in GUI mode
Using pre-instrumented executables

Executable Manager window

Filter Profile dialog



Description

Use the Filter Profile dialog to specify whether metrics graphed in the 2D Profile and 3D Profile windows include values for called routines (child routines) and/or inner loops. By default, they are excluded (Exclude Child/Inner Metrics).

Buttons

Heading	Meaning
Metric Value Computation	
Include Child/Inner	For routines, include metrics for called routines (child routines) when computing values for the 2D and 3D Profile graphs. For loops, include metrics for inner loops when computing values.
Exclude Child/Inner	For routines, exclude metrics for called routines (child routines) when computing values for the 2D and 3D Profile graphs. For loops, exclude metrics for inner loops when computing values.

Buttons

Name	Action
OK	Accepts the new settings, closes the dialog, and updates the graphs displayed in 2D and 3D Profile windows.

Filter Profile dialog

Apply	Applies the new settings without closing the dialog and updates the graphs displayed in 2D and 3D Profile windows.
Cancel	Closes this dialog without making any changes.
Help	Displays a help page for this dialog.

Context

To open a Filter Profile dialog, select Filter Profile from the Options menu in the 2D Profile window or the 3D Profile window.

Related Windows

2D Profile window

3D Profile window

Filter Report dialog

Description

Use the Filter Report dialog contains to specify whether information presented in CXpa reports is presented at the process level or the thread level. The default setting is Process.

Buttons

Name	Action
Level of Detail	
Thread	Sets the level of detail for information presented in performance analysis reports to include data for individual threads in all reports.
Process	Sets the level of detail for information presented in performance analysis reports to the process level. Performance data is summed across all threads of the process (except in parallel region reports).
OK	Accepts the new settings, closes the dialog, and updates the reports in Analysis Report windows.
Apply	Applies the new settings without closing the dialog and updates the reports in Analysis Report windows.

Filter Report dialog

Cancel	Closes this dialog without making any changes.
Help	Displays a help page for this dialog.

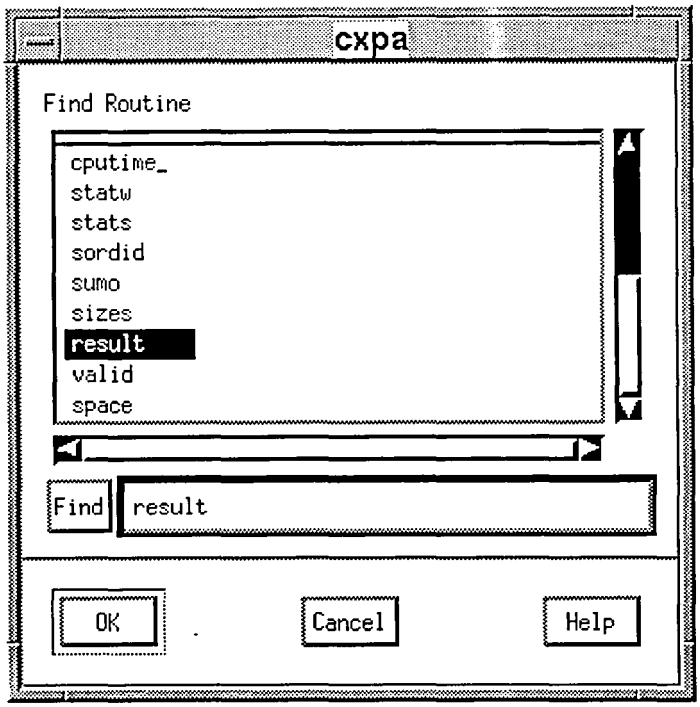
Context

To open a Filter Report dialog, select Filter Report from the Options menu in the Analysis Report window.

Related Windows

Analysis Report window
Region Subset Selection dialog
Save Report dialog

Find Routine dialog



Description

Use the Find Routine dialog to locate a routine and display it in the Call Graph window. When the Find Routine dialog is first opened, it displays a scrolling list of all profiled routines in your program.

To locate and display a routine:

1. Select the routine to display in the Call Graph window by highlighting its name in the list. You can select only one.

   If the program contains a large number of routines, you can specify a string in the Find field to search for, then use the Find button to locate the first routine name containing that string. CXpa scrolls the routine list so that the matching routine name is placed at the top of the list (or, if it is near the end of the list, it is visible in the list).

Use the Find button again to locate the next routine that contains the specified string and scroll the list so that it is placed at the top of the list, and so on. The search will wrap to the beginning of the list.

You must select the routine containing the string from the routine list after executing the search to make it the current routine.

2. Choose OK to close the dialog and display the routine in the Call Graph window.

   CXpa highlights the routine name in the Call Graph window and, if necessary, scrolls the window so that the currently selected routine is showing. If the selected routine is in a node that is currently collapsed, its node is expanded until the routine is displayed.

   The information associated with the selected routine is displayed in the Routine Detail dialog. If a Routine Detail dialog is not currently open, CXpa opens one.

Fields

Heading	Meaning
Find Routine	Contains a scrolled list of the profiled routines in your program. Highlight the name of a routine in the list to select it.

Buttons

Name	Action
Find	Searches the list of routine names for occurrences of the string specified in the text entry field opposite the Find button. If a match is found, the routine list scrolls so that the first matching routine name is placed at the top of the list (or, if it is near the end of the list, it is visible in the list). Use the Find button again to locate the next routine that contains the specified string and scroll the list so that it is placed at the top of the list, and so on. The search will wrap to the beginning of the list.
OK	Highlights the routine name in the Call Graph window and, if necessary, scrolls the window so that the currently selected routine is showing. If a Routine Detail dialog is not currently open, CXpa opens one that displays information for the selected routine.
Cancel	Closes the dialog.

Find Routine dialog

Help	Displays a help page for this dialog.

Context

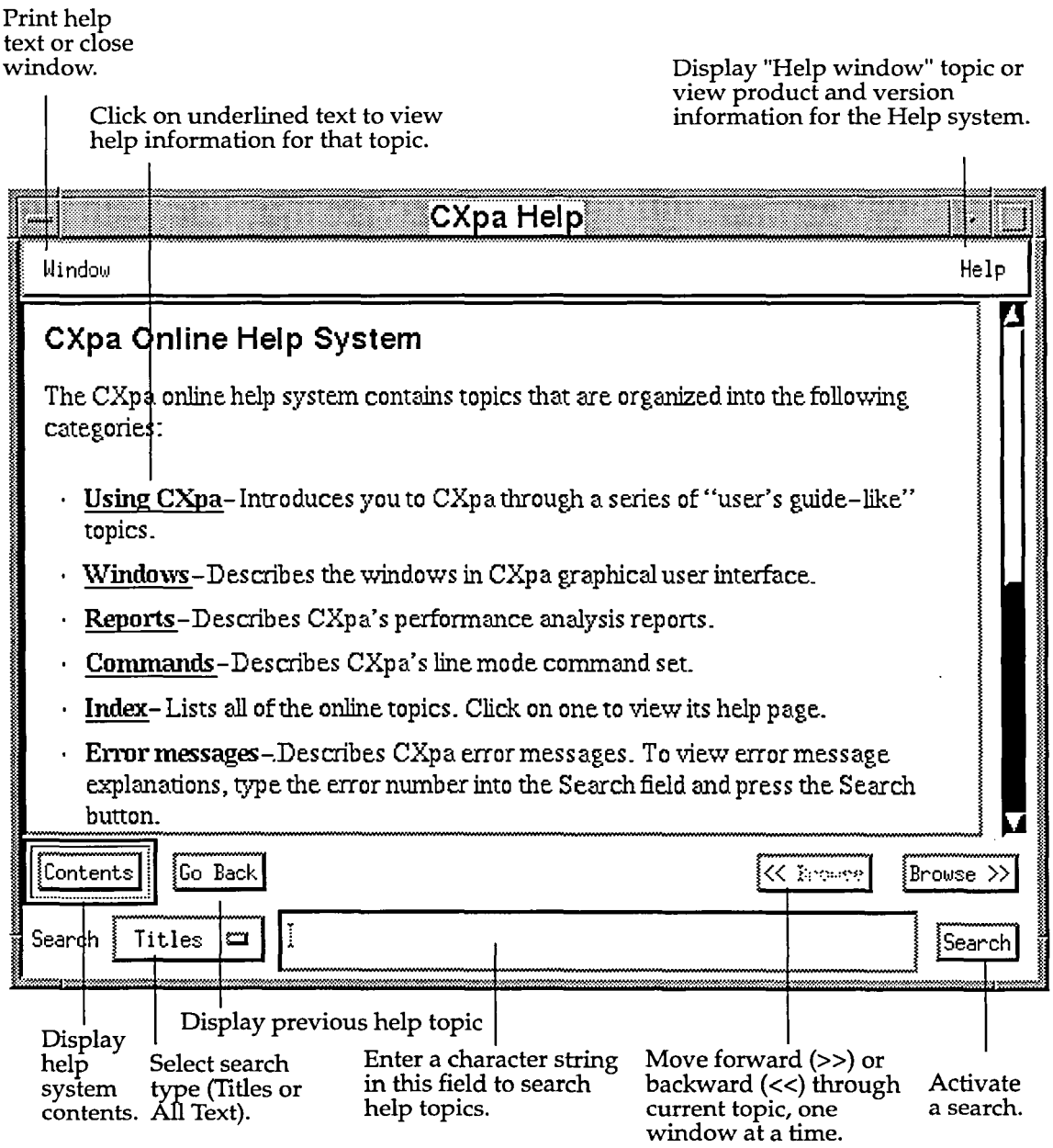
To open a Find Routine dialog, select Find Routine from the View menu in the Call Graph window.

Related Windows

Call Graph window
Routine Detail dialog
Thread Selection dialog

Find Routine dialog

Help window


Print help text or close window.
Click on underlined text to view help information for that topic.
Display "Help window" topic or view product and version information for the Help system.
CXpa Help
Window
Help
CXpa Online Help System
The CXpa online help system contains topics that are organized into the following categories:
· Using CXpa–Introduces you to CXpa through a series of "user's guide-like" topics.
· Windows–Describes the windows in CXpa graphical user interface.
· Reports–Describes CXpa's performance analysis reports.
· Commands–Describes CXpa's line mode command set.
· Index–Lists all of the online topics. Click on one to view its help page.
· Error messages–Describes CXpa error messages. To view error message explanations, type the error number into the Search field and press the Search button.
Contents
Go Back
<< Browse
Browse >>
Search
Titles
Search
Display previous help topic
Display help system contents.
Select search type (Titles or All Text).
Enter a character string in this field to search help topics.
Move forward (>>) or backward (<<) through current topic, one window at a time.
Activate a search.

Help window

Description

Use the CXpa Help window to:

- Navigate the CXpa online Help system
- Print a formatted ASCII verion of the Help text
- Search help topics (by title or full text)
- Display instructions for using the Help system
- Display version information for the help system
- Display online help for CXpa messages by entering the message number (for example, A35) in the Search string field

Menus

Name	Items
Window	Contains the following items: **Print Text**—Pipes a formatted ASCII text version (without graphics) of the current help topic to the `lp` command. The `lp` command looks for the printer specified in your `PRINTER` environment variable. If this variable is not set, nothing is printed. **Close**—Closes the Help window.
Help	Contains the following items: **On Help**—Displays instructions for using the CXpa Help system. **On Version**—Displays a Product Information dialog that identifies the version of the CXpa Help system (Interactive Documentation Toolkit) you are using.

Buttons

Name	Action
Contents	Displays a hyperlinked table of contents for the CXpa online help system.
Go Back	Displays the previous topic stored in the help history. The help history contains all the topics you viewed during the current session.
Browse >>	Moves forward through the current help topic, one window length at a time. This button is deactivated when you reach the end of a topic.

<< Browse	Moves backward through the current help topic, one window length at a time. This button is deactivated when you reach the beginning of a topic.
Search	Searches for the character string you entered in the Search field.
Titles/All Text	Selects whether to search only help topic titles or the full text of all help topics.

Context

To open a Help window:

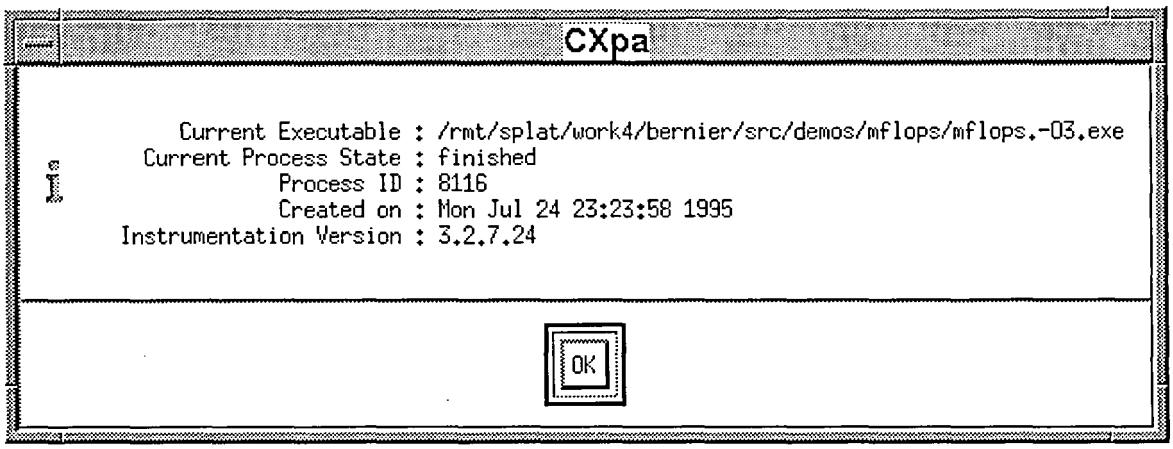
- Select Window Overview, CXpa Overview, Using Help, or Tutorial from the Help menu in the upper right corner of any CXpa window.
- Use the Help button on any CXpa dialog.
- Execute a `help` command from the command line at the `(CXpa)` prompt in the Command window.

Related Topics

Using online help

Help window

Info Executable dialog



Description

The Info Executable dialog displays information about the executable you are profiling.

Fields

Name	Meaning
Current Executable	Name of the executable being profiled.
Current Process State	Current state of the process you are profiling. The states include `running`, `paused`, `terminated`, `exited`, and `finished`.
Process ID	Process ID for the program you are profiling.
Created on	Date that the executable was created.
Instrumentation Version	Version of the CXpa profiling routines (cxpamon.o) linked into your program.

Buttons

Name	Action
OK	Closes this dialog.

Info Executable dialog

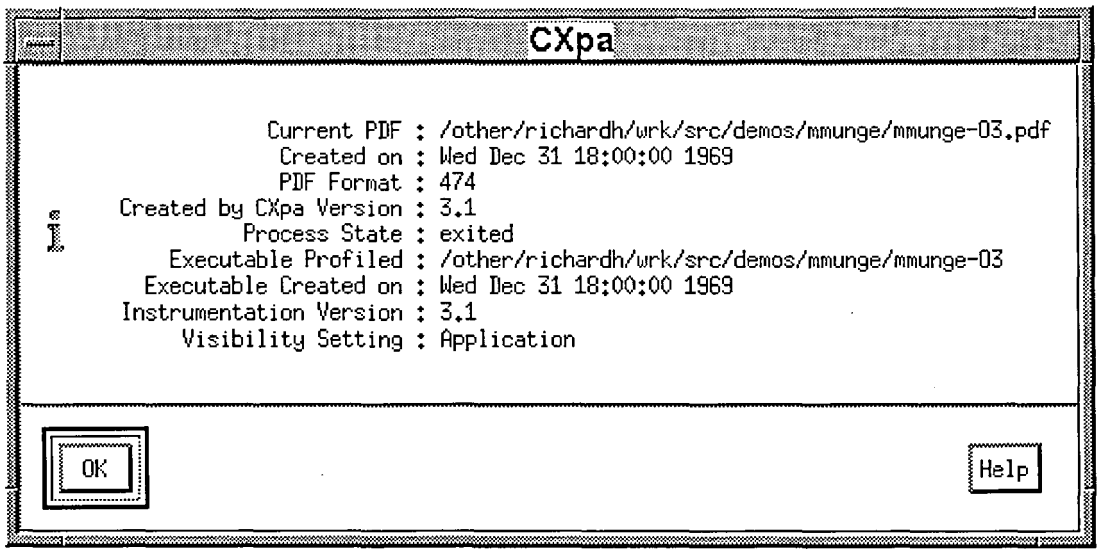
Context

To open the Info Executable dialog, select Executable from the Info menu in the Executable Manager window.

Related Windows

Executable Manager window
Info Session dialog
Info PDF dialog
New PDF dialog

Info PDF dialog



Description

The Info PDF dialog displays information on the currently selected performance data file (PDF).

Fields

Name	Meaning
Current PDF	Name of the current performance data file (PDF).
Created on	Date and time that the PDF was created.
PDF Format	Format version number of the PDF.
Created by CXpa Version	Version of CXpa that created the PDF.
Process State	Process state recorded in the PDF.
Executable Profiled	Name of the executable being profiled.
Executable Created on	Date that the executable was created.

Info PDF dialog

Instrumentation Version	Version of the CXpa profiling routines (cxpamon.o) linked with the executable when the PDF was created.

Buttons

Name	Action
OK	Closes this dialog.

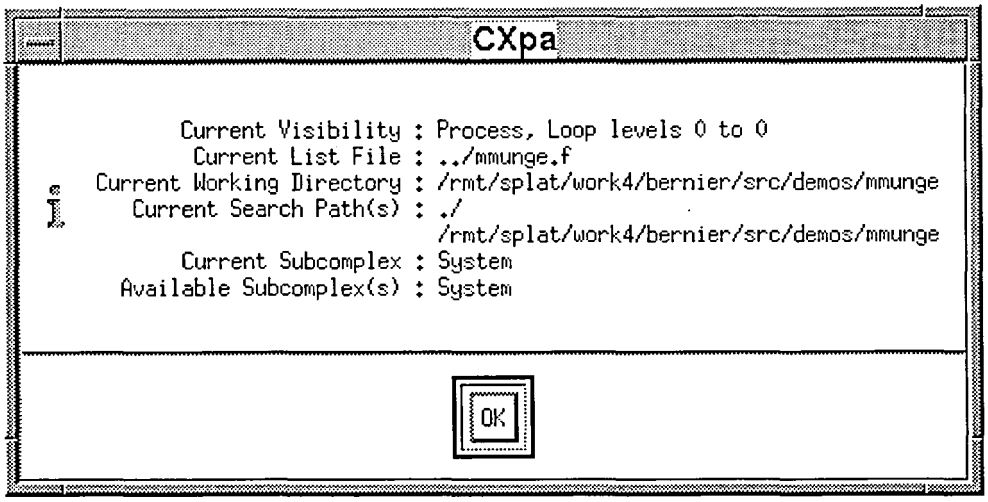
Context

To open the Info PDF dialog, select PDF from the Info menu in the Executable Manager or Analysis Control window.

Related Windows

Analysis Control window
Executable Manager window
Info Executable dialog
Info Session dialog
New PDF dialog

Info Session dialog



Description

The Info Session dialog displays information about your CXpa session.

Fields

Name	Meaning
Current Visibility	Current settings for level of detail in reporting (processes or threads) and loop nesting levels.
Current List File	Name of the file used when you display the Source Code window.
Current Working Directory	Current directory.
Current Search Path(s)	List of directories CXpa uses to locate source files to display in the Source Code window.
Current Subcomplex	Name of the subcomplex that your process is set to run on.
Available Subcomplex(s)	List of available subcomplexes on your system.

Info Session dialog

Buttons

Name	Action
OK	Closes this dialog.

Context

To open the Info Session dialog, select Session from the Info menu in the Executable Manager or Analysis Control window.

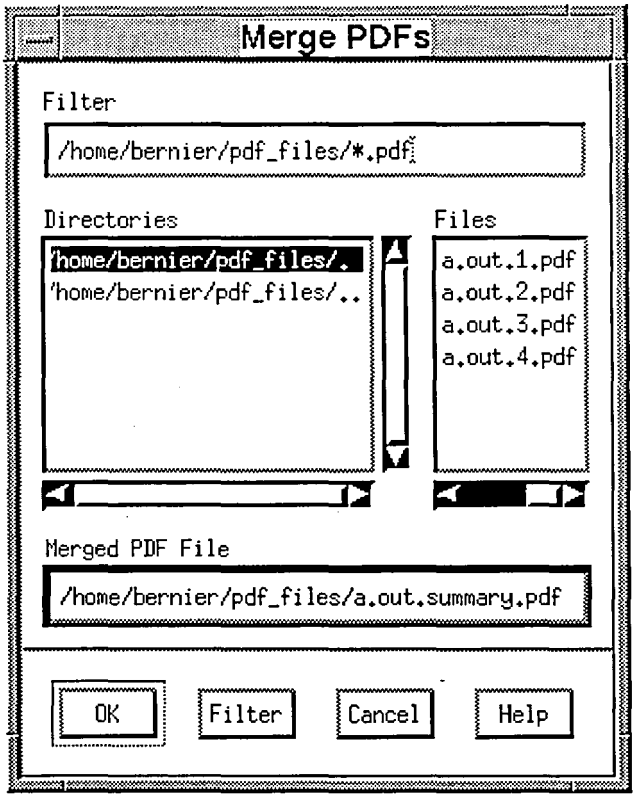
Related Windows

Analysis Control window
Executable Manager window
Info Executable dialog
Info PDF dialog
Profile Selection dialog
Subcomplex Selection dialog

Related Commands

`info`
`set visibility`

Merge PDFs dialog



Description

Use the Merge PDFs dialog to combine profiling data from multiple PDF files generated from the same executable into a single PDF file for analysis.

The PDF files to be merged must be generated from the same executable. In GUI mode, CXpa uses the executable file associated with the first PDF file specified in the Open PDF list in the Analysis Control window as the base executable.

Use the merge feature when analyzing CXpa profiling data generated from MPI (message-passing interface) or PVM (parallel virtual machine) applications.

To merge multiple PDF files:

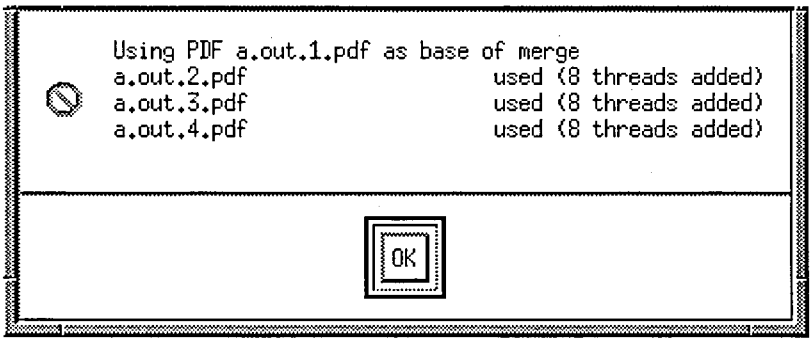
1. Specify the name of the merged PDF file in the Merged PDF file field.
2. Choose OK.

   CXpa merges all PDF files listed in the Open PDF list in the Analysis Control window into a single PDF file, using the file name specified in the Merged PDF file field, then closes the Merge PDFs dialog.

   If an invalid PDF is encountered, an error message is displayed, and CXpa continues to merge the data from valid PDF files.

   When the merge is complete, CXpa displays a dialog that lists the name of the base PDF file associated with the new, merged PDF and the name of each PDF file merged. The base PDF file specifies the executable file associated with the merged PDF.

   For example:



   The name of the new PDF file is displayed at the bottom of the list, is highlighted, and becomes the active PDF.

3. To analyze the profiling data in the new PDF file, press one of the buttons at the bottom of the Analysis Control window (Create 2D Profile, Create 3D Profile, or Create Report).

Refer to the "Profiling MPI and PVM applications with CXpa" section of this book for information about merging PDF files generated from PVM or MPI applications.

Fields

Heading	Meaning
Filter	Displays a search pattern (usually a path and a search pattern containing wildcards) that is applied to the Files and Directories lists when you use the Filter button.

Directories	Lists all subdirectories in the directory specified in the Filter field. Click once on a directory name in this list to insert it into the path in the Filter field. Double-click on a directory name in this list to insert the directory name into the path in the Filter field, search for files that match the filter search pattern, and display them in the Files list.
Files	Lists all files and/or subdirectories in the directory that match the pattern in the Filter field. Highlight the name of a file in this list to insert it into the Merged PDF File field.
Merged PDF File	Contains the full path name of the PDF file that will contain the merged profiling data.

Buttons

Name	Action
OK	Merges the data from the files in the PDF list to the file specified in the PDF File field.
Filter	Searches the directory in the Filter field for files that match the filter search pattern and displays them in the Files list.
Cancel	Closes the dialog without creating a new PDF file.
Help	Displays a help page for this dialog.

Context

To open a Merge PDFs dialog, select Merge from the File menu in the Analysis Control window.

Related Concepts

Analyzing PDFs only
Performance data files
Profiling MPI and PVM applications with CXpa
Using pre-instrumented executables

Related Windows

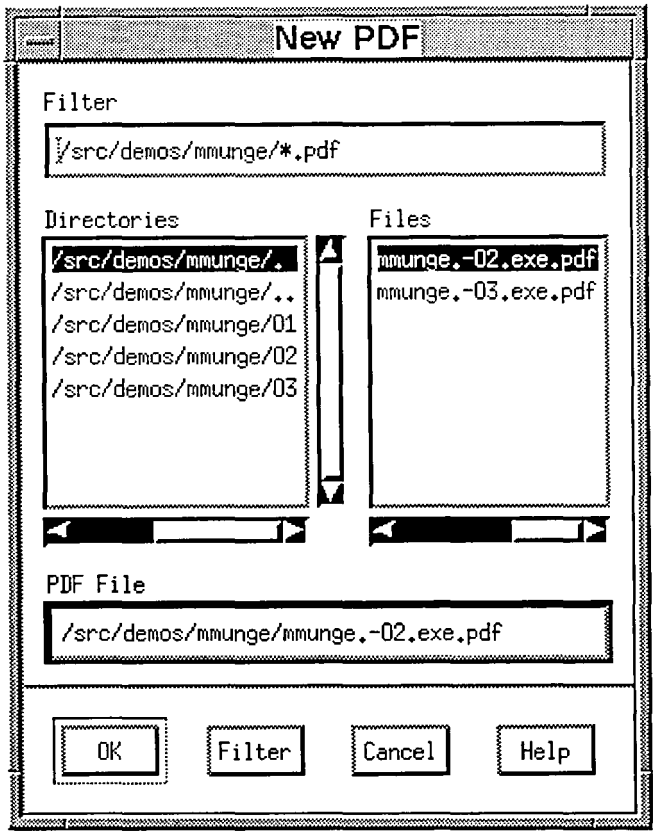
Analysis Control window Open PDF dialog

Related Commands

`merge`

Merge PDFs dialog

New PDF dialog



Description

Use the New PDF dialog to specify the name of the performance data file (PDF). A PDF is a file in which CXpa stores performance data for a single run of your program. During analysis, CXpa uses the data in a PDF to create 2D and 3D Profile graphs and textual performance reports.

If you have invoked CXpa with the name of an executable file, when you run your program, CXpa overwrites all of the data in the PDF. If you do not want this to occur, you must change the name of the PDF between runs of your program using the New PDF dialog.

If the file you specify does not exist, CXpa creates it. If you do not set the PDF name, CXpa uses the default PDF name of *<executable>*.pdf, where *<executable>* is the name of the executable file for your program.

Specifying a new PDF

Use any of the following methods to specify or select a new PDF file name:

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

Fields

Heading	Meaning
Filter	Displays a search pattern (usually a path and a search pattern containing wildcards) that is applied to the Files and Directories lists when you use the Filter button.
Directories	Lists all subdirectories in the directory specified in the Filter field. Click once on a directory name in this list to insert it into the path in the Filter field. Double-click on a directory name in this list to insert the directory name into the path in the Filter field, search for files that match the filter search pattern, and display them in the Files list.
Files	Lists all files and/or subdirectories in the directory that match the pattern in the Filter field. Click on the name of a file in this list to insert it into the PDF File field.
PDF File	Contains name of the current PDF. In exeuction mode, collected profiling data is placed in this file as the program runs. In analysis mode, graphs and reports are generated from the data in this PDF.

Buttons

Name	Action
OK	Accepts the file name in the PDF selection field as the new PDF and closes the dialog.

Filter	Searches the directory in the Filter field for files that match the filter search pattern and displays them in the Files list.
Cancel	Closes this dialog without selecting a new PDF.
Help	Displays a help page for this dialog.

Context

To open a New PDF dialog, select New PDF from the File menu in the Executable Manager window.

Use the New PDF dialog when you do not want to use the default PDF name of *<executable>*.pdf.

Related Windows

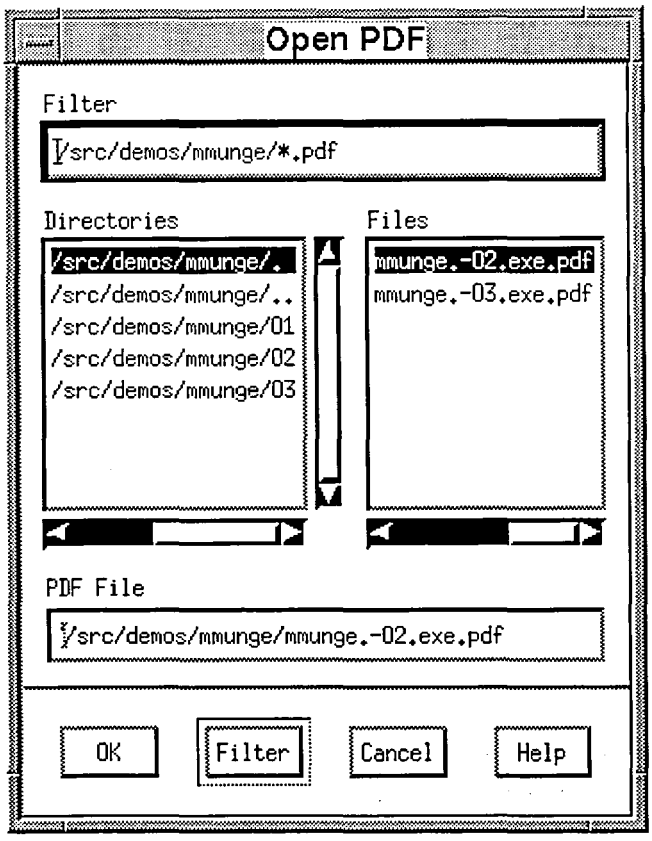
Analysis Control window
Executable Manager window
Filter Profile dialog
Merge PDFs dialog
Open PDF dialog

Related Topics

Analyzing PDFs only
Performance data files

New PDF dialog

Open PDF dialog



Description

The Open PDF dialog allows you to add a PDF (performance data file) to the PDF list in the Analysis Control window.

A PDF is a file that CXpa uses to store performance data for a single run of your program. During analysis, CXpa uses the data in the PDF to create performance graphs and reports.

You can select PDFs created in a previous CXpa session, including PDFs created on other architectures or PDFs created from different executables.

Selecting a file

You can select a PDF to open using any one of the following methods:

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

Fields

Heading	Meaning
Filter	Displays a search pattern (usually a path and a search pattern containing wildcards) that is applied to the Files and Directories lists when you use the Filter button.
Directories	Lists all subdirectories in the directory specified in the Filter field. Click once on a directory name in this list to insert it into the path in the Filter field. Double-click on a directory name in this list to insert the directory name into the path in the Filter field, search for files that match the filter search pattern, and display them in the Files list.
Files	Lists all files and/or subdirectories that match the search pattern in the Filter field. Click on the name of a file in this list to insert it into the PDF File field.
PDF file	Specifies the name of the PDF file to open.

Buttons

Name	Action
OK	Adds the file name in the PDF File field to the PDF list in the Analysis Control window.
Filter	Searches the directory in the Filter field for files that match the filter search pattern and displays them in the Files list.
Cancel	Closes this dialog without opening a new PDF file.
Help	Displays a help page for this dialog.

Open PDF dialog

Context

To display the Open PDF dialog, select Open from the File menu in the Analysis Control window.

Related Windows

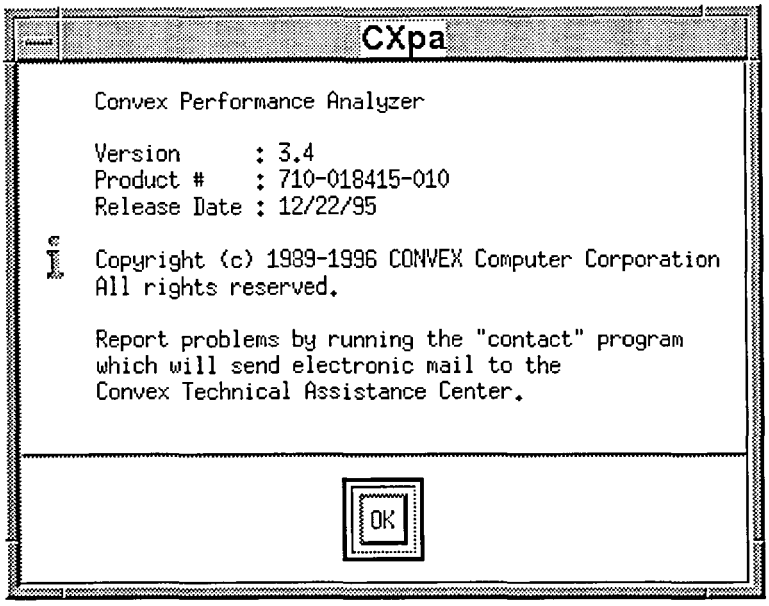
Analysis Control window
Filter Profile dialog
Info PDF dialog
Merge PDFs dialog
New PDF dialog

Related Topics

Analyzing PDFs only
Performance data files

Open PDF dialog

Product Information dialog



Description

The Product Information dialog displays release information:

- CXpa's version number
- Release date of this version of CXpa
- CXpa product part number
- Copyright information
- Information about using the `contact` utility to report problems

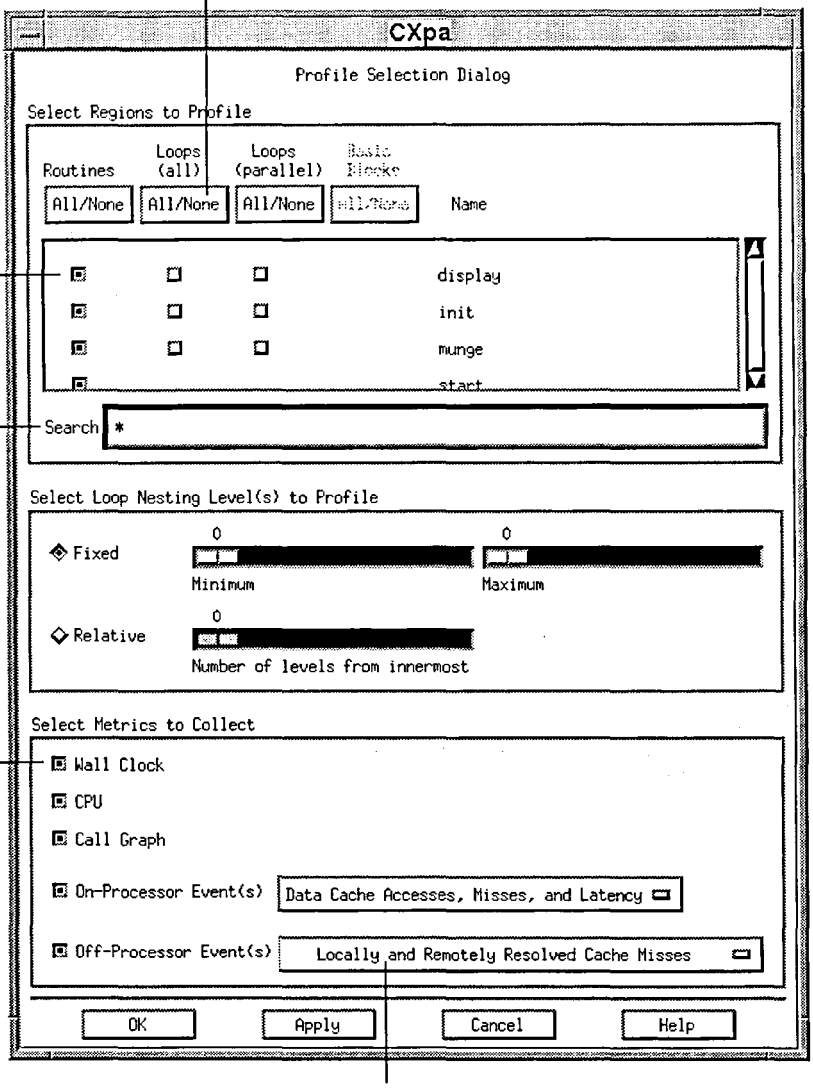
Context

To open a Product Information dialog, select Product Information from the Help menu on any CXpa window.

Product Information dialog

Profile Selection dialog

Use All/None buttons to select or deselect all routines containing the corresponding source code region.



Select/deselect regions to profile in specific routines.

Enter a routine name to search for (you can use * or ? wildcards).

Specify a loop nesting level setting to control the number of loops selected for profiling.

Select/deselect wall clock time, CPU time, call graph, and event(s) metrics for collection.

Use event selection option menu(s) to select the type of events to collect for a single run of your program. These menus are enabled only when On-Processor or Off-Processor Event(s) are selected. Available events vary according to machine architecture.

Description

Use the Profile Selection dialog to select or deselect source code regions for profiling and to specify the types of metrics (including events) to collect for the these regions during profiling.

You do not have to recompile your program to select or deselect regions for profiling.

You can select/deselect different types of source code regions in all routines or limit region and metric selection to specific routines.

The first time you profile your program with CXpa, use the default region and metrics selections. The default selections collect CPU time and wall clock time metrics for all instrumented routines in your program.

Region selection should proceed from coarse-grained (all routines) to fine-grained (loops or parallel loops in specific routines) as code regions that exhibit performance problems are identified. You should collect only the metrics you are interested in per program run (for example, do not enable event collection for a run unless you really need to examine cache performance or instruction metrics).

Refer to the "Profiling strategy" section of this book for more information about profiling intrusion, and a step-by-step strategy for profiling.

NOTE: Selecting all source code regions in all routines for profiling is not recommended, due to increased profiling time and the amount of profiling intrusion that may be incurred.

NOTE: Profiling time increases with the number of regions and metrics selected for profiling. In general, selecting loop regions at all nesting levels for profiling increases execution time more than selecting routine regions. This is because the number of sampled data points in the loop nest grows by a factor of 2 x the number of iterations with each level of nesting.

Selecting source code regions to profile

The upper panel of the Profile Selection dialog (Select Regions to Profile) is where you select the types of source code regions you want to profile and specify a set of routines that contain these regions to profile during a specific run of your program. You can specify either all routines or a subset of routines. The metrics you select will only be collected at these regions in the specified set of routines.

Depending on how you compiled your program, three types of source code regions can be selected for profiling:

- Routines
- All loops (including parallel loops)
- Parallel loops only—parallel loops created by Exemplar compilers at optimization level `+O3 +Oparallel`

NOTE: Only routine regions can be profiled for object files and archive libraries instrumented for profiling with the `cxoi` utility.

Only types of regions that actually appear in your program can be selected. For example, if none of the routines in your program contain loops that were parallelized by the compiler, parallel loop regions will not be selectable.

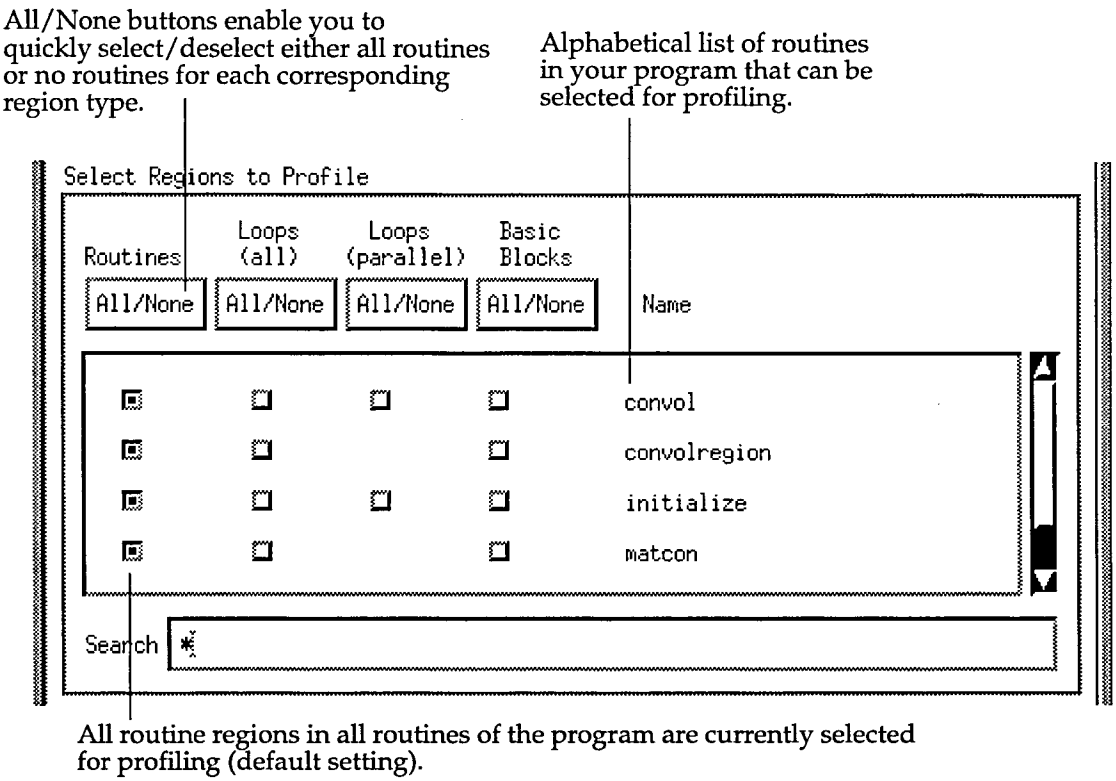
The routines in your program that can be selected for profiling are displayed in alphabetical order, and buttons are displayed opposite each routine name. If a button for a particular region type is not displayed for a routine, it means that no regions of that type can be profiled for that routine. For example, loops are not available for profiling unless the routine is compiled with a Exemplar compiler at optimization level `+O2` or `+O3` with the `+pa` option.

If your program contains a large number of routines, you can:

- Use the scrollbar to navigate the routine list.
- Type the name of a routine in the Search field, then press **RETURN** to scroll the routine list so that the desired routine is displayed at the top of the list.

Default setting—Selecting all routines for routine-level profiling

The default region selection setting, which selects all instrumented routine regions for profiling at the routine level, is shown in the following figure. This is the recommended setting for the first time you run your program under CXpa and will identify the routines that take the longest to execute. Because loops are not selected for profiling, loop nesting level settings are ignored.

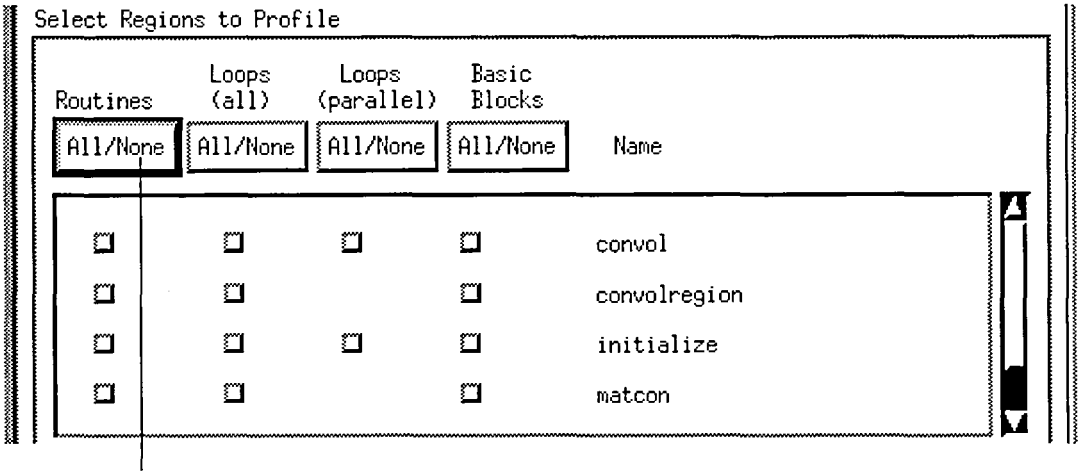


If you have changed the settings, but want to return the settings to this default, use the All/None buttons at the top of the region columns to select/deselect other types of source code regions until all routines are again selected.

Selecting types of source code regions in specific routines

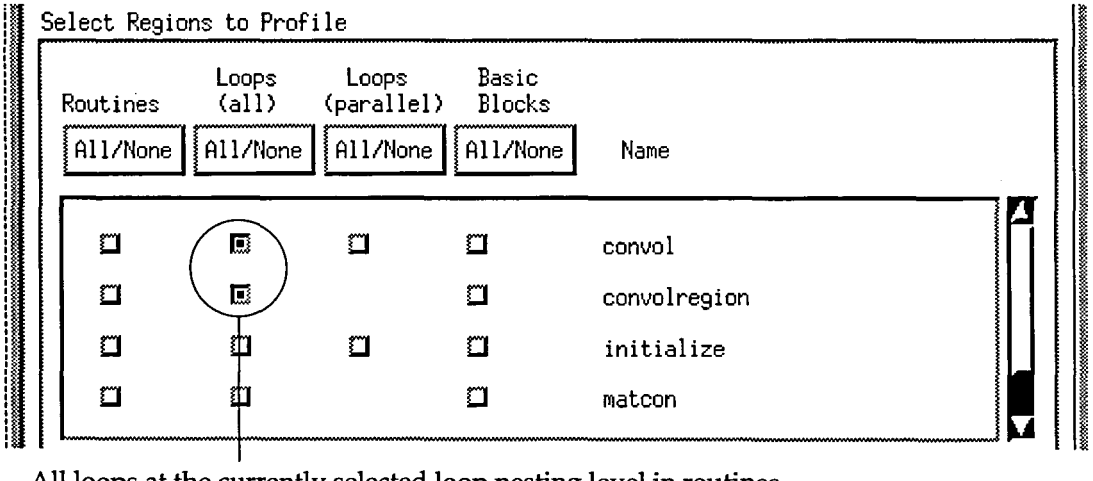
Once you have determined which routines take the longest time to execute, you will want to profile other types of source code regions (for example, loops) within those routines to further isolate performance problems. To select region types in specific routines:

1. Use the All/None buttons at the top of each region column to deselect all routines and loops.



All source code regions in all routines are deselected for profiling.

2. Highlight the button corresponding to the desired region type opposite the routine you want to profile. The following figure shows how to select all loops at the currently specified loop nesting level for profiling in routines `convol` and `convolregion` only.



All loops at the currently selected loop nesting level in routines `convol` and `convolregion` only are selected for profiling.

Do not select all available source code regions for every routine (that is, do not highlight every button). The amount of profiling intrusion introduced by this can produce invalid results.

If you selected loops for profiling (as in the above example), the current loop nesting level setting applies to all loops. If no loops in a routine fall within the specified loop nesting level range, then no loops in that routine are selected for profiling.

The loop nesting level setting is displayed in the Select Loop Nesting Level(s) to Profile panel in the Profile Selection dialog. Refer to the next section for information about selecting loop nesting levels.

Selecting loop nesting levels

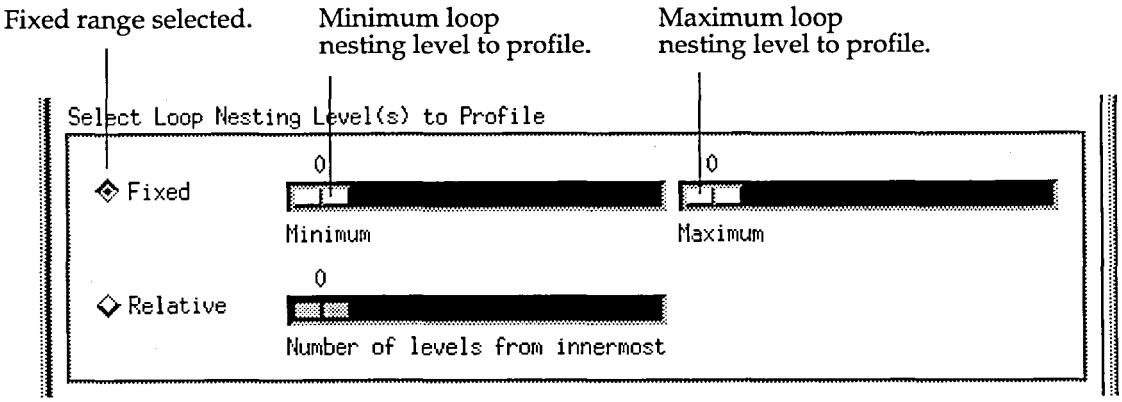
If you have chosen to profile loops, you can optionally specify either a fixed range of loop nesting levels to profile or the number of loop nesting levels to profile relative to each loop nest's innermost level.

The loop nesting level setting applies to all loops selected for profiling and is only active when loops are selected for profiling.

CXpa determines the number of loop nesting levels in your program and sets the maximum loop nesting levels and the maximum number of levels from the innermost loop appropriately. These nesting levels correspond to the loops that are created by the compiler and may not correspond directly to your original source code due to optimizations performed.

Default loop nesting level range settings

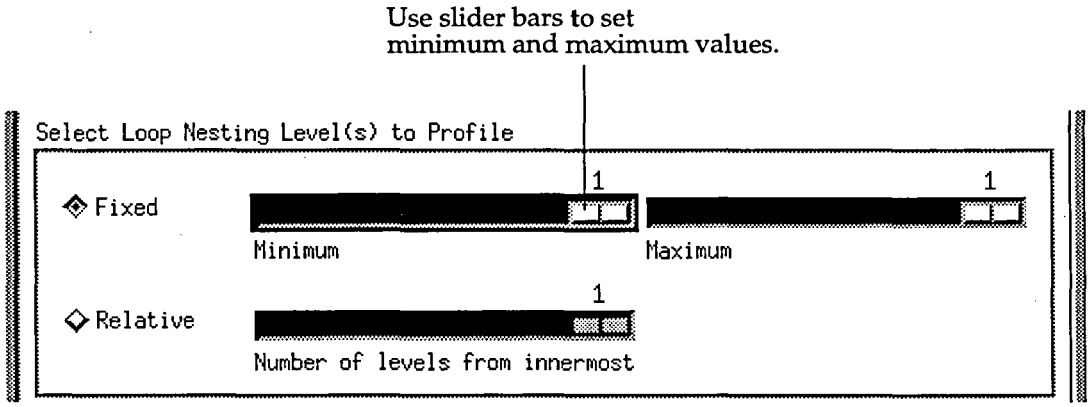
The first time you profile loops in your program, use the default setting for loop nesting levels (shown in the following figure). The default setting specifies a fixed loop nesting level range with a minimum of 0 and a maximum of 0 (after optimization). This means that all loops with a nesting level of 0 after optimization (outermost loops) are selected for profiling. This will minimize profiling intrusion.



Default loop nesting level setting—Selects only loops at nesting level 0 (outermost loops) for profiling.

Specifying a fixed loop nesting level range

On subsequent runs of your program, you can select different sections or "slices" of the loops within your program for profiling. When specifying a fixed ranged of loop nesting levels, you will generally want to set the minimum loop nesting loop level equal to the maximum loop nesting level, as shown in the following figure.



Sample fixed loop nesting level setting—Minimum and maximum nesting levels set to 1 selects only loops at nesting level 1 for profiling.

Specifying the number of relative loop nesting levels

Instead of choosing a fixed range of loop nesting levels for profiling, you can specify the number of loop nesting levels to profile relative to the innermost loop of each loop nest in your program.

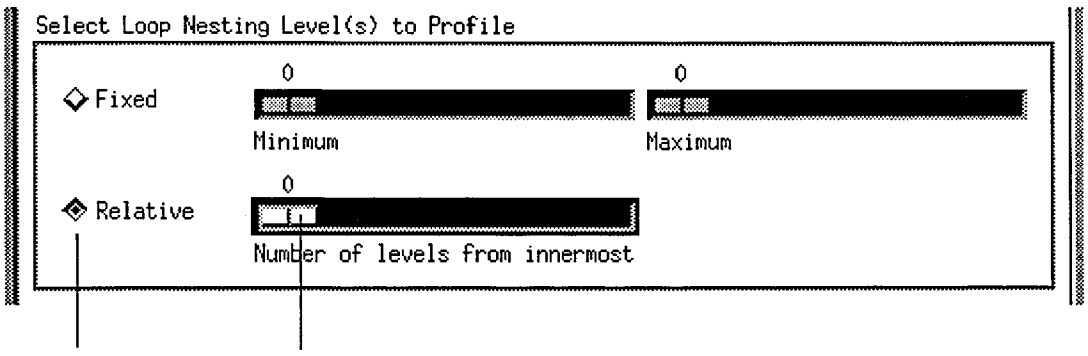
- A relative setting of 0 means that only the loops at the innermost (deepest) level of each loop nest are selected for profiling.
- A relative setting of 1 means that only the loops at the two innermost nesting levels of each loop nest are selected for profiling.

  For example, if the innermost nesting level of a loop nest is 4 and a relative loop nesting level of 1 is specified, the loops at nesting levels 3 and 4 of that loop nest are selected for profiling.
- A maximum setting (setting the slider bar as far to the right as it will go) is equivalent to selecting all loops at all loop nesting levels. This is not recommended, because of the amount of profiling intrusion that can be introduced.

When a relative loop nesting level is specified, all loops that are not part of a loop nest are also selected for profiling.

To specify a relative setting for loop nesting levels:

1. Highlight the Relative button in the Select Loop Nesting Level(s) to Profile panel.
2. Use the slider bars to select the number of loop nesting levels to profile relative to the innermost loops in your program, as shown in the following figure.



Relative loop nesting level selected.

A relative loop nesting level setting of 0 selects all loops at the innermost nesting level of each loop nest for profiling, along with any loops that are not part of a loop nest.

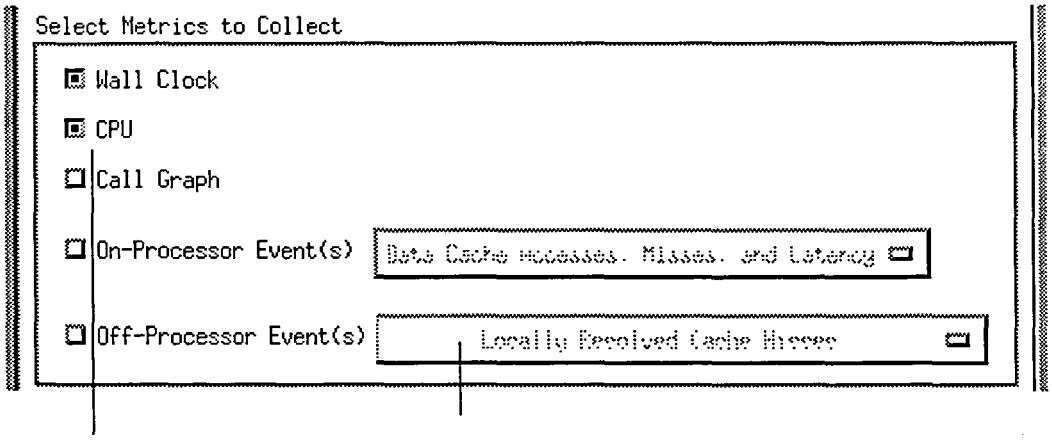
Selecting metrics to collect

Once you have specified the regions to profile, you can specify the metrics you want to collect at these regions using the Select Metrics to Collect section of the Profile Selection dialog.

CXpa collects these metrics at the regions of your program that are selected for profiling. You can choose to collect:

- Wall clock time (default)
- CPU time (default)
- Call Graph. If you select Call Graph, make sure that all routine regions are selected for profiling and that CPU and wall clock time metrics are collected.
- Events

By default, wall clock time and CPU time metrics are selected, as shown in the following figure.



By default, wall clock time and CPU time metrics are selected.

Event selection option menus are disabled when events are not selected for collection.

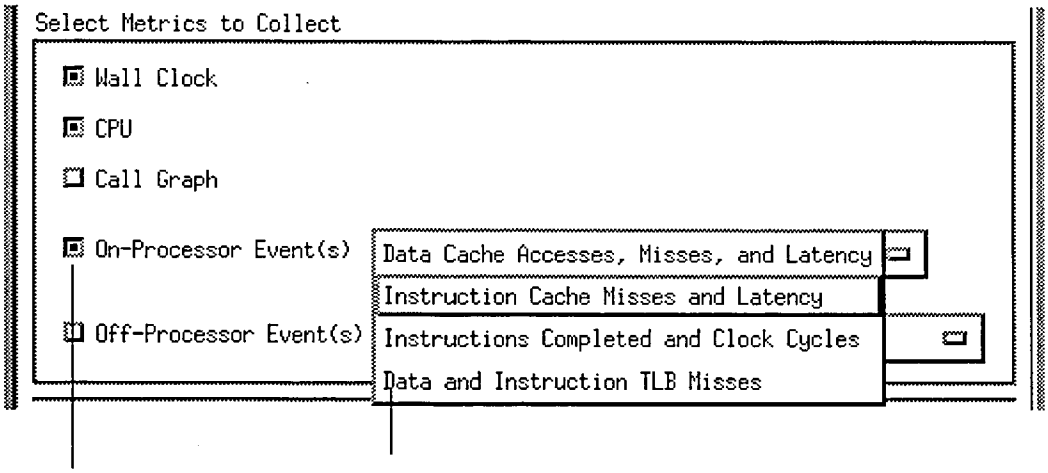
To specify the type of metrics to collect during profiling, perform the following steps:

1. Specify the metrics you want to collect using the buttons in the Select Metrics to Collect section of the Profile Selection dialog.

   If you select On-Processor or Off-Processor Events, you must specify the type of event or events you want to collect as described in step 2. Otherwise, continue with step 3.

When you select On-Processor or Off-Processor Event(s), the Event(s) option menus are enabled, and the default set of event metrics is selected.

2. Click and hold down the left mouse button on the Event(s) option menu to display a list of available event types (shown in the following figure). While holding down the left mouse button, position the mouse cursor over the desired event type, then release the mouse button.



Event collection is enabled.

Events(s) option menu now displays a list of hardware events that can be collected on your system. These vary according to machine architecture.

The number and type of events you can select vary according to machine architecture:

- Refer to the "Introducing metrics" online help topic or section in this book for a discussion of available event metrics for Exemplar S2000, Exemplar X2000, SPP1200 Series, and SPP1600 Series systems.
- Refer to the "Selecting Metrics in GUI mode" online help topic or section in this book for more information about selecting events.

3. Choose OK.

Select Regions panel

Select the types of source code regions you want to profile and specify a set of routines that contain these regions to profile during a specific run of your program.

Label	Meaning
Routines	Selects routine regions in specified routines for profiling.
Loops (all)	Selects all loop regions (including parallel loops) in specified routines for profiling.
Loops (parallel)	Selects only parallel loops in specified routines. These are parallel loops created by Exemplar compilers at optimization level `+O3 +Oparallel`.
All/None buttons	Selects/deselects all routines in your program that contain the indicated type of source code region (routines, all loops, or parallel loops only).
Name	Lists names of routines in your program that contain regions that can be selected for profiling. These are listed in alphabetical order.
Search	Allows you to search for a particular routine. Expressions with no wildcards will search for literal matches. Available wildcards include question marks (?) to match a single character and asterisks (*) to match multiple characters. When you press **RETURN**, CXpa executes the search, and, if a match is found, scrolls the list so that the first matching routine name is at the top.

Select Loop Nesting Level(s) panel

Specify either fixed range of loop nesting levels or a number of loop nesting levels relative to the innermost (deepest) loop in each nest. This setting applies to all loop nests in your program and is only active when loops are selected for profiling.

Fixed	Use the slider bars for each field to specify a range of loop nesting levels to profile by setting maximum and minimum values.
Relative	Use the slider bar to specify the number of loop nesting levels from the innermost loop to profile.

Select Metrics panel

Choose metrics to collect for the regions of your program selected for profiling.

Wall clock	Enables/disables collection of wall clock time.
CPU	Enables/disables collection of CPU time.
Call Graph	Enables/disables collection of dynamic call graph information. Make sure that all routines in your program are selected for profiling if this option is enabled.

On-Processor Event Counter(s)
(SPP1200 and SPP1600 Series systems only)

Off-Processor Event Counter(s)
(Exemplar S2000, Exemplar X2000, and SPP1600 Series systems only)

Displays an option menu where you can select the type of hardware event to collect. The number and types of events you can select differ among Exemplar S2000, Exemplar X2000, SPP1200 Series, and SPP1600 Series systems.

Refer to the "Introducing Metrics" and "Selecting metrics in GUI mode" online help topics or sections in this book for descriptions of on-processor and off-processor hardware event metrics available on each architecture.

Profile Selection dialog

Buttons

Name	Action
OK	Applies the changes and closes the dialog.
Apply	Applies the changes without closing the dialog.
Cancel	Closes this dialog without making any changes.
Help	Displays a help page for this dialog.

Context

To open the Profile Selection dialog, use the Profile Selection button in the Executable Manager window.

Related Windows

Analysis Report window

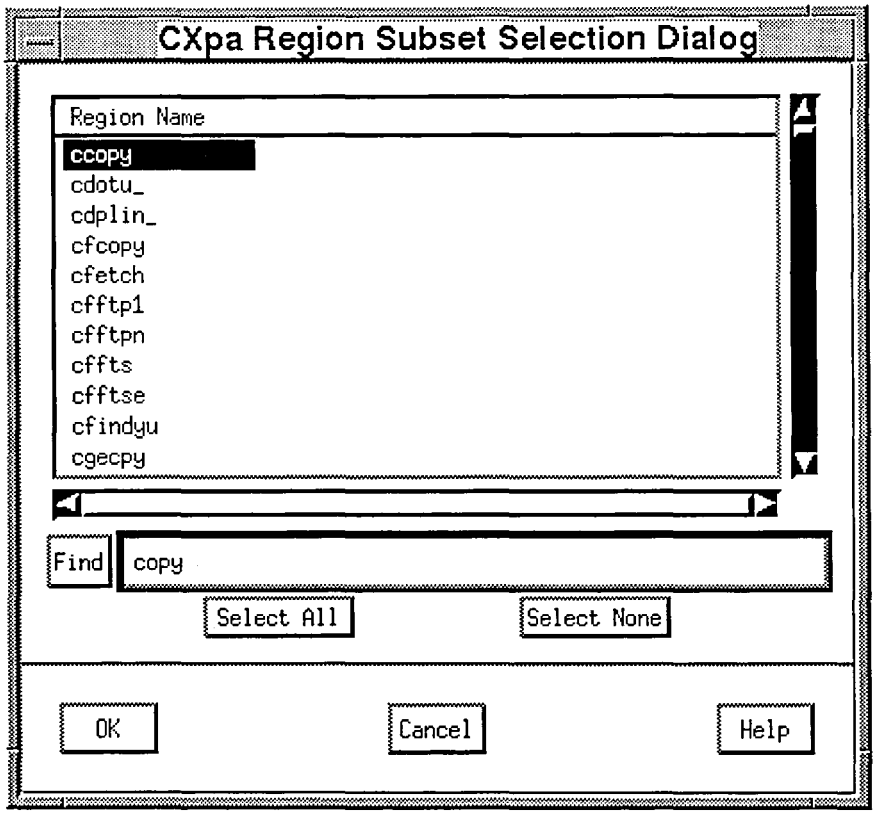
Executable Manager window

Related Topics

Introducing metrics
Introducing source code regions
Profiling strategy
Selecting metrics in GUI mode
Selecting regions in GUI mode
Using pre-instrumented executables

Profile Selection dialog

Region Subset Selection dialog



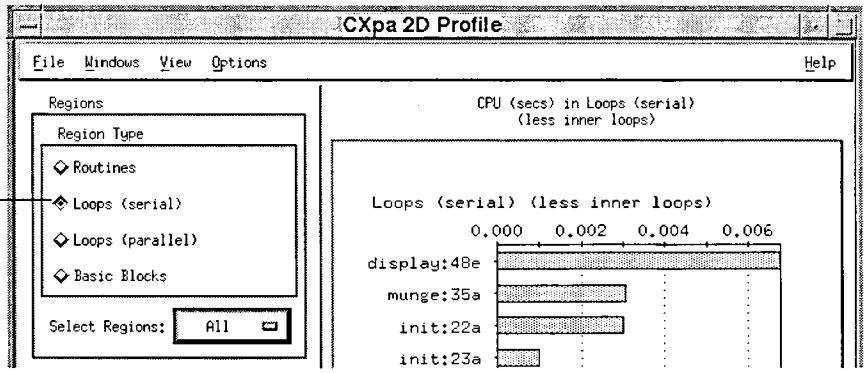
Description

The Region Subset Selection dialog allows you to create a customized 2D graph, 3D graph, or report by specifying a subset of routines that contain regions of the currently selected region type (routines, loops, or parallel loops) for analysis.

To create a customized profile graph or report:

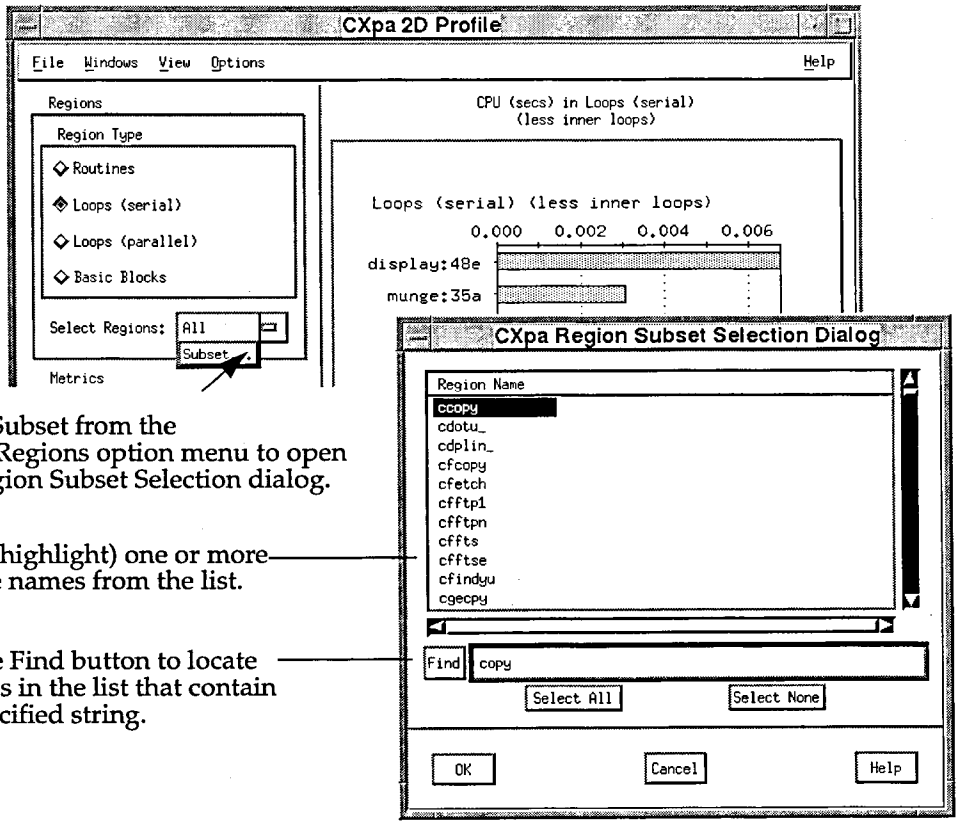
1. In the 2D Profile, 3D Profile, or Analysis Report window, select a region type. You can select only one.

Region Subset Selection dialog



Select a region type.

2. Select Subset from the Select Region option menu to open a Region Subset Selection dialog with a scrolled list of routines containing profiled regions of the selected type.



Select Subset from the Select Regions option menu to open the Region Subset Selection dialog.

Select (highlight) one or more routine names from the list.

Use the Find button to locate routines in the list that contain the specified string.

3. Use the mouse to highlight the routines that you want to include in the graph or report. You can select multiple routines.

   If the program contains a large number of routines, you can specify a string in the Find field to search for, use the Find button to locate the first routine name containing that string. CXpa scrolls the routine list so that the matching routine name is placed at the top of the list.

   Use the Find button again to locate the next routine that contains the specified string and scroll the list so that it is placed at the top of the list, and so on. The search will wrap to the beginning of the list. Be sure to highlight routine names to select them after executing the search.

4. Choose OK to display the new graph or report.

   If you choose OK without selecting any routines, CXpa displays an Info dialog displaying the message "`INFO A70: No regions selected for a customized 2D or 3D profile.`" Close the Info dialog, then either select at least one routine or choose Cancel to dismiss the Region Subset Selection dialog.

Fields

Heading	Meaning
Region name	Contains a scrolled list of routines in your program that contain regions of the currently selected type. Use the mouse to highlight one or more routines in the list to select for graphing or inclusion in text reports.

Buttons

Name	Action
Find	Searches the list of routine names for occurrences of the string specified in the text entry field opposite the Find button. If a match is found, the routine list scrolls so that the first matching routine name is placed at the top of the list (or, if it is near the end of the list, it is visible in the list). Press Find again to locate the next routine that contains the specified string and scroll the list so that it is placed at the top of the list, and so on. The search will wrap to the beginning of the list.
Select all	Selects all routines in the list. Selected routines are highlighted.
Select none	Deselects all routines in the list.

Region Subset Selection dialog

OK	Accepts the changes you have made, closes the dialog, and applies the changes to the graph.
Cancel	Does not apply any of the changes you have made and closes the dialog.
Help	Displays a help page for this dialog.

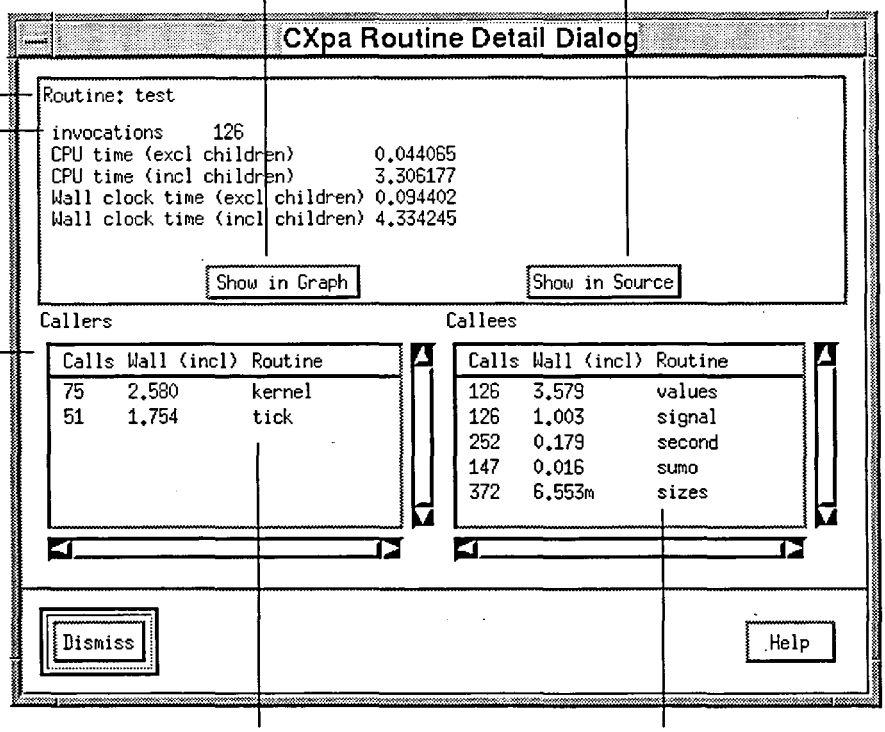
Context

To open a Region Subset Selection dialog, select Subset from the Select Regions option menu in the Regions panel in the 2D Profile, 3D Profile, or Analysis Report window.

Related Windows

2D Profile window
3D Profile window
Analysis Control window
Analysis Report window
Executable Manager window
Profile Selection dialog

Routine Detail dialog



Description

From the Routine Detail dialog you can:

- View inclusive and exclusive CPU time and wall clock time metrics collected for the routine currently selected in the Call Graph window.

- View a list of routines that were called by the currently selected routine (Callees).

  The list of callees is sorted by the value of the ranking metric that is currently selected in the Call Graph window. The routine that contributed the highest percentage of that metric to the total for the selected routine is listed first. Call counts and the value for the current ranking metric for each routine are also displayed.

- View a list of routines that called the currently selected routine (Callers).

  Call counts and the amount of the total value of the current ranking metric attributed to each calling routine are also displayed.

- Select any routine in the Callers or Callees list to make it the currently selected routine. Its location is highlighted in the Call Graph window, and the Routine Detail dialog updates to reflect the new selection.

  If the selected routine is in a collapsed node (one indicated by an asterisk (*) in the Call Graph window), the asterisk representing its node is highlighted. To expand the node so that the selected routine is showing in the Call Graph window, use the Show in Graph button.

- Use the Show in Source button to view source code associated with the routine currently selected in the Call Graph window.

Only one Routine Detail dialog is displayed per Call Graph window. As you select new routines to display by clicking on them in the Call Graph window or by using the Find Routine dialog, the display of information in the Routine Detail dialog updates to reflect the current choice.

Fields

Heading	Meaning
Routine	Displays the name of the currently selected routine, total number of invocations, and the available metric values for that routine.
Callers	Lists profiled routines in your program that called the currently selected routine (parent routines). The list is sorted by call counts. The number of calls and the value of the current ranking metric attributed to each routine are also displayed.
Callees	Lists profiled routines in your program that were called by the currently selected routine (child routines). The list of callees is sorted by the value of the ranking metric that is currently selected in the Call Graph window.

Routine Detail dialog

The routine that contributed the highest percentage of that metric to the total is listed first. The number of calls and the value of the current ranking metric attributed to each routine is also displayed.

Buttons

Name	Action
Show in Graph	Highlights the routine in the Call Graph window and, if necessary, scrolls the window so that the currently selected routine is showing. If the selected routine is in a node that is currently collapsed, its node is expanded until the routine is displayed.
Show in Source	Opens a Source Code window displaying source code associated with the currently selected routine.
Dismiss	Closes the dialog.
Help	Displays a help page for this dialog.

Context

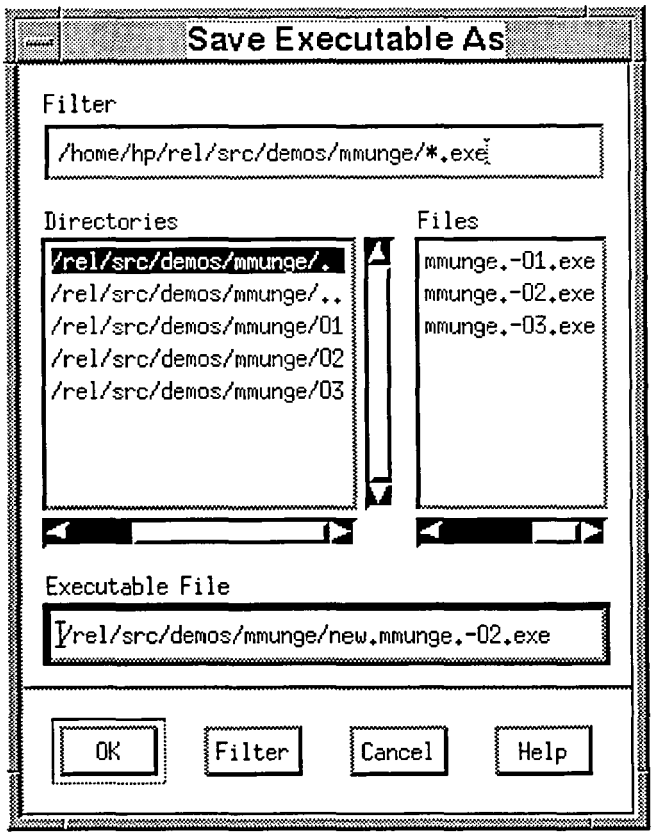
To open a Routine Detail dialog, click on the name of a routine in the Call Graph window.

Related Windows

Call Graph window
Find Routine dialog
Thread Selection dialog

Routine Detail dialog

Save Executable As dialog



Description

Use the Save Executable As dialog to save an executable file and its current region and metric selection settings (instrumentation) to a new executable file. This is referred to as *pre-instrumenting* an executable. Use pre-instrumented executables to:

- Profile with CXpa in environments that do not support CXpa's controlling a child process.
- Profile applications in conjunction with tools such as MPI or PVM that replicate processes or with applications where a driver program or script starts the process.

Refer to the "Profiling MPI and PVM applications with CXpa" online help topic or section of this book for information about using pre-instrumented executables when profiling message-passing applications with CXpa.

- Maintain separate copies of an executable with different regions and metrics selected for profiling. This makes it easier to generate multiple performance data files (PDFs) for comparison and analysis.
- Profile applications run with the `mpa` utility. Refer to the "CXpa and the `mpa` utility" online help topic or section of this book for more information.

The new executable file is an exact copy of the executable being profiled, except that it contains the current source code region and metric selections. All source code correlation is maintained. The size and permissions of the executable do not change, but the time stamp on the executable does.

You can then run the new executable file outside the control of CXpa and collect profiling data in a PDF file for later analysis. You can also profile the new executable in the usual way (that is, by invoking CXpa with the name of the executable and running it under CXpa).

Refer to the "Using pre-instrumented executables" online help topic or section of this book for information about using pre-instrumented executables.

Fields

Heading	Meaning
Filter	Displays a search pattern (usually a path and a search pattern containing wildcards) that is applied to the Files and Directories lists when you use the Filter button.
Directories	Lists all subdirectories in the directory specified in the Filter field. Click once on a directory name in this list to insert it into the path in the Filter field. Double-click on a directory name in this list to insert the directory name into the path in the Filter field, search for files that match the filter search pattern, and display them in the Files list.
Files	Lists all files and/or subdirectories in the that match the pattern in the Filter field. Click on the name of a file in this list to insert it into the Selection field.

Save Executable As dialog

Executable File	Contains the full path name of the file to save the current executable to.

Buttons

Name	Action
OK	Saves the executable file and its current region and metric selection settings (instrumentation) to the file specified in the Executable File field.
Filter	Searches the directory in the Filter field for files that match the filter search pattern and displays them in the Files list.
Cancel	Closes the dialog without saving the executable.
Help	Displays a help page for this dialog.

Context

To open a Save Executable As dialog, select Save As from the File menu in the Executable Manager window.

Related Concepts

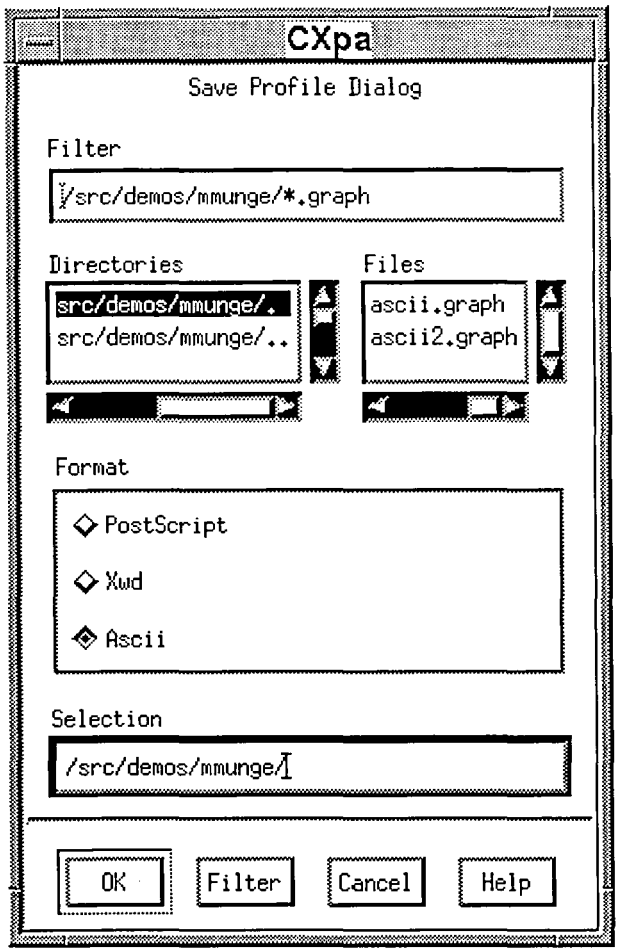
CXpa and the mpa utility
Profiling MPI and PVM applications with CXpa
Using pre-instrumented executables

Related Windows

Executable Manager window Profile Selection dialog

Save Executable As dialog

Save Profile dialog



Description

The Save Profile dialog allows you to save the graph in the 2D Profile or 3D Profile windows to a PostScript, ASCII, or xwd file. Only the area of the graph that is visible in the window is saved to the file. These files are immediately available for use by other utilities that accept these formats, as shown in the following table:

File format	Can be used with shell command	Can be imported to
PostScript	`lp, lpr`	Document publishing or word processing applications
xwd	`xwud, xpr`	Document publishing or word processing applications
ASCII	`lp, lpr`	Spreadsheet or graphic applications

Use one of the following methods to specify the name of a file to save 2D or 3D profile graphs to:

- Double-click on a file name in the Files list.
- Highlight a file name in the Files list and choose OK.
- Type the full path name of the file in the Selection field and choose OK or press **RETURN**.

Fields

Heading	Meaning
Filter	Displays a search pattern (usually a path and a search pattern containing wildcards) that is applied to the Files and Directories lists when you use the Filter button.
Directories	Lists all subdirectories in the directory specified in the Filter field. Click once on a directory name in this list to insert it into the path in the Filter field. Double-click on a directory name in this list to insert the directory name into the path in the Filter field, search for files that match the filter search pattern, and display them in the Files list.
Files	Lists all files and/or subdirectories in the directory that match the pattern in the Filter field. Click on the name of a file in this list to insert it into the Selection field.

Format	Selects a format type—ASCII, PostScript, or xwd.
Selection	Contains the file name to save the profile to.

Buttons

Name	Action
OK	Saves the profile graph in the selected format to the file specified in the Selection field.
Filter	Searches the directory in the Filter field for files that match the filter search pattern and displays them in the Files list.
Cancel	Closes this dialog without making any changes.
Help	Displays a help page for this dialog.

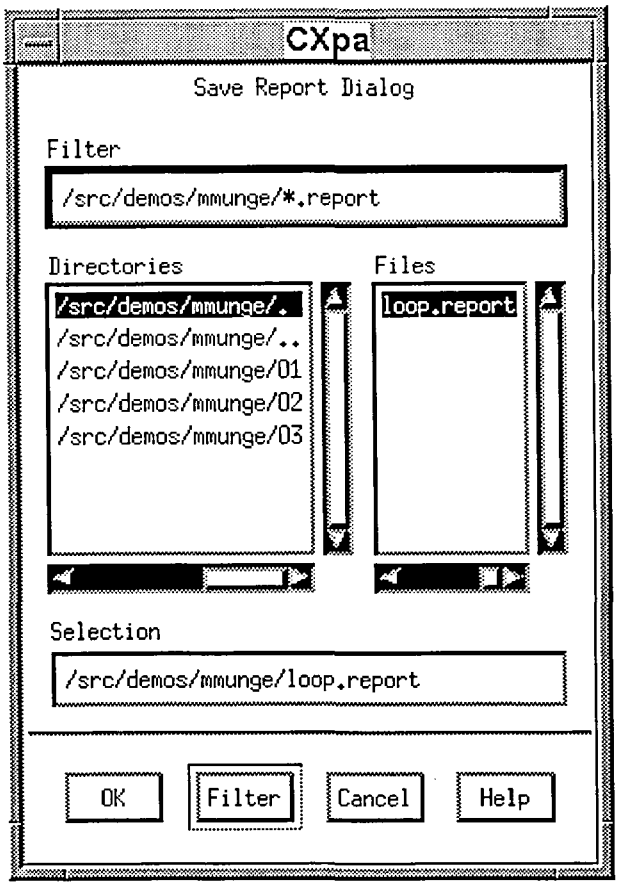
Context

To open a Save Profile dialog, select Save Profile from the File menu in the 2D Profile or 3D Profile windows.

Related Windows

2D Profile window
Filter Profile dialog
3D Profile window
Region Subset Selection dialog

Save Report dialog



Description

The Save Report dialog allows you to save the report in the Analysis Report window to an ASCII file. Use one of the following methods to specify the name of a file to save the report to:

- Double-click on a file name in the Files list.
- Highlight a file name in the Files list and choose OK.
- Type the full path name of the file in the Selection field and choose OK or press **RETURN**.

Save Report dialog

Fields

Heading	Meaning
Filter	Displays a search pattern (usually a path and a search pattern containing wildcards) that is applied to the Files and Directories lists when you use the Filter button.
Directories	Lists all subdirectories in the directory specified in the Filter field. Click once on a directory name in this list to insert it into the path in the Filter field. Double-click on a directory name in this list to insert the directory name into the path in the Filter field, search for files that match the filter search pattern, and display them in the Files list.
Files	Lists all files and/or subdirectories in the directory that match the pattern in the Filter field. Click on the name of a file in this list to insert it into the Selection field.
Selection	Contains the file name to save the report to.

Buttons

Name	Action
OK	Saves the report to the file specified in the Selection field.
Filter	Searches the directory in the Filter field for files that match the filter search pattern and displays them in the Files list.
Cancel	Closes this dialog without making any changes.
Help	Displays a help page for this dialog.

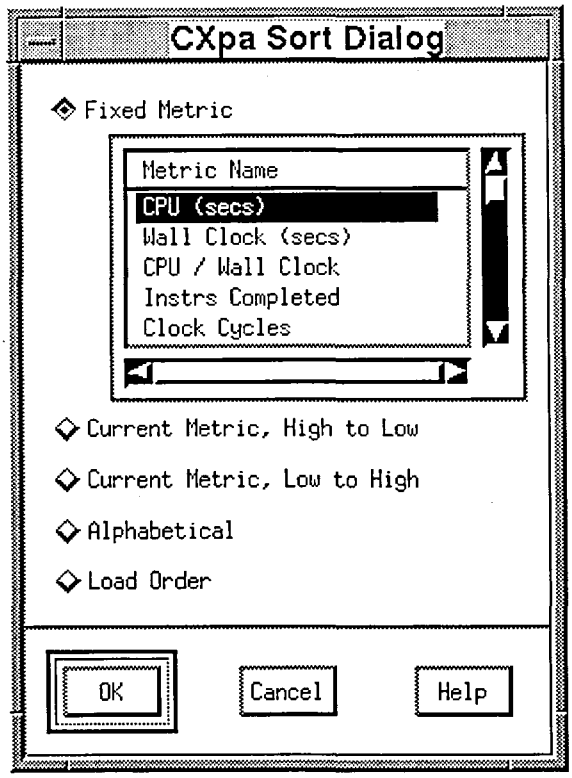
Context

To open a Save Report dialog, select Save Report from the File menu in the Analysis Report window.

Related Windows

Analysis Report window
Filter Report dialog
Region Subset Selection dialog

Sort dialog



Description

The Sort dialog allows you to sort the regions in the 2D and 3D profile graphs using one of the following methods:

- **Fixed metric**—Sorts the regions in the associated 2D or 3D Profile window from highest to lowest, by the value for the metric you choose from the Metric Name list. The Metric Name list contains the metrics collected for the run of the program that generated the current PDF file.

  As different metrics are selected for graphing in the corresponding 2D or 3D Profile window, the sort order of the regions displayed on the Regions axis remains constant and is based on the value of the fixed metric, from highest to lowest.

- **Current Metric, High to Low**—Sorts regions by the data values for the metric currently selected in the corresponding 2D or 3D Profile window, from highest to lowest. This is the default.
- **Current Metric, Low to High**—Sorts regions by the data values for the metric currently selected in the corresponding 2D or 3D Profile window, from lowest to highest.
- **Alphabetical**—Sorts regions alphabetically, by routine name.
- **Load Order**—Sorts regions in the order that the routines containing them were given to the link editor.

Order

Lists the sorting choices.

Fixed metric	Sorts regions in the associated 2D or 3D Profile graph from highest to lowest, by the data values for the metric that is selected from the Metric Name list.
Current metric, High to Low	Sorts regions in the associated 2D or 3D Profile graph from highest to lowest, by the data values for the currently selected metric in the 2D or 3D Profile window.
Current metric, Low to High	Sorts the regions in the associated 2D or 3D profile graph from lowest to highest, by the data values for the currently selected metric in the 2D or 3D Profile window. This is the default.
Alphabetical	Sorts the regions in the associated 2D or 3D profile graph alphabetically, by routine name.
Load order	Sorts the regions in the associated 2D or 3D profile graph in the order that the routines containing them were given to the link editor.

Buttons

Name	Action
OK	Sorts the regions in the graph in the chosen order and closes this dialog.
Apply	Sorts the regions in the graph in the chosen order without closing the dialog.

Cancel	Closes this dialog without making any changes.
Help	Displays a help page for this dialog.

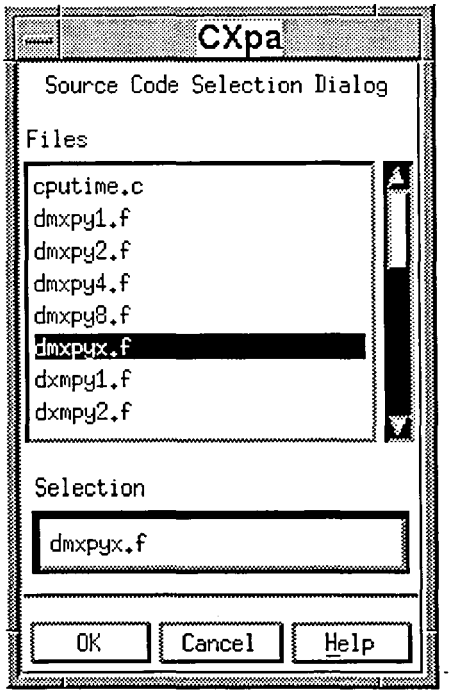
Context

To open a Sort dialog, select Sort from the View menu in the 2D or 3D Profile window.

Related Windows

2D Profile window
3D Profile window
Region Subset Selection dialog
Zoom dialog

Source Code Selection dialog



Description

Use the Source Code Selection dialog to select a different source file to display in the Source Code window.

To change the source file displayed in the Source Code window:

1. Highlight the name of the source file that you want to display. The selected file is displayed in the Selection field.
2. Choose OK. The dialog closes, and the new source file is displayed in the Source Code window.

If CXpa cannot find the source file, change the search path by selecting Source Search Path from the Options menu to open a Source Search Path dialog.

Source Code Selection dialog

NOTE: You cannot enter a full path name in the Selection field. To change paths:

1. **Add the desired directory path in the Source Search Path dialog and choose OK.**
2. **Type the file name in the selection field of the Source Code Selection dialog.**

Fields

Heading	Meaning
Files	Lists all the source files used to create the executable file.
Selection	Lists the file selected for display in the Source Code window.

Buttons

Name	Action
OK	Accepts the file name in the selection field as the new source file to display and closes this dialog.
Cancel	Closes this dialog without making any changes.
Help	Displays a help page for this dialog.

Context

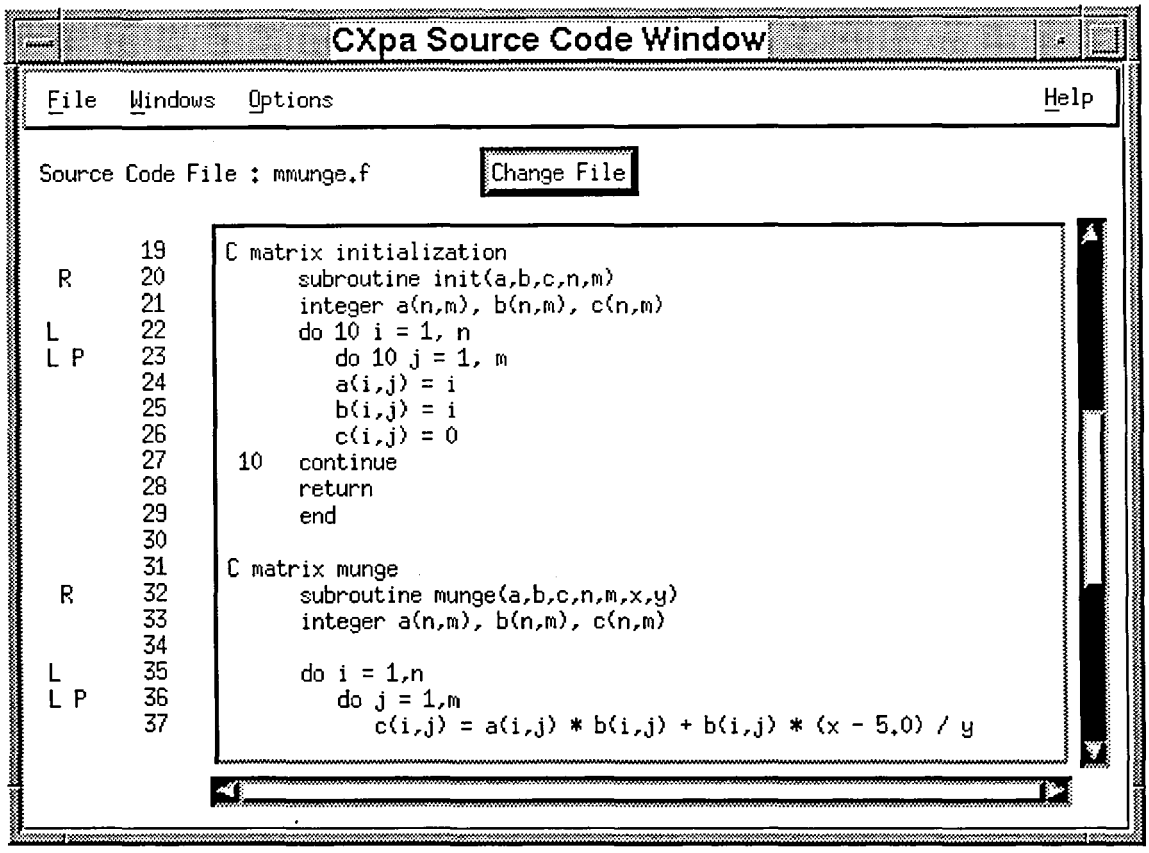
To open a Source Code Selection dialog, use the Change File button on the Source Code window.

Related Windows

Source Search Path dialog

Source Code window

Source Code window



Description

The Source Code window displays the source code for your program. By default, the Source Code window displays the file that contains your main routine. You can open multiple Source Code windows displaying different files in your program or use the Change File button to display a different source file in the current window.

Annotations

To the left of the line numbers, CXpa displays annotations that identify source code regions that can be profiled. Uppercase letters indicate source code regions currently selected for profiling. Lowercase letters indicate source code regions that are not selected for profiling.

- `R`, `r`—Indicates routine regions.
- `L`, `l`—Indicates loop regions.
- `P`, `p`—Indicates parallel loop regions.

The `=>` symbol to the right of a line number indicates the beginning of a section of code that corresponds to a bar in the 2D Profile or 3D Profile window that you clicked to display source code.

Changing source files

Use the following procedure to select a different source file to display:

1. Use the Change File button to open a Source Code Selection dialog.
2. Highlight the name of the source file that you want to display. The selected file name is displayed in the Selection field.
3. Choose OK. The dialog closes, and the new source file appears in the Source Code window.

   If CXpa cannot find the source file, change the search path by selecting Search Path from the Options menu to open a Source Search Path dialog.

Menus

Name	Meaning
File	Contains an item to close the window.
Windows	Contains items for opening additional CXpa windows that display 2D and 3D profile graphs, call graphs, performance reports, or source files.
Options	Contains an item for changing the search path CXpa uses to locate source code files (Source Search Path).
Help	Contains items for invoking the online help system.

Source Code window

Buttons

Name	Meaning
Change File	Displays a Source Code Selection dialog where you can choose another source file to display in the Source Code window.

Context

Use one of the following methods to open a Source Code window:

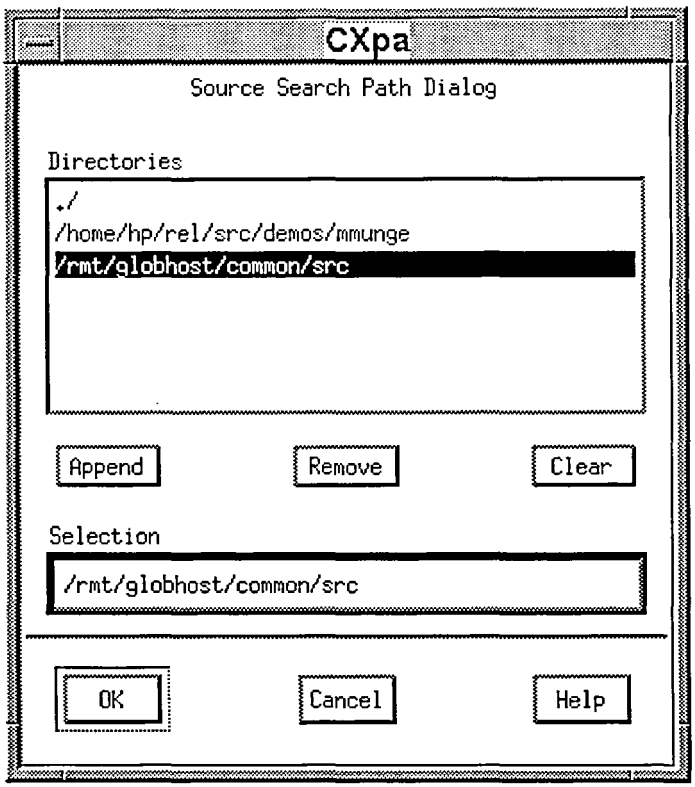
- Select Source Code from the Windows menu in any CXpa window.
- Click on a bar in the graph of the 2D or 3D Profile windows.
- Use the Show in Source button on the Routine Detail dialog.

Related Windows

2D Profile window
3D Profile window
Analysis Control window
Analysis Report window
Call Graph window
Executable Manager window
Routine Detail dialog
Source Search Path dialog
Source Code Selection dialog

Source Code window

Source Search Path dialog



Description

Use the Source Search Path dialog to change CXpa's search path. CXpa uses its search path to find source files when you list a source file (by clicking on graphs in the 2D or 3D Profile windows or by using the Source Code window). CXpa uses the location of the source files embedded in the executable by the compiler to look for source files, so you will need to modify the search path if you have moved the source files after compiling them.

Adding a directory to CXpa's search path

To add a directory to CXpa's search path:

1. Enter a directory path name in the Selection field.

2. Choose Append to display the path name in the Directories list.
3. Choose OK.

Removing a directory from CXpa's search path

To remove a directory from CXpa's search path:

1. Highlight the path name to remove in the Directories list.
2. Choose Remove.
3. Choose OK.

Fields

Heading	Meaning
Directories	Lists the directories in CXpa's search path.
Selection	Allows you to enter a directory name to add to or delete from CXpa's search path.

Buttons

Name	Action
Append	Adds the directory in the Selection field to the Directories list.
Remove	Removes a highlighted directory from the Directories list.
Clear	Removes all the directories from the list.
OK	Accepts the directories in the Directories list as CXpa's search path and closes this dialog.
Cancel	Closes this dialog without changing CXpa's search path.
Help	Displays a help page for this dialog.

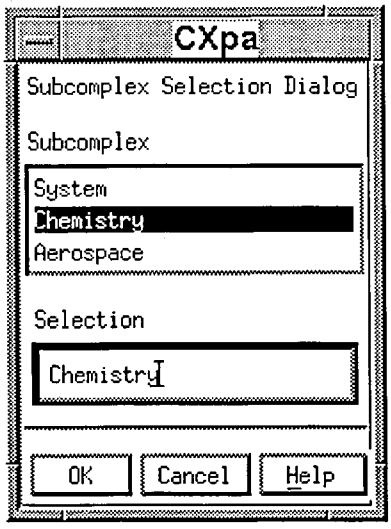
Context

To open a Source Search Path dialog, select Source Search Path from the Options menu in the Source Code, Executable Manager, or Analysis Control windows.

Related Windows

Analysis Control window
Call Graph window
Executable Manager window
Source Code Selection dialog
Source Code window

Subcomplex Selection dialog



Description

Use the Subcomplex Selection dialog to select the name of the subcomplex on which you want to execute your program. When you start CXpa, it queries the system for the number of subcomplexes configured and only makes this dialog available if there is more than one.

A *subcomplex* is a collection of processors and memory from one or more hypernodes of an Exemplar or SPP Series system. The subcomplex defines the boundary within which all threads belonging to a process execute.

The default value is the subcomplex from which you invoked CXpa. You only need to use this dialog if you want to run your program on a different subcomplex. You must have the appropriate permissions to run the process on the specified subcomplex.

Subcomplex Selection dialog

The names of all subcomplexes on your system are displayed in the Subcomplex field. To select a different subcomplex:

1. Highlight the name of the subcomplex you want to select. Your selection is displayed in the Selection field. You can also type the name of the subcomplex you want to select in this field.
2. Choose OK.

For more information on subcomplexes on Exemplar and SPP Series systems, refer to the scm(1) and scm(4) man pages, the *SPP-UX System Administration Guide* or contact the system administrator at your site.

Fields

Field	Meaning
Subcomplex	Lists the names of all subcomplexes on your system.
Selection	When you first invoke the Subcomplex Selection dialog, this field contains the name of the subcomplex from which you invoked CXpa. Otherwise, it displays the currently selected subcomplex (the subcomplex your process is set to run on).

Buttons

Name	Action
OK	Selects the subcomplex displayed in the Selection field and closes the dialog.
Cancel	Closes this dialog without applying the changes.
Help	Displays a help page for this dialog.

Context

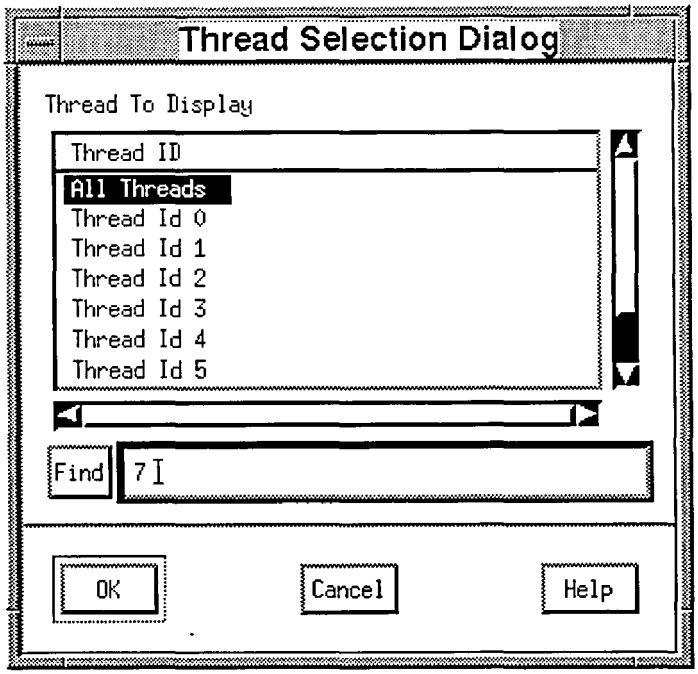
To open a Subcomplex Selection dialog, use the Subcomplex Selection button on the Executable Manager window. The Subcomplex Selection button is not available if only one subcomplex is configured on your system.

CXpa queries the system for available subcomplexes at start-up, so if a subcomplex is added during a CXpa session, it will not be visible to CXpa during that session.

Related Windows

Executable Manager window Info Session dialog

Thread Selection dialog



Description

Use the Thread Selection dialog to select the thread ID number of the thread whose data you wish to display in the Call Graph window. When the Thread Selection dialog is first opened, it displays a list of kernel thread ID numbers for the threads allocated for use by your program. The default selection is All Threads.

To select a thread ID:

1. Highlight the thread ID number of the thread whose call graph data you wish to display. You can select one thread ID or All Threads.

   If the program contains a large number of threads, you can specify a string containing the thread ID number in the Find field to search for, then use the Find button to locate that thread ID. CXpa scrolls the thread ID list so that the first thread ID containing the specified string is placed at the top of the list (or, if it is near the end of the list, is visible in the list).

Thread Selection dialog

Use the Find button again to locate the next thread ID that contains the specified string and scroll the list so that it is placed at the top of the list, and so on. The search will wrap to the beginning of the list. Be sure to highlight the thread ID number in the list to select it after executing the search.

2. Choose OK to close the dialog and update the Call Graph window so that it displays information for the specified thread or threads.

Fields

Heading	Meaning
Thread to Display	Contains a scrolled list of thread ID numbers for the threads allocated for use by your program. The currently selected thread is highlighted.

Buttons

Name	Action
Find	Searches the list of thread ID numbers for occurrences of the string specified in the text entry field opposite the Find button. If a match is found, the thread ID list scrolls so that the first matching routine name is placed at the top of the list (or, if it is near the end of the list, is visible in the list). Use the Find button again to locate the next thread ID that contains the specified string and scroll the list so that it is placed at the top of the list, and so on. The search will wrap to the beginning of the list.
OK	Updates the Call Graph window so that it only displays information for the selected thread.
Cancel	Closes the dialog.
Help	Displays a help page for this dialog.

Context

To open a Thread Selection dialog, select Thread from the View menu in the Call Graph window.

Related Windows

Call Graph window

Routine Detail dialog

X defaults

Description

Default X resource settings (Xdefaults) for the X Window System interface of CXpa are specified in the file /opt/cxpa/newconfig/X11/app-defaults/Cxpa.

To modify these resource settings, copy the default specifications into the file that contains your own X resource specifications (usually .Xdefaults or .Xresources). Then, modify the specifications to suit your needs.

After modifying the resource specifications, enter the following command in your xterm window:

```
% xrdb -merge ~/resource_file
```

resource_file is the name of the file that contains your resource specifications (usually .Xdefaults or .Xresources).

NOTE: If any of these resource specifications conflict with the settings used by your window manager, the window manager may override your CXpa resource specifications.

The Xdefaults in the /opt/cxpa/newconfig/X11/app-defaults/Cxpa file are organized into the following categories:

- Generic resources
- 2D and 3D graph resources
- Session behavior resources

CXpa also supports standard X/Motif general resource specifications. Refer to your X Window System documentation for more information.

Generic application resources

These resources define basic properties for all CXpa windows:

```
Cxpa*font: <font>
Cxpa*fontList: <font>
```

NOTE: Displaying a 2D profile with a large font specified in an application X resource can result in an incorrectly sized initial 2D profile display. This can also occur when no data is collected for the selected metric. To work around the problem, resize the 2D Profile window manually, then use the "Show All" button, or specify a smaller, fixed font.

```
Cxpa*background: <color>
Cxpa*foreground: <color>
```

2D and 3D graph resources

The following resources define the fonts used to display axis labels and titles in 2D Profile window graphs:

```
Cxpa*xrtAxisFont: <font>
Cxpa*xrtHeaderFont: <font>
```

The following resources specify the font size scaling factor for axis labels and titles in 3D Profile window graphs. The larger the number, the larger the font, relative to the size of the graph. As the 3D graph is enlarged, the axis and title fonts are enlarged proportionately. The default font size scaling factor is `80`.

```
Cxpa*xrt3dAxisStrokeSize: <size>
Cxpa*xrt3dAxisTItleStrokeSize: <size>
```

Session behavior resources

These resources define:

- Foreground and background colors for graphs, reports, and source code displayed in CXpa analysis windows (2D and 3D Profile, Call Graph, Analysis Report, and Source Code windows).
- Auto-positioning behavior for xterms CXpa creates.
- Automatic window creation behavior for graph, source code, and report windows.

```
Cxpa*analysisBackgroundColor: <color>
Cxpa*analysisForegroundColor: <color>
```

The above resources define foreground and background colors for graphs, reports, and source code displayed in the 2D and 3D Profile, Call Graph, Analysis Report, and Source Code windows. The default foreground color is `white`; the default background color is `black`. The `*foreground`, `*background`, `-bg`, and `-fg` X resource settings do not affect the background and foreground colors for graphs and reports in analysis windows. These must be set explicitly with the resources listed above.

```
Cxpa*autoPosition: <Boolean>
```

By default, auto-positioning is enabled (`True`). Set this resource to `False` if you are running a virtual window manager (such as `tvtwm`) and you find that CXpa positions xterms that it creates (such as the process interface window) in unexpected locations (that is, not in the current virtual root window).

`Cxpa*defaultWindow: <windows>`

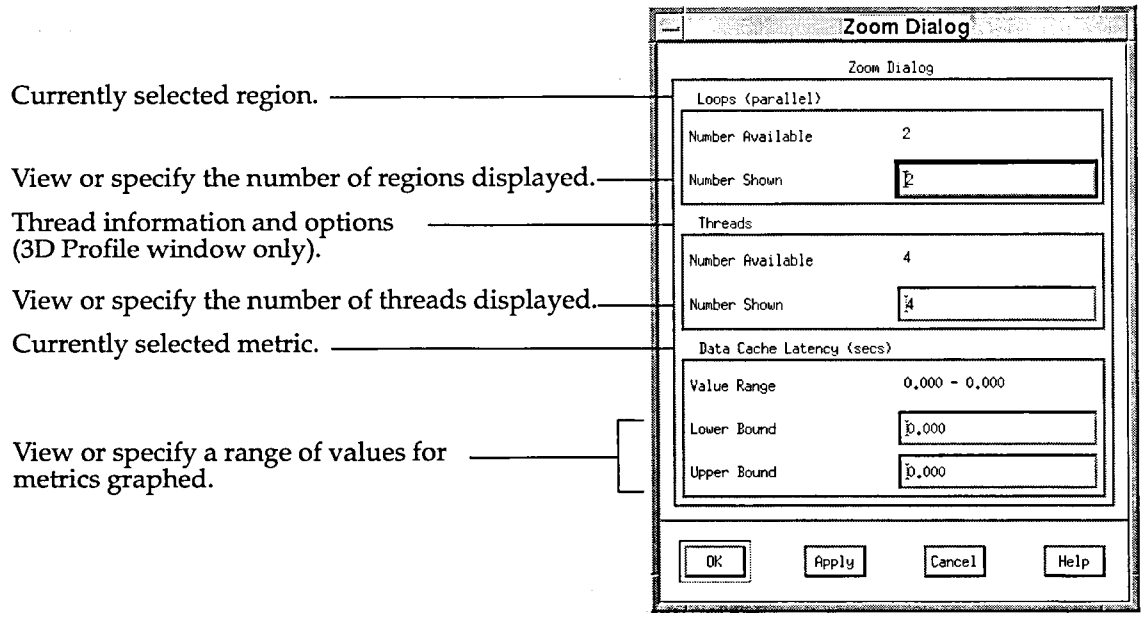
Use this resource to specify the window or windows that are created automatically when you invoke CXpa in analysis mode (without specifying an executable). You can specify multiple windows. Separate each value with a space. The default is `None`. Valid values for *<windows>* are as follows:

- `2DProfile`—Automatically create a 2D Profile window.
- `3DProfile`—Automatically create a 3D Profile window.
- `Callgraph`—Automatically creates a Call Graph window in analysis mode.
- `All`—Automatically create a 2D Profile window, 3D profile window, Analysis Report window, Call Graph, and Source Code window in analysis mode.
- `None`—Disable automatic creation of windows in analysis mode (the default).
- `Report`—Automatically create an Analysis Report window.
- `Source`—Automatically create a Source Code window.

Related Windows

2D Profile window
Analysis Control window
Call Graph window
Source Code window
3D Profile window
Analysis Report window
Executable Manager window

Zoom dialog



Description

Use the Zoom dialog to view or specify the range of metric values graphed, the number of regions displayed, or the number of threads displayed along a given axis in a 2D or 3D graph. This is especially useful in cases where there are a large number of data items to graph and you want to focus on a subset of the data.

Labels in the Zoom dialog indicated the currently selected region level and metric type and are updated as different metrics and regions are selected in the associated 2D or 3D Profile window.

2D graph zoom options

The following zoom options are available for the 2D graph:

- To change the number of source code regions displayed in the current window, enter a new value in the Number Shown field for the currently selected metric.
- To specify a different range or limit the range of metric values graphed along the Metric (horizontal) axis, enter a new value in the Upper- and/or Lower-bound fields for the currently selected metric.

3D graph zoom options

The following zoom options are available for the 3D graph:

- To change the number of source code regions graphed along the Region axis, enter a new value in the Number Shown field for the currently selected region.
- To change the number of threads graphed, enter a new value in the Number Shown field in the Threads panel of the dialog.
- To specify a different range or limit the range of data values (metrics) graphed, enter a new value in the Upper- and/or Lower-bound fields for the currently selected metric.

Fields

Field	Description/Valid values
Value Range	Displays the range of data values (metrics) that can be shown in the graph.
Lower Bound	Sets the lower-bound of the range for the specified axis. This value must be less than the value specified in the Upper Bound field but can be smaller than the minimum data value.
Upper Bound	Sets the upper-bound range for a given axis. This value must be greater than the value specified in the Lower Bound field but can be larger than the maximum data value.
Available	Displays number of available data items (regions or threads) displayed in the graph.
Number Shown	Shows the number of items (regions or threads) currently displayed. Valid values are integers from one to the number available. Values larger than the number available are ignored.

Buttons

Name	Action
OK	Updates the graph display according to the selected values and closes this dialog.
Apply	Updates the graph display according to the selected values, but does not close the dialog.
Cancel	Closes this dialog without making any changes.
Help	Displays a help page for this dialog.

Context

To open a Zoom dialog, select Zoom from the View menu in the 2D Profile or 3D Profile window.

Related Windows

2D Profile window
Filter Profile dialog
3D Profile window
Region Subset Selection dialog

Zoom dialog

Commands 7

This chapter contains a reference page for each CXpa command. Commands can be:

- Entered in line mode (when you invoke CXpa with the `-nw` option) at the `(CXpa)` command prompt
- Used in CXpa command files
- Executed in batch mode

You can abbreviate CXpa commands. For example, the command `analyze cpu loop` can be abbreviated to `an c l`. The shortest unique abbreviation for each command is shown in the upper right corner of the first reference page for each command, immediately below the command name.

Each reference page displays its command name and shortest abbreviation at the top, followed by a one-line description. The rest of the page is divided into the following sections:

- **Syntax**—Lists the format rules for the command and its parameters.
- **Description**—Explains the purpose and functionality of the command.
- **Examples**—Shows one or more examples illustrating the use of the command.
- **Related Commands or Topics**—Lists CXpa commands or topics related to the command being described.

The heading at the top of each command description contains the following lines of information:

Full command name ——— add path
Shortest abbreviation ——— ad p

add path

ad p

Add directories to CXpa's search path for source files.

Syntax

add path *<directory-list>*

Parameter	Meaning
<directory-list>	Specifies one or more directories, separated by a space, to add to CXpa's search path.

Description

The `add path` command appends the specified list of directories to CXpa's search path. CXpa uses its search path to locate source files. By default, the search path contains the:

- Location of the original source files when the program was compiled.
- Location of the executable and/or PDF.
- Current working directory.

CXpa uses the location of the source files embedded in the executable by the compiler to look for source files, so you will need to use the `add path` command to modify the search path if you have moved the source files after compiling them.

Use the `info` command to list CXpa's current search paths. Use the `path` command to replace CXpa's current search path.

You may find it convenient to place the `add path` command in CXpa's initialization file, .cxpainit, to automatically set CXpa's search path each time you invoke CXpa.

Examples

The following examples show typical uses of the `add path` command.

```
(CXpa) add path c_progs
(CXpa) info

(Skipping command output not relevant to this example.)

Current Search Path(s) : /mnt/user/progs/
                         /mnt/user/progs/c_progs
```

The above example of the `add path` command appends the relative path of `c_progs` to CXpa's search path.

```
1.(CXpa) list selectable SUB2
             File prog.f not found in search path.
2.(CXpa) info

(Skipping command output not relevant to this example.)

     Current Search Path(s) : /mnt/dev_tree/wrk

3.(CXpa) add path /mnt/dev_tree/progs
4.(CXpa) list selectable SUB2
       R  1 SUBROUTINE SUB2(MATRIX, VAR3)
      L P 5 DO I=1,5
      L   6    DO J=1,5
```

The above example shows a scenario for using the `add path` command when CXpa cannot find a source file you are trying to list. The line numbers are for reference only.

1. When the `list selectable SUB2` command is issued, CXpa cannot find the file that contains the routine `SUB2`.
2. The `info` command displays CXpa's current search path at the bottom of the `info` command's output.
3. The `add path` command adds /mnt/dev_tree/progs to CXpa's search path.
4. Now that CXpa's search path has been modified, CXpa finds the needed source file when the `list selectable SUB2` command is reexecuted.

Related Commands

`info` `list`
`list selectable` `path`

analyze

an

Display performance reports for profiled source code regions.

Syntax

analyze [*<metric-list>*] [*<region-type>*] [*<routine-list>*] [*<i/o_redirection>*]

Parameter	Meaning

When parameters are used with this command, they must be specified in the order shown in the syntax statement above.

<metric-list>

Restricts the output of this command to the specified metrics. If no metrics are specified, all available metrics are displayed.

When used, this parameter should precede any other parameter. Separate multiple metrics with a space. Valid values are as follows:

`call_graph`
`counts`—Valid for routines only.
`cpu`
`wall_clock`
`events`

<region-type>

Specifies the type of source code region for which you want to display reports. Valid values are as follows:

`routine`—Routines.
`loop`—All loops.
`pregion`—Parallel loops only.

If you do not specify a region type, reports are displayed for all profiled regions. You can only specify one region type.

<routine-list>

For the selected source code region type, specifies one or more routines. Separate multiple routines with a space. If you do not specify a *region-type* with the *routine-list* parameter, routine reports are displayed.

<i/o_redirection>

Redirects this command's standard output or error to the specified file when you include one of the redirection operators (<, >, >>, >&, >>&).

Description

Use the `analyze` command to create and display textual performance reports. When you execute the `analyze` command without specifying any parameters, all available reports and metrics are displayed for all profiled regions in your program.

The `analyze` command generates the reports from the data in the current performance data file (PDF), so a PDF must exist before you can display a performance report. CXpa creates a PDF when you select region types with a `select` command and execute the program with the `run` command. If you have not specified a PDF name in this profiling session, CXpa uses the default file name *<executable>*.pdf.

When you invoke CXpa with the name of a PDF file only, you can use the `analyze` command to look at reports from multiple PDFs or PDFs created in previous CXpa sessions (including PDFs created on different architectures). Then, use the `set pdf` command to change the name of PDF being analyzed.

NOTE: If you invoked CXpa with the name of an executable, you can only analyze PDFs that were created during the current CXpa session and generated from the executable you are currently profiling.

To generate reports for a specific source code region level, use the `analyze` command followed by a *region-type* parameter: `analyze routine`, `analyze loop`, or `analyze pregion`.

You can use the *metric-list* and *routine-list* parameters to display reports for a subset of metrics and/or routines, respectively.

CXpa displays reports in line mode using the pager specified with your `PAGER` environment variable. If the `PAGER` environment variable is not set, CXpa uses the `more` command to page output. You can also redirect output to a file using redirection operators.

Reports

The `analyze` command can display reports for the following source code regions under the listed conditions:

- **Routine reports**—Use the `analyze routine` command to display routine reports.

  Routine reports are available if the source files were compiled with an Exemplar compiler using the `+pa` option or instrumented for profiling with `cxoi` and routine regions were profiled.

- **Dynamic Call Graph report**—Use the `analyze call_graph` command to display a dynamic call graph report.

  Dynamic call graph reports are available if the program was compiled with an Exemplar compiler using the `+pa` option or instrumented for routine-level profiling with `cxoi`, CPU time and wall clock time metrics were collected, and routines were profiled.

- **Loop reports**—Use the `analyze loop` command to display reports for all loops (including summary information for profiled parallel loops).

  Loop reports are available if the program was compiled with an Exemplar compiler using the `+pa` option at optimization level `+O2` or `+O3` and loops were profiled.

- **Parallel Region reports**—Use the `analyze pregion` command to display reports for parallel loops.

  Parallel region reports are available if the program was compiled with an Exemplar compiler using the `+pa` option at optimization level `+O3 +Oparallel` and parallel loops were profiled.

For a detailed description of CXpa reports and the metrics displayed in each report, refer to the "Introducing metrics," "Reports," "Dynamic Call Graph report," "Loop reports," "Parallel Region reports," "Routine reports," and "Report fields" online help topics or sections of this book.

Examples

The examples in this section show how to use the `analyze` command.

```
1.(CXpa) select routine all
2.(CXpa) collect cpu wall_clock events
3.(CXpa) set events local_misses
4.(CXpa) run
   (Program runs to completion, and any output is displayed.)
5.(CXpa) analyze
   (Performance reports are displayed.)
```

The above scenario shows how you would normally use the `analyze` command in conjunction with other CXpa commands to instrument your program, collect performance data, and generate performance reports. The line numbers are for reference only.

1. The `select routine all` command tells CXpa to select all routines in your program for profiling.
2. The `collect cpu wall_clock events` command tells CXpa to collect CPU time, wall clock time, and events (execution counts are

always collected). You must then use the `set events` command to specify the type of event to collect.

3. The `set events local_misses` command tells CXpa to collect the number of times that a memory reference had to be satisfied from memory on the processor's node due to a miss in the processor's data cache. The `local_misses` parameter is specific to Exemplar S2000, Exemplar X2000, and SPP1600 Series systems.
4. The `run` command executes the program and initiates profile data collection. Profile data collection ends when the program runs to completion.
5. The `analyze` command calculates and displays all possible performance reports from data collected and stored in the PDF file.

The following examples show how to use the `analyze` command after regions and metrics have been selected for profiling and profiling data has been collected and stored in a PDF.

```
(CXpa) analyze events loop SUB2
```

The above command creates and displays loop performance analysis reports for event metrics collected at all profiled loop regions executed in subroutine `SUB2`. The order of the parameters is significant. The *metric-list* parameter (in this case, `events`) must be specified first, and the routine specifier `SUB2` must follow the `loop` *region-type* parameter.

```
(CXpa) analyze loop > report_output
```

The above command creates loop performance analysis reports containing all metrics collected for all profiled loop regions in the program and redirects the output to a file named report_output. Any existing data in the file report_output is overwritten.

```
(CXpa) analyze events cpu
```

The above command creates and displays performance reports containing events and CPU time metrics collected for all profiled regions in the program.

```
(CXpa) analyze loop INIT DISPLAY
```

The above command creates and displays loop performance reports containing all collected metrics for the profiled loop regions executed in subroutines `INIT` and `DISPLAY`.

```
(CXpa) analyze counts
```

The above command creates and displays a Call Counts reports for all profiled routines in the program.

```
(CXpa) analyze call_graph
```

The above command creates and displays a dynamic call graph report for all profiled routines in the program.

Related Commands

`collect`
`run`
`select`
`set events`
`set pdf`

Related Topics

Dynamic Call Graph report
Introducing metrics
Introducing source code regions
Loop reports
Report fields
Reports
Routine reports
Parallel Region reports

collect

col

Specify the metrics that you want to collect while profiling your program.

Syntax

`collect [call_graph] [counts] [cpu] [wall_clock] [events]`

Specify one or more of the following parameters with the `collect` command. Separate multiple parameters with a space.

Parameter	Meaning
`call_graph`	Collects dynamic call graph information for all profiled routine regions. If you specify `call_graph`, then you must also specify the `cpu` and `wall_clock` parameters.
`counts`	By default, execution counts are always collected. To restrict metric collection to execution counts, specify the `counts` parameter only with the `collect` command (for example, `collect counts`).
`cpu`	Collects CPU time and execution counts.
`wall_clock`	Collects wall clock time.
`events`	Collects event metrics specified with the `set events` command.

Description

The `collect` command tells CXpa which metrics to collect at the regions of your program that are selected for profiling.

Each use of this command replaces the values previously set.

NOTE: Unless you have selected source code regions to profile with the `select` command, no region-level analysis will be possible.

If you specify event collection with the `events` parameter of the `collect` command, you must use the `set events` command to specify the type of event collected. Refer to the "`set events`" online help topic or section of this book for more information.

Examples

The examples in this section show various ways to use the `collect` command, assuming your program is compiled appropriately for the regions selected for profiling.

```
1.(CXpa) select routine all
2.(CXpa) collect cpu wall_clock call_graph events
3.(CXpa) set events local_misses
```

The above scenario shows how you would normally use the `collect` command in conjunction with other CXpa commands to specify the metrics you want to collect. The line numbers are for reference only.

1. The `select routine all` command tells CXpa to select all available routines in your program for profiling.
2. The `collect cpu wall_clock call_graph events` command tells CXpa to collect CPU time, wall clock time, dynamic call graph information, and events (by default, execution counts are also collected). You must use the `set events` command to specify the type of event metrics to collect.
3. The `set events local_misses` command configures off-processor event counters to collect locally resolved data cache miss counts and latency. The `local_misses` parameter is specific to Exemplar S2000, Exemplar X2000, and SPP1600 Series systems.

```
(CXpa) select loop in SUB1 SUB2
(CXpa) collect wall_clock cpu
```

The first command in the above example selects all loops at the currently specified loop nesting level in routines `SUB1` and `SUB2` for profiling. The second command specifies collection of CPU time and wall clock time metrics for these loops.

Related Commands

`analyze`
`run`
`select`
`set events`
`set pdf`
`set visibility`

Related Topics

Introducing metrics

Selecting metrics in line mode

continue

con

Continue profiling a paused program.

Syntax

```
continue
```

Description

The `continue` command resumes the profiling of a paused program. When you continue profiling, the selected program resumes execution, and CXpa resumes profile data collection.

Examples

The following example shows a scenario in which you would use the `continue` command. The line numbers are for reference only.

```
1.(CXpa) run
2.(CXpa) CTRL-c
   (Pressing CTRL-c pauses program execution.)
3.(CXpa) analyze
   (Intermediate performance reports are displayed.)
4.(CXpa) continue
   (Program runs to completion, and data collection ends.)
5.(CXpa) analyze
   (Complete performance reports are displayed.)
```

The following lines explain the example above by number:

1. The `run` command runs the program and initiates profile data collection.
2. Pressing **CTRL-c** pauses program execution.
3. The first `analyze` command displays the performance information that has been collected to this point.
4. The `continue` command resumes the program execution and data collection.
5. When the program has run to completion, CXpa is finished collecting profile data. The `analyze` command is used again to display the final profile results of this run.

Related Commands `run` `stop`

cxpa

Invoke CXpa, the Exemplar performance analyzer.

Syntax

cxpa [*<executable>*] [**-help**] [**-nap**] [**-nw**] [**-nx**]
[[**-path** *<dir>*] ...] [[**-pdf**] *<filename>*] [**-pid**][**-tm** *<architecture>*]
[**-stack** *<bytes>*] [**-w**] [**-x** *<cmdfile>*]
[**-win** *<windows>*][*<X-Toolkit-options>*]
[**-e** *<executable>* [*<arguments ...>*]]

Parameter	Meaning
<executable>	Specify the name of the executable you wish to profile. The default is `a.out`.
-e *<executable>* [*<arguments ...>*]	Supply command-line arguments to the program you are profiling. This must be the last option on the command line. This option is useful for batch execution; for example: **cxpa -x** *<cmdfile>* **-e a.out** *<args-list>*
-help	Display the CXpa usage message, which lists and describes command line options.
-nap	Disables auto-positioning of xterms in GUI mode. Use this option if you are running a virtual window manager and you find that CXpa positions xterms that it creates in unexpected locations (that is, not in the current virtual root window). You can also fix the problem by setting the X application resource `Cxpa*autoPosition:` to `False` (the default setting is `True`).
-nw	Invoke CXpa in line mode. Do not bring up the GUI.
-nx	Do not execute the CXpa commands in .cxpainit, the CXpa start-up command file.

-path *<dir>*

Append the specified directory to the end of CXpa's search path. CXpa uses its search path when it looks for a source file. Any number of directories can be specified by repeating the `-path` option several times. The list of directories is initialized to the current working directory and the directory where the executable or PDF is located.

-pdf *<filename>*

Invoke CXpa with the specified performance data file. CXpa uses the PDF to store profiling data and to generate the performance reports. The default PDF name is *<executable>*.pdf.

-pid

Adds the process identification number (of the process you are profiling) to the name of the PDF that CXpa creates during a profiling session.

-stack *<bytes>*

Specify the size of the profiling stack in bytes. The default is 10240 bytes. If CXpa aborts because the profiling stack is too small, use this option to increase its size.

-tm *<architecture>*

If you are pre-instrumenting an executable on a different architecture than the one the executable will be run on to generate profiling data, you must specify this option when you invoke CXpa. This ensures that the correct timing routines are called to collect metrics for the target system. Valid values for *<architecture>* are as follows:

`s2000`—Exemplar S2000
`x2000`—Exemplar X2000
`spp1200`—SPP1200 Series
`spp1600`— SPP1600 Series

-w

Suppress warning messages issued by CXpa.

-win *<windows>*

Specify the window or windows that are created automatically when you invoke CXpa in analysis mode (without specifying an executable). The default is `None`.

Valid values for *<windows>* are as follows:

```
2DProfile
3DProfile
Callgraph
All
None
Report
Source
```

You can specify multiple values. Separate multiple values on the command line with a space.

-x *<cmdfile>*

Execute the CXpa commands in the specified file. After CXpa has executed the commands in the file, the profiling session is terminated.

<X_Toolkit_options>

Specifies X Toolkit options. For more information about these options, refer to your X Window System documentation.

Description

CXpa is an interactive performance analyzer for C, C++, and Fortran 77 applications. You can profile routines and loops (including parallel loops) if your application is compiled with Exemplar compilers using the `+pa` option. You can also use the Exemplar object file instrumentor, `/opt/cxpa/bin/cxoi`, to instrument object files and archive libraries created with any PA-RISC targeting compiler for routine-level profiling.

CXpa has three interfaces: an X/Motif graphical user interface (GUI mode), a character-oriented tty interface (line mode), and a batch mode interface.

Using CXpa, you can collect performance data for routines, loops, or parallel loops. You can select any of these source code regions for profiling, provided they exist in your program. The data collected for profiled regions can be viewed and analyzed in 2D and 3D graphs and reports.

CXpa stores the performance information collected for a specific run of your program in a performance data file, called a PDF. By default, CXpa appends .pdf to the name of the executable file to form the name of the PDF file (for example, a.out.pdf).

PDFs are platform-independent—the data stored in a PDF created on one platform can be viewed and analyzed on a different platform. For example, you can collect performance data for a large program in batch mode running on an SPP1600 Series system, then move the PDF file to a different subcomplex or to an Exemplar S2000 system to interactively view and analyze the data.

The tasks you can perform during a CXpa session depend on how you invoke CXpa:

- To select regions and metrics for profiling and execute your program under CXpa's control to collect performance data, invoke CXpa with the name of an executable file.

  In this mode, you can also write region and metric selections directly to the executable or to a copy. This is called *pre-instrumenting* an executable. You can then exit CXpa and run the pre-instrumented executable outside CXpa's control to collect performance data.

- To analyze performance data files (PDFs) created in previous CXpa sessions or with pre-instrumented executables, invoke CXpa with the name of a PDF file or a list of PDF files.

Preparing programs for profiling with CXpa

To prepare an application for profiling with CXpa:

- Compile the application with an Exemplar compiler, using the `+pa` option.

  or

- Use the Exemplar object file instrumentor (`/usr/convex/bin/cxoi`, a separate utility shipped with CXpa) to instrument object and archive library files produced by any PA-RISC targeting compiler for routine-level profiling with CXpa.

Refer to the following for information about preparing applications for profiling:

- "Compiling" online help topic or section of this book
- "Using cxoi to instrument object files and libraries" online help topic or section of this book
- `cxoi` man page

Starting CXpa in GUI mode

To start CXpa in GUI mode:

1. Set your X DISPLAY environment variable. For example, using C shell syntax:

   ```
   % setenv DISPLAY display_name:0.0
   ```

2. Invoke CXpa with either the name of an executable that has been prepared for profiling with CXpa or with the name of a PDF file or list of PDF files. For example:

   ```
   % /opt/cxpa/bin/cxpa a.out &
   ```

The above command starts CXpa in GUI mode. Because an executable file is specified, the Executable Manager window appears. The & runs CXpa in the background.

```
% /opt/cxpa/bin/cxpa myprog.pdf &
```

The above command also starts CXpa in GUI mode. Because the name of a PDF file is specified, the Analysis Control window appears. The & runs CXpa in the background.

Refer to the "Learning CXpa quickly" online help topic or section of this book for step-by-step instructions on how to select regions and metrics for profiling and how to run your program to collect performance data.

Refer to the "Analyzing PDFs only" or "Analysis Control window" online help topics or sections of this book for instructions on analyzing PDF files.

Starting CXpa in line mode

Line mode allows you to use CXpa interactively without the graphical user interface. Line mode is designed primarily for use on windowless terminals (for example, VT-100s), but you can also use it on X terminals. In line mode, performance data is displayed in textual reports.

To start CXpa in line mode:

1. Invoke CXpa with the `-nw` option (no windows) and specify the name of an executable that has been prepared for profiling with CXpa, the name of a PDF file, or a list of PDF files. For example:

   ```
   % /opt/cxpa/bin/cxpa -nw a.out
   ```

   The above command starts CXpa in line mode. Because an executable file is specified, you can use the `select`, `collect`, and `set events` commands to select source code regions and metrics for profiling, use the `run` command to execute the program and collect profiling data, then use the `analyze` command to view textual performance reports.

   ```
   % /opt/cxpa/bin/cxpa -nw myprog.pdf &
   ```

   The above command starts CXpa in line mode and specifies the PDF file myprog.pdf as the current PDF. In this "analysis-only" mode of CXpa, you can use the `analyze` command to analyze the data in the specified PDF file and the `set pdf` command to specify a new PDF file to analyze.

   Refer to the "Learning CXpa quickly," "`select`," "`collect`," `set events`," "`run`," and "`analyze`" command online help topics or

section of this book for step-by-step instructions on how to select, collect, and analyze performance data in line mode.

Refer to the "Analyzing PDFs only," "`analyze`," and "`set pdf`' command online help topics or sections of this book for instructions on analyzing PDF files in line mode.

Using the CXPA environment variable

You can specify CXpa command line options in the CXpa environment variable (`CXPA`). This frees you from having to enter frequently used options repeatedly from the command line.

To use this environment variable in C shell (csh), use the following syntax:

```
% setenv CXPA '-option -option ...'
```

For example, the command

```
% setenv CXPA '-nw -pid -w'
```

forces CXpa to start in line mode (`-nw`). The `-pid` option tells CXpa to add the process ID number of the process you are profiling to the name of the PDF file it creates when you run your program. The `-w` option suppresses warning messages issued by CXpa.

Examples

The following examples show various ways to invoke CXpa.

```
% cxpa -help
```

The above command displays a usage message for the `cxpa` command.

```
% cxpa prog.out
```

The above command invokes CXpa in GUI mode and specifies the executable file prog.out for profiling. Because an executable file is specified, the Executable Manager window appears. Profiling data will be collected in the PDF file named prog.out.pdf.

```
% cxpa -pdf my.pdf
```

The above command invokes CXpa in GUI mode; the `-pdf` option specifies the PDF file my.pdf. The `-pdf` option is optional if the file name ends in .pdf. CXpa uses the PDF to store the performance data and to generate performance reports.

Because a PDF file is specified without an executable file, the Analysis Control window appears. From this window, you can analyze PDFs created from different executables, but you cannot execute your program or gather more performance data without specifying an executable at the CXpa invocation line.

```
% cxpa -nw prog.out
```

In the above example, the `-nw` option invokes CXpa in line mode and the executable `prog.out` is specified.

```
% cxpa -nx a.out
```

The above command invokes CXpa but prevents it from processing any initialization files.

```
% cxpa -x cmdfile my.out >& output_file < input_file
```

The above command invokes CXpa in batch mode (which is not an interactive mode). The `-x` option tells CXpa to execute the CXpa commands in the file cmdfile. Program output and messages are redirected to the file output_file, and input for the executable `my.out` is read from the file input_file (unless redirected in the command file).

Related Commands

`analyze`
`collect`
`continue`
`deselect`
`merge`
`quit`
`run`
`select`
`set events`
`set visibility`

Related Windows

2D Profile window
3D Profile window
Analysis Control window
Analysis Report window
Call Graph window
Executable Manager window
Profile Selection dialog

Related Topics

Analyzing PDFs only
Compiling
Introducing metrics
Introducing source code regions
Learning CXpa quickly
Performance data files
Profiling MPI and PVM applications with CXpa
Reports
Using cxoi to instrument object files and libraries
Using pre-instrumented executables

deselect

d

Deselect source code regions for profiling.

Syntax

deselect [*<region-type>*] [**all** | **in** *<routine-list>* | **at** *<line-number-list>*]

Parameter	Meaning
<region-type>	Specifies the type of region to deselect. Valid values are as follows: `routine`—routines. `loop`—all loops. `pregion`—parallel loops only.
all	Deselects the specified *<region-type>* in all routines in your program. If you do not specify a region type, all regions are deselected.
in *<routine-list>*	Specifies the names of one or more routines whose region types you want to deselect. Separate routine names with a space. If two routines have the same name, prefix them with a file name followed by a colon: *<file-name>* : *<routine-name>*.
at *<line-number-list>*	Deselects all region types at *<line-number-list>*. *<line-number-list>* specifies one or more line numbers of the region type that you want to deselect. Separate line numbers with a space. To deselect a region type in another source file, prefix the line number with a file name followed by a colon: *<file-name>* : *<line-number>*.

Description

The `deselect` command deselects source code regions for profiling in one or more routines or at one or more line numbers. The `deselect` command has no effect on regions that are already deselected.

deselect

When you invoke CXpa in line mode, all source code regions in your program are deselected for profiling until you select one or more of them with the `select` command. CXpa ignores deselected regions when you profile a program.

Use the `list selectable` command to list source code regions that can be selected or deselected for profiling.

Examples

The following examples show various ways to use the `deselect` command.

```
(CXpa) deselect loop in init matrix_mult sub3
```

The above command deselects all loop regions for profiling in the routines `init`, `matrix_mult`, and `sub3`.

```
(CXpa) deselect at 14
```

The above command deselects all regions at line 14 in the current source file.

```
(CXpa) deselect pregion all
```

The above command deselects all parallel loops for profiling in all routines in your program.

```
(CXpa) deselect loop in sub2
Routine sub2 has no instrumented loop entries.
```

In the above example, the loops in subroutine `sub2` could not be deselected because the source file containing `sub2` was not compiled with the `+pa` option at optimization level `+O2` or `+O3` or did not contain loops.

Related Commands

`list selectable` `run` `select`

help
h

Display help information.

Syntax

help [*<string>*]

Parameter	Meaning
<string>	A character string used to search for help topics. All topic titles that contain the string are listed. If only one match is found, the help text for that topic is automatically displayed. If you enter the `help` command without specifying a search string, CXpa displays the text for the `help` command. The string can contain white space without being enclosed in quotes.

Description

The `help` command displays help information on specified commands, CXpa error, warning, and informational messages, and other topics. CXpa searches the help topic titles for the string you specify.

The output of the `help` command is sent to the pager specified in your `PAGER` environment variable. If you have not specified a pager with the `PAGER` environment variable, CXpa uses the `more` command to page output.

To display an ASCII text version of the release notice for the version of CXpa currently installed on your system, enter `help release`.

To view help for CXpa messages, use the command `help` *<msg_number>*.

Examples

The following examples show how to use the `help` command.

```
(CXpa) help continue
```

The above command displays help information for the `continue` command.

```
(CXpa) help A19
A19
       Message     PDF <filename> is empty or corrupt.
       Type        ERROR
      Explanation Regenerate the PDF or try a different PDF.
```

The above command displays help information for CXpa error message number A19.

```
(CXpa) help pdf
Searching the topic titles found the following matches for
       'PDF':

   set pdf
   Analyzing PDFs only
   Info PDF dialog
   New PDF dialog
   Open PDF dialog

(CXpa) help set pdf

                                                    set pdf
                                                    set p

          Specify the name of the performance data file (PDF).

Syntax       set pdf <filename>

(Skipping lines of output.)
```

In the above example, the command `help pdf` is ambiguous; CXpa displays all help topic titles that match the string `pdf`. The command `help set pdf` displays the help text for the `set pdf` command.

```
(CXpa) help index list
Index

      Enter 'help <topic_name> to view its  help page.

Windows                                   Using CXpa

About_CXpa_dialog                         Overview_of_CXpa

Analysis_Control_window                   CXpa_limitations
```

(...lines of output omitted)

```
(CXpa) help commands

Commands

      Enter 'help <command_name>' to view the help page
      for that command.

add path

    Add the specified list of directories to CXpa's search path
       for source files.

analyze

     Display performance analysis reports for profiled source
       code regions.
```

(...lines of output omitted)

As shown in the above example, you can view lists of online help topics by entering the following commands:

- `help using list`
- `help index`
- `help commands`
- `help windows`
- `help reports list`

Related Commands

`info`

info

i

Display information about your CXpa session.

Syntax

```
info
```

Description

The `info` command lists three categories of information:

Executable information

- **Current executable**—Name of the executable that you are profiling.
- **Current Process State**—Current state of the process you are profiling. The states include `running`, `paused`, `terminated`, `exited`, and `not started`.
- **Process ID**—Executable's process ID (PID).
- **Created on**—Date that the current executable was created.
- **Instrumentation Version**—Version of the CXpa profiling routines (cxpamon.o) linked into your program.

PDF information

- **Current PDF**—Name of the current performance data file (PDF).
- **Created on**—Date and time that the PDF was created.
- **PDF Format**—Format version number of the PDF.
- **Created by CXpa Version**—Version number of CXpa that created the PDF.
- **Process State**—Process state recorded in the PDF.
- **Executable Profiled**—Name of the executable you are profiling.
- **Executable Created on**—Date that the executable that produced the current PDF file was created.
- **Instrumentation Version**—Version of the CXpa profiling routines linked into your executable when the PDF was created.
- **Visibility Setting**—Indicates whether process-level data or thread-level data will be displayed in CXpa text reports, and displays the current loop nesting level setting.

File and directory information

- **Current List File**—The name of the source file CXpa currently displays with the `list` or `list selectable` commands.
- **Current Working Directory**—Current directory.
- **Current Search Path(s)**—Search path that CXpa uses to find source files.
- **Current Subcomplex**—Subcomplex that your process is currently set to run on.
- **Available Subcomplex(s)**—Lists subcomplexes available on your system. Use the `set subcomplex` command to specify a different subcomplex to run your program on.

Examples

The following example shows the output of the `info` command.

```
(CXpa) info
       Current Executable : /src/demos/mmunge/mmunge.-O3.exe
    Current Process State : Not started
                Process ID : Not started
                Created on : Fri Jul 12 10:24:18 1996
  Instrumentation Version : 3.3
           Current PDF : /src/demos/mmunge/mmunge.-O3.exe.pdf
                Created on : Fri June 28 15:54:15 1996
                PDF Format : 478
  Created by CXpa Version : 3.5
             Process State : exited
      Executable Profiled : /src/demos/mmunge/mmunge.-O3.exe
       Executable Created on : Fri June 28 10:24:18 1996
  Instrumentation Version : 3.5
       Current Visibility : Process, Loop levels 0 to 0
        Current List File : mmunge.f
Current Working Directory : /src/demos/mmunge
   Current Search Path(s) : ./
                            /src/demos/mmunge
       Current Subcomplex : System
  Available Subcomplex(s) : System, Chemistry
```

Related Commands

`version` `set visibility`

list
1

List lines of text from a source file.

Syntax

`list` [*<routine>* | [*<filename>*] [`:`] {*<first-line>* [*<last-line>*] | *<routine>*}]

Parameter	Meaning
<routine>	Specifies the name of a routine to display.
<filename>	Specifies the name of a source file to display.
<first-line>	Specifies a source code line number as the first line to display.
<last-line>	Specifies a source code line number as the last line to display.

Description

The `list` command lists lines of text from the source files that were compiled to form the current executable. Lines containing source code regions that can be selected or deselected for profiling are prefaced with one or more of the following letters: `r` for routines, `l` for loops, and `p` for parallel loops. Lowercase letters indicate regions that are currently deselected, while uppercase letters indicate regions that are currently selected for profiling.

When you execute the `list` command without parameters, CXpa lists the current source file. The current source file is either the last source file specified in a CXpa command or the source file that contains the program's main entry point.

When you use the `list` command, CXpa uses the directories in its search path to find the needed source file. If a source file has been moved after compiling, use the `add path` command to add the new directory to CXpa's search path.

The following list describes each permutation of the `list` command:

- `list`—List the current source file. Press the **SPACEBAR** to page forward.
- `list` *<routine>*—List the routine specified.
- `list` *<filename>*—List the specified source file. This source file must be one of the files compiled to form the current executable.

- `list` *<first-line>* [*<last-line>*]—List parts of the current source file by specifying the first and possibly the last line to display.
- `list` *<filename>*:*<first-line>* [*<last-line>*]—List parts of the specified file by specifying the first and possibly the last line to display.
- `list` *<filename>*:*<routine>*—List the routine specified. The file name allows you to choose between routines with the same names that are in different files.

Examples

The following examples show various ways to use the `list` command.

```
(CXpa) list
 R 1 PROGRAM GENERIC
   2 INTEGER VAR, VAR3, VAR4, COUNT
   3
   4 VAR = 126
   5 VAR4 = VAR * 16
     .
     .
     .
```

The above command lists the current source file. The uppercase `R` at line 1 indicates the routine region starting at line 1 is selected for profiling.

```
(CXpa) list sub2
 r    1 SUBROUTINE SUB2(MATRIX, VAR3)
    . 2 INTEGER MATRIX(5,5), VAR3, VAR5
      3
      4 PRINT *,'ENTERING SUB2'
 l p  5 DO I=1,5
 l    6 DO J=1,5
      7 MATRIX(I,J) = (I + J) - VAR3
      8 ENDDO
      9 ENDDO
     10 VAR5 = 38
     11 PRINT *,'LEAVING SUB2'
     12 PRINT *
     13 VAR5 = 12
     14 END
```

The above command lists the routine named `sub2` in the current source file. The lowercase letters `r`, `l`, and `p` indicate routine, loop, and parallel regions that are deselected for profiling.

```
(CXpa) list subs.f:8 30
   8
 L 9 DO 88 COUNT = 1,10,1
  10
  11 IF (COUNT .LT. 5) THEN
  12 VAR = VAR + 3
  13 CALL SUB1(VAR, VAR3)
  14 ELSE
  15 VAR = VAR + 32
  16 CALL SUB1(VAR, VAR3)
  17 ENDIF
  18
  19 VAR3 = VAR3 - 1
  20
  21 88 CONTINUE
  22
  23 CALL SUB4
  24
  25 IF (VAR .EQ. 2000) THEN
  26 CALL SUB5
  27 END IF
  28
  29 END
  30
```

The above command lists lines 8 through 30 in the file named subs.f. The uppercase L indicates that the loop region beginning at line 9 is selected for profiling.

```
(CXpa) list matrix_mult.c
```

The above command lists the source file named matrix_mult.c.

```
(CXpa) list 20 55
```

The above command lists lines 20 through 55 of the current source file.

```
(CXpa) list 85
```

The above command lists the current source file starting at line 85.

Related Commands

`add path`	`info`
`list selectable`	`path`

list selectable

ls

List source code regions available for profiling in a source file.

Syntax

`list selectable` [*<routine>* | [*<filename>*] [**:**] {*<first-line>* [*<last-line>*] | *<routine>*}]

Parameter	Meaning
<routine>	Specifies the name of a routine to display.
<filename>	Specifies the name of a file to display.
<first-line>	Specifies a source code line number as the first line to display.
<last-line>	Specifies a source code line number as the last line to display.

Description

The `list selectable` command lists only lines of source code that contain source code regions that can be selected for profiling in the source files that were compiled to form the current executable.

Source code regions that can be selected or deselected for profiling are prefaced with one or more of the following letters: `r` for routines, `l` for loops, and `p` for parallel loops. Lowercase letters indicate regions that are currently deselected, while uppercase letters indicate regions that are currently selected for profiling (refer to the "Selecting regions in line mode" online help topic or section in this book for more information).

When you enter the `list selectable` command without parameters, CXpa lists the lines of source code in the current source file that contain selectable source code regions. The current source file is either the last source file specified or the source file that corresponds to the first object file given to the linker to form the current executable.

When you execute the `list selectable` command, CXpa uses the directories in its search path to find the needed source file. If a source file has been moved after compiling, use the `add path` command to add the needed directory to CXpa's search path.

The following list describes each permutation of the `list selectable` command:

- `list selectable`—List the lines of source code containing selectable regions in the current source file.
- `list selectable` *<routine>*—List the lines of source code containing selectable regions in the specified routine.
- `list selectable` *<filename>*—List the lines of source code containing selectable regions in the specified file.
- `list selectable` *<first-line>* [*<last-line>*]—List the lines of source code containing selectable regions in the specified range.
- `list selectable` *<filename>*`:`*<first-line>* [*<last-line>*]—List the lines containing selectable regions in the specified source file in the specified range.
- `list selectable` *<filename>*`:`*<routine>*—List selectable regions in the specified routine. Specifying the file name allows you to distinguish between two routines with the same name that are in different files.

If you enter the `list selectable` command and get no output, it is because there were no selectable source code regions in the range you specified.

Examples

The following examples show various ways to use the `list selectable` command.

```
(CXpa) list selectable
  r    1 PROGRAM GENERIC
 L     9 DO 88 COUNT = 1,10,1
```

The above command lists the regions that can be selected for profiling in the current source file. The annotations show that the routine region at line 1 is currently selected for profiling and the loop region at line 9 is deselected.

```
(CXpa) list selectable sub2
  r   1 SUBROUTINE SUB2(MATRIX, VAR3)
L p   6 DO I=1,5
L     7 DO J=1,5
L     9 DO K=1,30
```

The above command lists the regions that can be selected for profiling in the routine named sub2. The annotations show that the loops at lines 6, 7, and 9 in sub2 are currently selected for profiling and routine SUB2 at line 1 is deselected.

```
(CXpa) list selectable prog.f:68 99
  r  68 SUBROUTINE SUB4
L    76 DO I = 1,100,2
L    79 DO WHILE (K .LT. 10)
L p  82 DO J = 1,20,1
  r  99 SUBROUTINE SUB5
```

The above command lists the regions available for profiling in lines 68 through 99 in the file named prog.f. The annotations indicate that loops are currently selected for profiling, and routines are currently deselected.

```
(CXpa) list selectable subs.f
```

The above command lists the regions that can be selected for profiling in the file named subs.f. In this case, no regions are available for profiling.

```
(CXpa) list selectable 20 55
R 33 SUBROUTINE SUB1(VAR, VAR3)
R 51 SUBROUTINE SUB3(MATRIX)
```

The above command lists the regions that can be profiled on lines 20 through 55 of the current source file.

```
(CXpa) list selectable 51
 R 51 SUBROUTINE SUB3(MATRIX)
 L 56 DO I=1,5
```

The above command lists the regions that can be profiled in the current source file starting at line 51. Both of these regions are currently selected for profiling, as indicated by the uppercase `R` and `L` annotations.

Related Commands

`add path` `info` `list` `path`

merge

m

Combines profiling data from multiple PDF files generated from the same executable into a single PDF file for analysis.

Syntax

merge *<filename>*.pdf

Parameter	Meaning
<filename>.pdf	Specifies the name of the PDF file in which the merged profiling data will be stored.

Description

The merge command combines the profiling data from multiple PDF files generated from the same executable into a single PDF file.

This feature is useful when profiling message-passing applications with CXpa. Refer to the "Profiling MPI and PVM applications with CXpa" online help topic or section of this book for more information.

The PDF files must be generated from the same executable, and the region and metric selections must be identical. CXpa uses the executable associated with the first PDF file specified on the command line when CXpa was invoked. If an invalid PDF is specified, an error message is displayed, and CXpa continues to merge the data from valid PDF files.

When you execute the merge command, CXpa merges all PDF files specified from the command line when you invoke CXpa plus all PDF files specified during the current session with the set pdf command.

To analyze the merged data in line mode, execute the set pdf command, specifying the name of the new PDF file.

Examples

The following example shows how to use the `merge` command.

```
% /opt/cxpa/bin/cxpa a.out.*.pdf -nw

                        CONVEX Performance Analyzer

Type 'help' for help.
Selecting profile a.out.20703.pdf...
Selecting profile a.out.20704.pdf...
Selecting profile a.out.20705.pdf...
Selecting profile a.out.20706.pdf...
Selecting profile a.out.20707.pdf...
Selecting profile a.out.20708.pdf...
Selecting profile a.out.20709.pdf...
Selecting profile a.out.20710.pdf...
Selecting profile a.out.20711.pdf...
(CXpa) merge a.out.summary.pdf
Using PDF a.out.20703.pdf as base of merge
a.out.20704.pdf                    used (1 thread added)
a.out.20705.pdf                    used (1 thread added)
a.out.20706.pdf                    used (1 thread added)
a.out.20707.pdf                    used (1 thread added)
a.out.20708.pdf                    used (1 thread added)
a.out.20709.pdf                    used (1 thread added)
a.out.20710.pdf                    used (1 thread added)
a.out.20711.pdf                    used (1 thread added)
(CXpa) set pdf a.out.summary.pdf
(CXpa) analyze
```

In the above example:

- The command `/opt/cxpa/bin/cxpa a.out.*.pdf -nw` invokes CXpa from the shell in line mode with multiple PDF files. CXpa displays the names of the PDF files specified.
- The command `merge a.out.summary.pdf` specifies that the merged data will be stored in the file a.out.summary.pdf.

  The output of the `merge` command indicates that the PDF file a.out.20703.pdf was specified first; additional PDF files specified must match the base information in that PDF in order to be merged (the base information includes name of the executable file that generated the PDF, along with other information that ensures the PDF files to be merged are compatible).

  The name of each PDF file used to create the merged PDF is also displayed.

- The `set pdf` command opens the new PDF file, a.out.summary.pdf, and makes it the current PDF file.
- The `analyze` command generates all available performance reports from the data in a.out.summary.pdf.

Related Commands

`cxpa` `save executable`

Related Concepts

Profiling MPI and PVM applications with CXpa
Using pre-instrumented executables

path

p

Set CXpa's search path for source files.

Syntax

path *<directory-list>*

Parameter	Meaning
<directory-list>	Specifies one or more directories, separated by a space, as CXpa's search path.

Description

The `path` command replaces CXpa's current search path with the one specified in *<directory-list>*.

By setting CXpa's search path, you tell CXpa where to look for source files. When you compile your source code with CXpa options, CXpa embeds the location of the source files in the executable so that CXpa can find these files. If they are moved after compiling, then CXpa cannot find them.

You can append one or more directories to CXpa's current search path with the `add path` command. You can display CXpa's current search path with the `info` command.

Examples

The following example shows how to use the `path` command.

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

(Skipping lines of command output not relevant to this example.)

```
Current Search Path(s) : /usr/data
                         /usr/data/input
                         /usr/data/output
```

The `path` command in the above example sets CXpa's search path to the specified directories. The `info` command shows CXpa's search path.

Related Commands

`add path`
`info`
`list`
`list selectable`

quit

q

Exit CXpa.

Syntax

```
quit
```

Description

The `quit` command exits you from CXpa. If you issue the `quit` command during an active profiling session, CXpa notifies you and asks if you still wish to quit. If so, the performance data file (PDF) is closed, and the program you were profiling is terminated.

Examples

This section shows examples of the `quit` command.

```
(CXpa) quit
%
```

The `quit` command exits CXpa and returns you to your shell prompt.

```
(CXpa) quit
Program is paused, quit anyway? ([y]/n)?
```

In the above example, CXpa informs you that you have paused a process and asks if you still wish to quit. The default is yes (`[y]`). Press **RETURN** to accept the default and quit CXpa, or type `n` to cancel the quit operation.

Related Commands

`continue` `stop`

rerun

re

Run and profile your program again using the arguments specified in the last `run` command.

Syntax

`rerun` [*<i/o_redirection>*]

Parameter	Meaning
<i/o_redirection>	Redirects the program's standard input, output, or error from or to the specified file when you include one of the redirection operators (<, >, >>, >&, >>&).

Description

The `rerun` command runs the current executable with the arguments that you specified in the last `run` command.

NOTE: If you use a preexisting PDF, the performance data in the PDF will be overwritten with new data when you execute the `run` or `rerun` command. Use the `set pdf` command to specify a new name for the PDF to avoid overwriting an existing PDF.

NOTE: The `rerun` command does not retain file redirection specified with the `run` command.

While your program is running, you can type **CTRL-c** to pause profiling. Once paused, you can use the `analyze`, `stop`, or `continue` commands.

Examples

The following examples show how the `rerun` command works.

```
(CXpa) run 8 5
The arguments are 8 and 5.
(CXpa) rerun
The arguments are 8 and 5.
```

The above example shows that the `rerun` command uses the arguments specified in the last `run` command.

```
(CXpa) rerun < /usr/data/input
```

The above command runs your program using the arguments from the last `run` command and redirects standard input from the file /usr/data/input.

```
(CXpa) rerun > /usr/data/output
```

The above command runs your program, using the arguments from the last `run` command and redirects standard output to the file /usr/data/output.

```
(CXpa) rerun >& /usr/data/output
```

The `>&` in the above command tells CXpa to redirect standard output and standard error to the file /usr/data/output.

Related Commands

`continue` `run` `stop`

run

ru

Run and profile your program with the specified arguments.

Syntax

`run` [*<argument>* ...] [*<i/o_redirection>*]

Parameter	Meaning
<argument>	Specifies any number of command line arguments to the program you are profiling. Separate multiple arguments with spaces.
<i/o_redirection>	Redirects the program's standard input, output, or error from or to the specified file when you include one of the redirection operators (<, >, >>, >&, >>&).

Description

The `run` command starts profile data collection and executes the current executable. You specify this executable when you invoke CXpa.

As your program runs, CXpa collects metrics at the regions you selected for profiling with the `select` command. CXpa writes the data it collects to the performance data file (PDF). The default PDF name is *<executable>*.pdf. Use the `set pdf` command to specify another PDF name.

NOTE: If you use a preexisting PDF, the performance data in the PDF will be overwritten with new data when you execute the `run` or `rerun` command. Use the `set pdf` command to specify a new name for the PDF to avoid overwriting an existing PDF.

While your program is running, you can type **CTRL-c** to pause profiling. Once paused, you can use the `analyze`, `stop`, or `continue` commands.

Examples

The examples in this section show various ways to use the `run` command.

```
1.(CXpa) select routine all
2.(CXpa) collect wall_clock cpu
3.(CXpa) run
4.(CXpa) CTRL-c
   (Program execution is paused.)
5.(CXpa) stop
Process 24144 terminated with signal: SIGKILL.
```

The above scenario shows how the `run` command can be used. The line numbers are for reference only.

1. The `select routine all` command selects all routines in your program for profiling.
2. The `collect wall_clock cpu` command tells CXpa to collect wall clock time and CPU time metrics.
3. The `run` command begins program execution and data collection.
4. Program execution is paused by pressing **CTRL-c**.
5. The `stop` command terminates program execution.

```
(CXpa) run
(Program runs to completion, and any output is displayed.)
```

The command in the above example executes the current executable. No arguments are passed to the program. The program's output follows the `run` command.

```
(CXpa) run 18 364
```

The above command executes the program you are profiling and supplies 18 and 364 as arguments to the program.

```
(CXpa) run < /usr/data/input
```

The above command executes the program you are profiling and redirects standard input from the file /usr/data/input.

```
(CXpa) run > /usr/data/output
```

The above command executes the program you are profiling and redirects standard output to the file /usr/data/output.

```
(CXpa) run >& /usr/data/output
```

The `>&` in the above command tells CXpa to redirect standard output and standard error to the file /usr/data/output.

```
(CXpa) run > /usr/data/output >& /usr/data/errors
```

The above example redirects standard output and standard error to different files.

Related Commands

`continue`
`rerun`
`stop`

save executable

sa e

Save profile selection settings (instrumentation) to the current executable file or to a copy of the current executable.

Syntax

`save executable` [*<filename>*]

Parameter	Meaning
<filename>	An optional parameter that specifies the name of the new executable. The new file is an exact copy of the executable being profiled, except that it contains profiling instrumentation. If you enter the `save executable` command without specifying a file name, the profile selection settings are saved in the current executable.

Description

Use the `save executable` command to write profile selection settings (instrumentation) to the current executable file or to a copy of the current executable. This is referred to as *pre-instrumenting* an executable. Use pre-instrumented executables to:

- Profile with CXpa in environments that do not support CXpa controlling a child process.
- Profile applications in conjunction with tools such as MPI or PVM that replicate processes or with applications where another driver program or script starts the process.

  Refer to the "Profiling MPI and PVM applications with CXpa" online help topic or section of this book for information about using pre-instrumented executables when profiling message-passing applications with CXpa.
- Maintain separate copies of an executable with different regions and metrics selected for profiling. This makes it easier to generate multiple PDF files for comparison and analysis.
- Profile applications run with the `mpa` utility. Refer to the "CXpa and the `mpa` utility" online help topic or section of this book for more information.

You can then run the pre-instrumented executable file outside the control of CXpa and collect profiling data in a .pdf file for later analysis. You can also profile the new executable in the normal way (that is by invoking CXpa with the name of the executable and running it under CXpa).

When you enter the `save executable` command without specifying a file name, CXpa writes the instrumentation to the executable currently being profiled, without changing its name.

To write the instrumentation to a copy of the current executable, specify a file name with the `save executable` command. The resulting file is an exact copy of the executable being profiled, except that it contains the current source code region and metric selections (instrumentation). All source code correlation is maintained. The size and permissions of the executable do not change, but the time stamp on the executable does.

Refer to the "Using pre-instrumented executables" online help topic or section of this book for information about using pre-instrumented executables.

Examples

The following examples demonstrate how to use the `save executable` command.

```
% /opt/cxpa/bin/cxpa -nw a.out

       CONVEX Performance Analyzer


Type 'help' for help.
Reading executable a.out...
Selecting profile a.out.pdf...

(CXpa) select routine all
(CXpa) collect cpu wall_clock events
(CXpa) set events data_cache
(CXpa) save executable
(CXpa) instrumenting, enter <CTRL+C> to abort.
...10%...20%...30%...40%...50%...60%...70%...80%...90%..100%
(CXpa) instrumentation finished.
(CXpa) quit
```

In the previous example:

- The first command invokes CXpa from the shell with an executable file, `a.out`. The `-nw` (no windows) option specifies that CXpa's line mode (tty) interface will be used.

- The `select` routine all, `collect cpu wall_clock events`, and `set events data_cache` commands specify the regions and metrics selected for profiling.
- The `save executable` command writes the instrumentation directly to the executable file named `a.out`.
- The `quit` command exits CXpa.

When the instrumented executable `a.out` is run from the shell, it will generate a PDF file named a.out.pdf, which can then be analyzed with CXpa.

```
% /opt/cxpa/bin/cxpa -nw a.out

       CONVEX Performance Analyzer


Type 'help' for help.
Reading executable a.out...
Selecting profile a.out.pdf...

(CXpa) select routine all
(CXpa) collect cpu wall_clock
(CXpa) save executable new.a.out
(CXpa) instrumenting, enter <CTRL+C> to abort.
...10%...20%...30%...40%...50%...60%...70%...80%...90%..100%
(CXpa) instrumentation finished.
(CXpa) quit
```

In the above example:

- The first command invokes CXpa from the shell with an executable file, `a.out`. The `-nw` (no windows) option specifies that CXpa's line mode (tty) interface will be used.
- The `select routine all` and `collect cpu wall_clock` commands specify the regions and metrics selected for profiling.
- The `save executable new.a.out` command writes the information to a copy of the `a.out` executable named `new.a.out`.
- The `quit` command exits CXpa.

When the instrumented executable `new.a.out` is run from the shell, it will generate a PDF file named new.a.out.pdf, which can then be analyzed with CXpa.

Related Concepts

CXpa and the mpa utility
Using pre-instrumented executables

Related Commands

`analyze`
`collect`
`merge`
`run`
`select`
`set events`
`set visibility`

select

sel

Select source code regions in your program for profiling.

Syntax

select [*<region-type>*] [**all** | **in** *<routine-list>* | **at** *<line-number-list>*]

Parameter	Meaning
<region-type>	Specifies the type of region to select. Valid values are as follows: `routine`—Routines. `loop`—loops. `pregion`—parallel loops only.
all	Selects the specified *<region-type>* in all routines in your program. If you do not specify a region type, all available regions are selected.
in *<routine-list>*	Specifies the names of one or more routines whose regions you want to select. Separate routine names with a space. If two routines have the same name, prefix them with a file name followed by a colon: *<file-name>* : *<routine-name>*.
at *<line-number-list>*	Selects regions at *<line-number-list>*. *<line-number-list>* specifies one or more line numbers that contain regions you want to select. Separate line numbers with a space. To select a region that is not in the current source file, prefix the line number with a file name followed by a colon: *<file-name>* : *<line-number>*.

Description

Use the `select` command to select source code regions for profiling in all routines, in specific routines, or at specific lines in source files. You do not have to recompile your program to select or deselect regions for profiling.

Depending on the option(s) with which you compiled your source code, four types of source code regions can be selected for profiling:

- **Routines**—Routine regions are only available for profiling if your source code is compiled with an Exemplar compiler using the `+pa` option or instrumented for routine-level profiling with the `cxoi` utility.
- **All loops (including parallel loops)**—Loop regions are only available for profiling if your source code contains loops and is compiled with an Exemplar compiler using the `+pa` option at optimization level `+O2` or `+O3`.
- **Parallel loops only**—Compiler-generated parallel loops are only available for profiling if your source code contains loops and is compiled with an Exemplar compiler using the `+pa` option at optimization level `+O3 +Oparallel`.

When you invoke CXpa in line mode, all regions in your program are initially deselected for profiling, so you must select one or more of them with the `select` command. When you run your program, CXpa collects metrics only at the regions selected for profiling.

Select all routine regions the first time you profile your program with the `select routine all` command. This provides a complete picture of your program's performance.

NOTE: The loop nesting level setting affects the number of loops selected for profiling. The default loop nesting level setting only selects loops at nesting level 0 (outermost loops) for profiling. Refer to the "Profile Selection dialog" or "`set visibility`" command online help topics or sections of this book for more information about loop nesting levels.

Refer to the "Profiling strategy" section of this book for guidelines for selecting regions and metrics and a step-by-step profiling strategy.

To list source code regions that can be selected for profiling, use the `list selectable` command.

Examples

The examples in this section show several ways to use the `select` command.

```
(CXpa) select routine all
```

The above command selects all routines in your program for profiling.

```
(CXpa) select pregion in SUB2
```

In the above example, all parallel loops within the currently specified loop nesting level range in routine SUB2 are selected for profiling.

```
(CXpa) select pregion at 9
```

The above command selects the parallel loop at line 9 in the current source file for profiling.

```
(CXpa) select loop in sub2 init display
```

The above command selects all loops that fall within the currently specified loop nesting level range in routines sub2, init, and display for profiling.

```
(CXpa) select at 82
(CXpa) list selectable SUB4
      r       68   SUBROUTINE SUB4
     l        76   DO I = 1,100,2
     l        79      DO WHILE (K .LT. 10)
     L P      82         DO J = 1,20,1
```

In the above example, the `select at 82` command selects all source code regions at line 82 of the current source file for profiling.

The `list selectable SUB4` command lists the regions that can be selected for profiling in the routine named SUB4. The letters to the left of the line numbers in the `list selectable` output represent available regions for collection. The uppercase L and P indicate that line 82 contains loop and parallel loop regions that are selected for profiling.

Related Commands

`deselect`
`list selectable`
`run`
`set pdf`
`set visibility`

Related Topics

Profiling strategy

set events

set e

Specify the type of events to collect at source code regions selected for profiling.

Syntax

`set events` <*event*(s)> [`read_only`] [`write_only`]

Parameter	Meaning
<*event(s)*>	Specifies the type of event or events to collect. The number and type of events that can be collected per program run differ according to machine architecture:.

- On Exemplar S2000 and X2000 systems, you can collect one type of off-processor event per program run.
- On SPP1600 systems, you can collect one type of off-processor events and one set of on-processor events per program run.
- On SPP1200 systems, you can collect one set of on-processor events per program run.

Exemplar S2000, Exemplar X2000, and SPP1600 Series off-processor events

Off-processor events on Exemplar S2000, Exemplar X2000, and SPP1600 Series systems are monitored by event counters on the processor agent chip. Valid off-processor events parameters for these architectures are as follows:

`local_misses`— Configures off-processor event counters to monitor the number of times that data not found in the processor cache was found in local memory (memory allocated to that processor's hypernode). Requests for memory on the same hypernode use the hypernode crossbar.

`remote_misses`—Configures off-processor event counters to monitor the number of times that data not found in the processor cache was found in remote memory (memory allocated to a different hypernode). Requests for memory on other hypernodes use the CTI rings.

local_and_remote_misses—Configures off-processor event counters to collect the number of times that data not found in the processor cache was found in local and remote memory, combined.

By default, off-processor event counters monitor events during reads and writes.

On SPP1600 Series systems, you can also use one or both of the following parameters of the `set events` command to specify whether the off-processor event that you selected is collected for events caused by read operations, write operations, or both:

read_only

The `read_only` parameter can only be specified for `local_and_remote_misses`, `remote_misses`, or `local_misses` on SPP1600 Series systems. This option tells CXpa to monitor these events during read operations. A *read miss* is a data cache miss caused by a load.

write_only

The `write_only` parameter can only be specified for `local_and_remote_misses`, `remote_misses`, or `local_misses` on SPP1600 Series systems. This option tells CXpa to monitor these events during write operations. A *write miss* is a data cache miss caused by a store or a data cache miss caused by a load and clear.

SPP1200 and SPP1600 Series on-processor events

SPP1200 and SPP1600 Series systems use HP PA-RISC 7200 processors which have on-processor event counters. These can be configured to collect one of the following sets of events per program run using the parameters listed below:

data_cache—Configures on-processor event counters to collect total data cache access counts, miss counts, and latency.

instruction_cache—Configures on-processor event counters to collect total instruction cache miss counts and latency.

instruction_counts—Configures on-processor event counters to collect completed instruction counts and CPU clock cycles.

tlb_misses—Configures on-processor event counters to collect data and instruction TLB (translation lookaside buffer) misses.

Description

The `set events` command specifies the collection of events at the source code regions of your program you have selected for profiling. Each use of this command replaces the values previously set.

For more information about event metrics on Exemplar S2000, Exemplar X2000, SPP1200 Series, and SPP1600 Series architectures, refer to the "Introducing metrics" and "Selecting metrics in line mode" online help topics or sections in this book.

NOTE: Unless you have selected source code regions to profile with the `select` command and specified event collection with the `events` parameter of the `collect` command, the `set events` command will not cause any events to be collected.

Examples

The following examples show various ways to use the `set events` command.

```
1. (CXpa) select routine all
2. (CXpa) collect cpu wall_clock events
3. (CXpa) set events local_misses
```

The above sequence of commands shows how to use the `set events` command in combination with the `select` and `collect` commands to select regions for profiling and metrics to collect. The line numbers are for reference only.

1. The `select routine all` command selects all routines in your program for profiling.
2. The `collect` command specifies data collection for CPU time, wall clock time, and events. You must define the types of events to collect with the `set events` command.
3. The `local_misses` parameter of the `set events` command is valid for Exemplar S2000, X2000, and SPP1600 Series machines. The `set events local_misses` command specifies that you want to collect the number of times that data not found in the processor cache was found in memory allocated on the processor's hypernode.

```
(CXpa) set events remote_misses write_only
```

Remote misses are specific to multinode Exemplar X2000 and SPP1600 Series systems. The above command tells CXpa to collect the number of times that data not found in the processor cache was found in remote memory (memory allocated to a different hypernode) for the profiled regions of your program.

The `write_only` parameter specifies that remote misses will only be collected during write operations. The `write_only` parameter is only valid for SPP1600 Series systems.

```
(CXpa) set events instruction_counts remote_misses
```

The above example is specific to SPP1200 and SPP1600 Series systems. The `instruction_counts` parameter configures the on-processor event counters to collect completed instruction counts and clock cycles, while the `remote_misses` parameter configures the off-processor event counters to collect the number of data cache misses that were resolved remotely.

Related Commands

`collect`	`deselect`
`list selectable`	`run`
`select`	`set visibility`

Related Topics

Introducing metrics
Selecting metrics in line mode

set pdf

set p

Specify the name of the performance data file (PDF).

Syntax

set pdf *<filename>*

Parameter	Meaning
<filename>	Specifies the name of a PDF. The file name you specify should end in .pdf.

Description

The `set pdf` command sets the name of the performance data file (PDF) to be written to and/or read from during a CXpa session.

The PDF is a binary file that contains the performance data for a single run of your program. The data in a PDF is used to calculate the performance reports that appear when you use one of the `analyze` commands. If you do not set the PDF name, CXpa uses the default PDF name of *<executable>*.pdf.

Using the `set pdf` command, you can change the name of the PDF. You may want to do this for two reasons:

- **To prevent CXpa from overwriting an existing PDF**—If you invoked CXpa with the name of an executable, and there is an existing PDF that has the same name as the current PDF, CXpa overwrites all of the data in that PDF when you run your program. If you do not want this to occur, you must change the name of the PDF between runs of your program with the `set pdf` command.
- **To analyze a different PDF**—If you have invoked CXpa without an executable or with the name of a PDF only, you can use the `set pdf` command to select a PDF created in a previous CXpa session. You can then display performance reports for that PDF with the `analyze` command.

NOTE: If you invoke CXpa with an executable, you can only analyze a PDF created during the current CXpa session. To analyze a PDF created with a different executable or multiple PDF files, invoke CXpa without specifying an executable or with the name of a PDF file.

Examples

The following examples show how to use the `set pdf` command.

```
(CXpa) set pdf /usr/data/my.pdf
```

The previous command sets the name of the PDF to my.pdf. If a program is now run, performance data is collected in the file /usr/data/my.pdf. If a report is generated, CXpa analyzes the data in /usr/data/my.pdf.

```
(CXpa) set pdf ../profiles/prog1.pdf
```

The previous command sets the name of the PDF to prog1.pdf in the ../profiles directory.

Related Commands

`analyze`
`merge`
`run`
`select`
`set events`

Related Topics

Analyzing PDFs only

Performance data files

set subcomplex

set s

Specify the subcomplex you want to run your program on.

Syntax

`set subcomplex` *<subcomplex_name>*

Parameter	Meaning
<subcomplex_name>	Specifies the name of the subcomplex you want to run your program on. The default is the subcomplex from which you invoked CXpa.

Description

Use the `set subcomplex` command to specify the name of the subcomplex you want to run your program on. A *subcomplex* is a collection of processors and memory from one or more hypernodes of an Exemplar or SPP Series system (the subcomplex defines the boundary within which all threads belonging to a process execute).

The default value is the subcomplex from which you invoked CXpa. You need only use this command if you want to run your program on a different subcomplex. You must have the appropriate permissions to run the process on the specified subcomplex.

For more information on subcomplexes on Exemplar and SPP Series systems, refer to the scm(1) and scm(4) man pages or the *SPP-UX System Administration Guide* or contact the system administrator at your site.

To list available subcomplexes on your system or to list the current subcomplex, use the CXpa `info` command.

CXpa queries the system for available subcomplexes at start-up, so if a subcomplex is added during a CXpa session, it will not be visible to CXpa during that session.

Examples

The following example shows how to use the `set subcomplex` command.

```
(CXpa) info

       Current Executable : /home/smith/a.out
   Current Process State : Finished

(Skipping lines of output not relevant to this example)

       Current Subcomplex : System
  Available Subcomplex(s) : Chemistry, System
(CXpa) set subcomplex Chemistry
```

The last two lines in the output of the `info` command in the above example lists the current subcomplex (subcomplex your process is set to run on) and available subcomplexes on your system. The `set subcomplex Chemistry` command specifies that the process will now run on the Chemistry subcomplex, assuming you have the appropriate permissions for that subcomplex.

Related Commands

`info`

set visibility

set v

Set filters for profile data collection and/or analysis.

Syntax

```
set visibility
   { process | threads }
   {loop_levels <min> <max>] |
   loop_levels innermost <num_levels> }
```

Parameter	Meaning
Loop nesting level filters	
`loop_levels <min> <max>`	Specifies a fixed range of loop nesting levels to profile. <*min*> and <*max*> are positive integers specifying the minimum and maximum loop nesting levels to profile, respectively. Separate the <*min*> and <*max*> values with a space.
`loop_levels innermost <num_levels>`	Specifies the number of loop nesting levels to profile relative to each loop nest's innermost level. <*num_levels*> must be a positive integer.

The default loop nesting level is `loop_levels 0 0`.

Process/thread filters	
`process`	Enables you to display process-level performance data in reports.
`threads`	Enables you to display performance data for individual threads in reports.

The default setting is `process`.

Description

The `set visibility` command allows you to set the following filters for profile data collection and/or analysis:

- Loop nesting levels
- Process or thread visibility in CXpa reports

To view the current settings for any these filters, use the `info` command. These settings are explained in more detail in the sections that follow.

Loop nesting level filters

When loop regions are selected for profiling, only loops at nesting level 0 (after optimization) are selected by default. This default setting reduces the number of loops initially selected for profiling, thus minimizing the profiling intrusion that can be incurred when profiling nested loops with large iteration counts.

The `loop_levels` parameter of the `set visibility` command allows you to specify either a fixed range of loop nesting levels to profile or a number of nesting levels relative to each loop nest's innermost level. Loop nesting level settings apply to all loop regions selected for profiling.

CXpa automatically determines the number of loop nesting levels in your program and sets the maximum loop nesting levels and the maximum number of levels from the innermost loop appropriately. These nesting levels correspond to the loops that are created by the compiler, and may not correspond directly to your original source code due to optimizations performed.

Specifying a fixed loop nesting level range

The first time you profile loop regions in your program, use the default setting for loop nesting levels. On subsequent runs of your program, you can select different sections or "slices" of the loops within your program for profiling by specifying a minimum and maximum loop nesting level. When specifying a fixed ranged of loop nesting levels, you will generally want to set the minimum loop nesting loop level equal to the maximum loop nesting level. For example, the command

```
(CXpa) set visibility loop_levels 1 1
```

selects only loops at nesting level 1 (after optimization) for profiling.

Specifying relative loop nesting levels

Instead of choosing a fixed range of loop nesting levels for profiling, you can specify the number of loop nesting levels to profile relative to the innermost loop of each loop nest in your program.

- A relative setting of 0 means that only the loops at the innermost (deepest) level of each loop nest are selected for profiling.
- A relative setting of 1 means that only the loops at the two innermost nesting levels of each loop nest are selected for profiling.

  For example, if the innermost nesting level of a loop nest is 4 and a relative setting of 1 is specified, the loops at nesting levels 3 and 4 of that loop nest will be selected for profiling.

- A maximum setting is equivalent to selecting all loops at all loop nesting levels.

When a relative loop nesting level is specified, loops that are not part of a loop nest are also selected for profiling. When specifying a relative loop nesting level setting, you must use the `innermost` keyword with the `loop_levels` parameter of the `set visibility` command. For example:

```
(CXpa) set visibility loop_levels innermost 1
```

Process/thread filters

By default, performance data in reports (except for parallel region reports) is displayed for processes; by setting visibility for threads you can display performance data for on a thread-by-thread basis.

Examples

The following examples show how to use the `set visibility` command.

```
(CXpa) set visibility thread
```

The above command enables you to display performance data on a thread-by-thread basis in CXpa reports.

```
(CXpa) select loop in sub1 sub2 sub3
(CXpa) set visibility loop_levels 1 1
```

The above two commands illustrate how to use the `loop_levels` parameter of the `set visibility` command to specify a fixed loop nesting level range.

The `select loop` command selects all loops in routines `sub1`, `sub2`, and `sub3` for profiling. The `set visibility loop_levels 1 1` command further specifies that only loops at nesting level 1 (after optimization) in the specified routines are selected for profiling.

```
(CXpa) set visibility loop_levels innermost
```

The above command specifies a relative loop nesting level setting of 0, and selects only the innermost loop of each loop nest for profiling. As shown above, if you do not specify a number of levels with the `innermost` keyword, CXpa assumes a default value of 0. This setting also selects any loops that are not part of a loop nest.

```
(CXpa) set visibility loop_levels innermost 1
```

The above command specifies a relative loop nesting level setting. Only the two innermost loops of each loop nest will be selected for profiling.

Related Commands

`analyze`	`deselect`
`info`	`select`
`set events`	`set pdf`

Related topics

Profiling strategy	Reports

source

so

Execute a CXpa command file.

Syntax

source *<filename>*

Parameter	Meaning
<filename>	Specifies the name of a CXpa command file.

Description

The `source` command executes the CXpa commands in a specified CXpa command file. Sourcing a command file is useful when you find that you are repeating a series of commands during a profiling session.

A CXpa command file is a text file containing a list of CXpa commands. Each command must be on a separate line. Comment lines are denoted with the pound sign (#).

In line mode, CXpa exits and returns you to the shell prompt when it encounters a `quit` command; otherwise, CXpa returns you to the CXpa prompt when it reaches the end of the command file.

In batch mode, CXpa exits when it encounters either a `quit` command or the end of the file.

If CXpa encounters an error in a command file, CXpa stops executing the command file, displays an error message, and returns the CXpa prompt.

Examples

The following examples show how to use the `source` command and CXpa command files.

```
(CXpa) source my_cmd_file
```

The above command executes the CXpa commands in the file named my_cmd_file.

```
# This is a comment line.
select loop in sub4
run
analyze loop > sub4.report
quit
```

The above example shows the format of a CXpa command file.

Related Commands

```
analyze                    quit
run                        select
```

Related Topics

Using batch mode

stop

st

Stop profiling a paused program.

Syntax

```
stop
```

Description

The `stop` command stops data collection, saves the collected data, and terminates the process being profiled. Before you can use the `stop` command, you must pause the program first by pressing **CTRL-c**. Then enter the `stop` command.

When you stop or pause profiling, you can use the `analyze` command to view profile data that was collected up until your program was stopped.

When you display performance reports for a stopped program, the information in the reports is incomplete and only reflects the partial execution of your program.

Examples

The following example shows a scenario in which you would use the `stop` command. The line numbers are for reference only.

```
1.(CXpa) run
   (Program output is displayed.)
2.(CXpa) CTRL-c
   (Pressing CTRL-c pauses profiling.)
3.(CXpa) stop
4.(CXpa) analyze
   (Incomplete performance reports are displayed.)
```

The following lines explain the previous example by number:

1. The `run` command runs the program and initiates profile data collection.
2. **CTRL-c** pauses profiling. You cannot execute the `stop` command unless program execution is paused.
3. The `stop` command stops the profiling of your program.
4. The `analyze` command displays the profile data collected up to the point that the program was stopped.

Related Commands

`analyze` `continue` `run`

version

v

Display version and release information for CXpa.

Syntax

```
version
```

Description

The `version` command lists:

- CXpa's version number
- The release date of this version of CXpa
- The product number
- Copyright information

Examples

The output of the `version` command is shown in the following example.

```
(CXpa) version
Convex Performance Analyzer

Version : 3.5
Product # : 710-018415-002
Release Date : 6/20/96

Copyright (c) 1989-1996 Hewlett-Packard Company
All rights reserved.

Report problems by running the "contact" program
which will send electronic mail to the
HP-Convex Technical Assistance Center.
```

Related Commands

`info`

Messages

8

This chapter contains reference pages that explain CXpa messages. The messages are listed in order by identification number (ID). Each explanation contains the following sections:

- **Message**—The exact text of the message. Variable parameters are enclosed in angle brackets (<>) and are shown in italic type. For example, *<collection type>* is a variable that represents a type of source code region.
- **Type**—The message type, which can be one of the following:
  - `INFO`—A condition that is normal or expected processing under the given circumstances.
  - `WARNING`—A condition that might not be normal or expected, but is not severe enough to cause an error. CXpa continues to process your command after issuing the warning.
  - `ERROR`—A condition that prevents completion of the current CXpa command.
  - `FATAL`—A condition that prevents further execution of the process being profiled. Both the process and the CXpa session are terminated.
- **Explanation**—A more detailed explanation of the message, including possible actions to correct the situation.

To display online help for CXpa messages:

- In GUI mode, enter the message number in the Search field in the CXpa Help window.
- In line mode, enter the command `help` *<message_number>* at the CXpa prompt.

ID		Description
A1	Message	<*filename*> has been stripped of all symbolic information, unusable.
	Type	ERROR
	Explanation	Quit CXpa, recompile executable source code with one of the CXpa options, and then reinvoke CXpa with the new executable file.
A2	Message	Missing or unmatched <*token*>.
	Type	ERROR
	Explanation	Correct the quotation marks and reexecute the command.
A3	Message	Command syntax error - <*expecting or missing*> <*token*>.
	Type	ERROR
	Explanation	For more information about this command, refer to the online help system or the reference manual.
A4	Message	Command '<*string*>' is ambiguous. Could refer to: <*keyword list*>
	Type	ERROR
	Explanation	Complete your command futher so that it can be uniquely recognized.
A5	Message	Command line too complex.
	Type	ERROR
	Explanation	Try to simplify and reexecute the command.
A6	Message	Cannot open input file <*filename*>.
	Type	ERROR
	Explanation	Check to make sure the file exists and the permissions are set so that you can access it.
A7	Message	Cannot open output file <*filename*>.
	Type	ERROR
	Explanation	Check directory and file permissions to see if you are allowed write access.
A8	Message	Cannot have more than one redirection to/from <*filename*>.
	Type	ERROR
	Explanation	Reexecute the command (in line/batch mode) or GUI action (in X window mode) and redirect it to/from another file.
A9	Message	Cannot redirect to <*filename*>.
	Type	ERROR
	Explanation	Reexecute the command (in line/batch mode) or GUI action (in X window mode) and redirect it to/from another file.

ID		Description
A10	Message	Cannot start a new process while another process is still running.
	Type	ERROR
	Explanation	Wait until the current process completes or use the "stop" command (in line/batch mode) or the Abort button (in X window mode) to stop it.
A11	Message	Cannot access PDF *<filename>*.
	Type	ERROR
	Explanation	Check directory and file permissions to see if you have read/write access.
A12	Message	Cannot open file *<filename>*. *<errno description>*
	Type	ERROR
	Explanation	Check directory and file permissions to see if you have write access.
A13	Message	Cannot read from PDF. *<errno description>*.
	Type	ERROR
	Explanation	Check directory and file permissions to see if you have read access.
A14	Message	Cannot read from PDF.
	Type	ERROR
	Explanation	PDF may be corrupt.
A15	Message	Cannot write to PDF. *<errno description>*.
	Type	ERROR
	Explanation	Check directory and file permissions to see if you have write access.
A16	Message	Cannot write to PDF.
	Type	ERROR
	Explanation	PDF may be corrupt.
A17	Message	Profile data collected is corrupt, cannot create PDF.
	Type	ERROR
	Explanation	The performance data that CXpa has collected is corrupt and a performance data file cannot be created. Usually this occurs when the executable writes into memory that it did not allocate.
A18	Message	Executable required before using this command.
	Type	ERROR
	Explanation	Quit CXpa and invoke it with the name of an executable file. Then reexecute this command (in line/batch mode) or GUI action (in X window mode).

ID		Description
A19	Message	PDF <*filename*> is empty or corrupt.
	Type	ERROR
	Explanation	Regenerate the PDF again or try a different PDF.
A20	Message	Cannot create PDF <*filename*>.
	Type	ERROR
	Explanation	Check directory and file permissions to see if you have write access.
A21	Message	PDF <*filename*> cannot be analyzed with this executable.
	Type	ERROR
	Explanation	PDF and this executable are imcompatible. Invoke CXpa in analysis mode with the PDF file only or regenerate PDF with this executable.
A22	Message	PDF incompatible with this version of CXpa; regenerate PDF.
	Type	ERROR
	Explanation	PDF can no longer be analyzed; regenerate PDF.
A23	Message	Cannot read name of executable file from PDF.
	Type	ERROR
	Explanation	PDF may be corrupt.
A24	Message	Obsolete
	Type	ERROR
	Explanation	None.
A25	Message	Source file <*filename*> has changed since the PDF was created.
	Type	INFO
	Explanation	Line numbers reported in the Source Window may be incorrect. Regenerate the PDF to correct any line number inaccuracies.
A26	Message	Obsolete
	Type	ERROR
	Explanation	None.
A27	Message	Source file must be specified with this command.
	Type	ERROR
	Explanation	Reexecute this command with an existing source file (in line/batch mode) or use the Windows menu option, Source Code, to select a source file (in X window mode).

ID		Description
A28	Message	File <*filename*> is not an executable file.
	Type	ERROR
	Explanation	Check the directory and file permissions to see if you are allowed to execute it.
A29	Message	Error in command file <*filename*>.
	Type	ERROR
	Explanation	Edit the command file to correct any errors, then reexecute it.
A30	Message	PDF contains incomplete data.
	Type	ERROR
	Explanation	Regenerate the PDF or select a different PDF.
A31	Message	PDF contains no data.
	Type	INFO
	Explanation	The PDF was created with no regions or metrics selected. Select the desired regions and metrics and regenerate the PDF.
A32	Message	PDF, <*filename*>, has the same name as the executable file.
	Type	ERROR
	Explanation	PDF and executable files must be unique. Choose a different filename for the PDF using the "set pdf" command (in line/batch mode) or the File menu option, New PDF (in X windows mode).
A33	Message	Executable has been modified since startup, please restart CXpa.
	Type	ERROR
	Explanation	The executable (<*filename*>) has been modified since product invocation. Please quit and restart CXpa to read in the new executable.
A34	Message	Error reading from file <*filename*>.
	Type	ERROR
	Explanation	An error occured reading from file <*filename*>. Check to make sure the file exists and you have permissions to read it.
A35	Message	Error writing to <*filename*>
	Type	ERROR
	Explanation	An error occured writing to file <*filename*>. Check file and directory permissions and make sure there is disk space available.
A36	Message	Maximum number of threads reached.
	Type	ERROR
	Explanation	You have reached an internal limit on the number of threads a PDF can store.

ID		Description
A37	Message	PDF file is incompatible with the merge command; regenerate PDF.
	Type	ERROR
	Explanation	The merge command requires PDF files from this version of CXpa. PDF files generated from prior versions cannot be merged.
A38	Message	Cannot read source file <*filename*>.
	Type	ERROR
	Explanation	Source file is empty or you do not have permission to read the file.
A39	Message	PDF required before using this command.
	Type	ERROR
	Explanation	Choose a PDF with the "set pdf" command and reexecute the command (in line/batch mode) or choose a PDF with the File menu option, New PDF, and reexecute the GUI action (in X window mode).
A40	Message	Obsolete.
	Type	ERROR
	Explanation	None.
A41	Message	Obsolete.
	Type	ERROR
	Explanation	None.
A42	Message	Obsolete.
	Type	ERROR
	Explanation	None.
A43	Message	Obsolete.
	Type	ERROR
	Explanation	None.
A44	Message	Obsolete.
	Type	ERROR
	Explanation	None.
A45	Message	Directory <*filename*> does not exist.
	Type	ERROR
	Explanation	Reexecute the command (in line/batch mode) or GUI action (in X window mode) with a different directory name.
A46	Message	Obsolete.
	Type	ERROR
	Explanation	None.

ID		Description
A47	Message	Terminating with signal *<signal identifier>*. *<errno description>*
	Type	FATAL
	Explanation	CXpa has received a fatal signal and is terminating. There is no recovery from this condition.
A48	Message	No regions selected for profiling. No region-level analysis will be possible.
	Type	INFO
	Explanation	To obtain region-level metrics use the "select" and "collect" commands (in line/batch mode) or the Region Selection dialog (in X window mode).
A49	Message	Obsolete.
	Type	ERROR
	Explanation	None.
A50	Message	Obsolete
	Type	ERROR
	Explanation	None.
A51	Message	Obsolete
	Type	FATAL
	Explanation	None.
A52	Message	Obsolete
	Type	ERROR
	Explanation	None.
A53	Message	Obsolete
	Type	ERROR
	Explanation	None.
A54	Message	Obsolete
	Type	ERROR
	Explanation	None.
A55	Message	Obsolete
	Type	ERROR
	Explanation	None.
A56	Message	Obsolete
	Type	ERROR
	Explanation	None.

ID		Description
A57	Message	No profilable source code regions could be found. Recompile with current compiler with a CXpa option.
	Type	ERROR
	Explanation	Your executable contains no sources files compiled for profiling with CXpa. Recompile source file(s) and relink with one of the CXpa options to generate a new executable file.
A58	Message	Profiling routines incompatible or missing. Relink with current compiler or linker with a CXpa option.
	Type	ERROR
	Explanation	The profiling routines contained in your executable file are no longer compatible with the current version of CXpa or it was not compiled and linked for profiling with CXpa. Relink with one of the CXpa options to generate a new executable file.
A59	Message	No active PDF selected.
	Type	ERROR
	Explanation	Cannot create a new window until you select an active PDF. Select a PDF from the Active PDF list.
A60	Message	No loop, parallel, or block source code regions are available in <*filename*>.
	Type	INFO
	Explanation	The indicated PDF does not contain any profiling data for loop, parallel, or block source code regions. Only routine-level analysis is available for this PDF.
A61	Message	No <*metric level*> data to graph for <*metric region*> source code regions.
	Type	INFO
	Explanation	There was no data to graph for the source code region you specified. Try graphing a different metric or a different region.
A62	Message	PDF <*filename*> is already in the Analysis Control list; cannot add again.
	Type	ERROR
	Explanation	You can select this PDF from the list to create multiple windows.
A63	Message	Obsolete.
	Type	ERROR
	Explanation	None.
A64	Message	Obsolete.
	Type	ERROR
	Explanation	None.

ID		Description
A65	Message	Cannot save <*graph type*> to <*type identifier*> file. <*errno description*>.
	Type	ERROR
	Explanation	CXpa could not save the graph in the format you requested. Try saving it in a different format.
A66	Message	Exceeded maximum nesting level for CXpa command files.
	Type	ERROR
	Explanation	The command files you were executing contained more than 20 nesting levels. Redefine some of the command files to reduce the level of nesting.
A67	Message	No subcomplexes are available at this time.
	Type	ERROR
	Explanation	If needed, contact your system administrator for assistance in creating a new subcomplex.
A68	Message	Subcomplexes are not available on this architecture.
	Type	ERROR
	Explanation	None.
A69	Message	Could not access subcomplex <*filename*>.
	Type	ERROR
	Explanation	Verify that the specified subcomplex exists by using the "info" command (in line/batch mode) or Info Session dialog (in X window mode). If you are still unable to set the subcomplex, please contact your system administrator for assistance.
A70	Message	No regions selected for a customized 2D or 3D profile.
	Type	INFO
	Explanation	Bring up the Profile Customization dialog by pressing the Customize button and select one or more Profile Regions.
T1	Message	<*string*>.
	Type	ERROR
	Explanation	Product test message.

ID		Description
C1	Message	Cannot open executable file <*filename*>.
	Type	ERROR
	Explanation	The indicated executable file could not be opened. Check to make sure the file exists and that you have permission to access it. If the file does exist, it might contain data that has been corrupted. Try recompiling source file(s) with the appropriate compiler option. For CXdb compile and link with the -g option. For CXpa either compile with the +pa option or use CXoi (refer to the cxoi(1) man page) to preprocess object and library files; link with the +pa option. The -g and +pa options are incompatible with the -s option or the strip utility (refer to the strip(1) man page).
C2	Message	Obsolete.
	Type	ERROR
	Explanation	None.
C3	Message	Cannot find <*string*> for <*filename*>.
	Type	ERROR
	Explanation	The table containing the symbolic debugging or profiling information exists, but does not contain data in a format compatible with the current version of CXdb or CXpa. Try recompiling the file with the appropriate compiler option. For CXdb compile and link with the -g option. For CXpa either compile with the +pa option or use CXoi (refer to the cxoi(1) man page) to preprocess object and library files; link with the +pa option. The -g and +pa options are incompatible with the -s option or the strip utility (refer to the strip(1) man page).
C4	Message	Corrupt <*string*> for <*filename*> encountered.
	Type	ERROR
	Explanation	The table containing the symbolic debugging or profiling information exists, but does not contain data in a format compatible with the current version of CXdb or CXpa. Try recompiling the file with the appropriate compiler option. For CXdb compile and link with the -g option. For CXpa either compile with the +pa option or use CXoi (refer to the cxoi(1) man page) to preprocess object and library files; link with the +pa option. The -g and +pa options are incompatible with the -s option or the strip utility (refer to the strip(1) man page).
C5	Message	Pathname <*filename*> is not a valid directory.
	Type	ERROR
	Explanation	The pathname you specified is not a directory. Try a different pathname.

ID		Description
C6	Message	Symbolic debugging or profiling information format incompatible with this version of CXdb or CXpa.
	Type	WARNING
	Explanation	The format of the symbolic information generated by the compiler is incompatible with this version of CXdb or CXpa. Try recompiling source file(s) with the appropriate compiler option. For CXdb compile and link with the -g option. For CXpa either compile with the +pa option or use CXoi (refer to the cxoi(1) man page) to preprocess object and library files; link with the +pa option. The -g and +pa options are incompatible with the -s option or the strip utility (refer to the strip(1) man page).
C7	Message	Cannot find source file for <*filename*> in search path.
	Type	WARNING
	Explanation	Could not find the source file associated with your executable file. If you have moved the source file since compiling it, use the "add path" command to specify the new location of this file.
C8	Message	Obsolete.
	Type	ERROR
	Explanation	None.
C9	Message	Obsolete.
	Type	ERROR
	Explanation	None.
C10	Message	Ambiguous file name <*filename*>. Use a qualified pathname.
	Type	ERROR
	Explanation	The specified source file name is ambiguous. Use a path name to qualify the source file name.
C11	Message	Source file <*filename*> is out of date. Last modified <*date*>; compiled <*date*>.
	Type	WARNING
	Explanation	The specified source file has been modified since the executable was last compiled. This is only a warning; the specified version of your source file will be used.
C12	Message	Obsolete.
	Type	ERROR
	Explanation	None.

ID		Description
C13	Message	Cannot find file <*filename*> within current search path.
	Type	ERROR
	Explanation	Could not find the specified file in your current search path. Use the "add path" command to add directories to the search path.
C14	Message	Obsolete.
	Type	ERROR
	Explanation	None.
C15	Message	Symbolic data file <*filename*> is out of date. Last compiled <*date*>; data file created <*date*>.
	Type	ERROR
	Explanation	The indicated symbolic data file is out of date with the executable file. If you have modified this file since its last compilation, try recompiling the file. For CXdb compile and link with the -g option. For CXpa either compile with the +pa option, or use `cxoi` (refer to the cxoi(1) man page) to preprocess object and library files; link with the +pa option. The -g and +pa options are incompatible with the -s option or the `strip` utility (refer to the strip(1) man page).
C16	Message	Obsolete.
	Type	ERROR
	Explanation	None.
C17	Message	Cannot read line <*count*> from file <*filename*>.
	Type	ERROR
	Explanation	Could not read from the indicated source file. If you have modified this file since its last compilation, try recompiling the file. For CXdb compile and link with the -g option. For CXpa either compile with the +pa option, or use `cxoi` (refer to the cxoi(1) man page) to preprocess object and library files; link with the +pa option. The -g and +pa options are incompatible with the -s option or the `strip` utility (refer to the strip(1) man page).
C18	Message	Obsolete.
	Type	ERROR
	Explanation	None.
C19	Message	File <*filename*> is not a valid object file. <*errno description*>
	Type	ERROR
	Explanation	The specified file is not a valid object file, for the reason indicated. Try a different object file name.

ID		Description
C20	Message	Routine <*routine identifier*> was not compiled for or rejected for profiling.
	Type	ERROR
	Explanation	Recompile the source file containing the routine to be profiled. For CXpa either compile with the +pa option, or use `cxoi` (refer to the cxoi(1) man page) to preprocess object and library files; link with the +pa option. The +pa option is incompatible with the -s option or the strip utility (refer to the strip(1) man page).
C21	Message	Cannot <*operation*> alternate entries to routine <*routine identifier*>.
	Type	ERROR
	Explanation	Reexecute the command (in line/batch mode) or GUI action (in X window mode) using the main entry point to the routine.
C22	Message	Routine <*routine identifier*> is not unique, qualify using filename:routine.
	Type	ERROR
	Explanation	Reexecute the command (in line/batch mode) or GUI action (in X window mode) qualifying the routine with a filename. Use the format filename:routine.
C23	Message	Cannot <*operation*> <*collection type*> in routine <*routine identifier*>.
	Type	ERROR
	Explanation	Reexecute the command with a different routine name. Use the "list selectable" command (in line/batch mode) or the Region Selection dialog (in X window mode) to view the available source regions and their state of selection.
C24	Message	Line <*count*> in file <*filename*> was not compiled for or rejected for <*collection type*> profiling.
	Type	ERROR
	Explanation	Recompile the source file containing the line to be profiled. For CXpa either compile with the +pa option, or use `cxoi` (refer to the cxoi(1) man page) to preprocess object and library files; link with the +pa option. The +pa option is incompatible with the -s option or the strip utility (refer to the strip(1) man page).
C25	Message	No <*collection type*> to <*operation*> at line <*count*> in file <*filename*>.
	Type	ERROR
	Explanation	Re-execute the command with a different line number. Use the "list selectable" command (in line/batch mode) or the Region Selection dialog (in X window mode) to view the available source regions and their state of selection.

ID		Description
C26	Message	Cannot find routine <*routine identifier*>.
	Type	ERROR
	Explanation	Check to see if the source file containing this routine is within the search path(s) or reexecute the command (in line/batch mode) or GUI action (in X window mode) with a different routine name.
C27	Message	Cannot find file <*filename*>.
	Type	ERROR
	Explanation	Check that the file exists and that you specified its name correctly. If it is a source file, try using the "add path" command (in line/batch mode) or the Source Search Path dialog (in X window mode) to help CXpa locate the file.
D1	Message	Cannot read <*string*> from executable. <*string*> <*string*>
	Type	ERROR
	Explanation	A critical data structure could not be found in your current program. This prevents the backtrace command from working. You may either continue cautiously or try recompiling and relinking your application. For CXdb compile and link with the -g option. For CXpa either compile with the `+pa` option, or use `cxoi` (refer to the cxoi(1) man page) to preprocess object and library files; link with the `+pa` option. The -g and `+pa` options are incompatible with the -s option or the `strip` utility (refer to the strip(1) man page).
D2	Message	Cannot read <*string*> from executable. <*string*> <*string*>
	Type	ERROR
	Explanation	The current executable file is corrupt and cannot be executed. Try recompiling your executable. For CXdb compile and link with the -g option. For CXpa either compile with the `+pa` option, or use `cxoi` (refer to the cxoi(1) man page) to preprocess object and library files; link with the +pa option. The -g and `+pa` options are incompatible with the -s option or the `strip` utility (refer to the strip(1) man page).
D3	Message	Current executable is unreadable.
	Type	ERROR
	Explanation	The current executable cannot be read. No further work can continue on this executable. Try recompiling and relinking your application. For CXdb compile and link with the -g option. For CXpa either compile with the +pa option, or use `cxoi` (refer to the cxoi(1) man page) to preprocess object and library files; link with the +pa option. The -g and +pa options are incompatible with the -s option or the strip utility (refer to the strip(1) man page).

ID		Description
D4	Message	File <*filename*> is not a core file.
	Type	ERROR
	Explanation	The given file is not recognized as a core file. A core file is a file created when your program aborts unexpectedly. No other file can be a core file. If this file was created by an unexpected abort, then it may have been corrupted. Recreate a new core file by running your program again and be sure that you have enough disk space.
D5	Message	Cannot find symbolic support in current executable.
	Type	WARNING
	Explanation	The current executable does not have any symbolic support. Both debugging and performance analysis may proceed; however, many important features will not work such as loop instrumentation, source code to program counter correlation, and printing user variables. If you want these features, recompile your program with the appropriate compiler option. For CXdb compile and link with the -g option. For CXpa either compile with the +pa option, or use `cxoi` (refer to the cxoi(1) man page) to preprocess object and library files; link with the +pa option. The -g and +pa options are incompatible with the -s option or the strip utility (refer to the strip(1) man page).
D6	Message	Cannot read <*string*> from executable. <*string*> <*string*>.
	Type	ERROR
	Explanation	The current executable contains symbolic support; however, the specified table could not be read. Both debugging and performance analysis may proceed; however, many important features such as loop instrumentation, source code to program counter correlation, and printing user variables will not work correctly. If you want these features, recompile your program with the appropriate compiler option. For CXdb compile and link with the -g option. For CXpa either compile with the +pa option, or use `cxoi` (refer to the cxoi(1) man page) to preprocess object and library files; link with the +pa option. The -g and +pa options are incompatible with the -s option or the strip utility (refer to the strip(1) man page).
D7	Message	<*filename*> not properly linked. Relink with the Exemplar linker.
	Type	WARNING
	Explanation	The Exemplar linker generates table that are used by CXdb and CXpa to index the symbolic information in the executable or shared library. You must relink with the Exemplar linker to take advantage of symbolic information. Both debugging and performance analysis may proceed; however, many important features such as loop instrumentation, source code to program counter correlation, and printing user variables will not work correctly. For CXdb link with the -g option. For CXpa link with the `+pa` option. The -g and +pa options are incompatible with the -s option or the strip utility (refer to the strip(1) man page).

ID		Description
D8	Message	Executable has been preprocessed for exclusive use with xdb or DDE. Relink with the Exemplar linker.
	Type	WARNING
	Explanation	CXdb cannot read symbolic debugging information from an executable which has been preprocessed by the pxdb utility (the HP symbolic information preprocessor). Note that the DDE and xdb debuggers automatically invoke pxdb on executables that have not already been preprocessed, which overwrites the symbolic information in the executable. To correct this problem, relink the application with the Exemplar linker. For CXdb link with the -g option. For CXpa link with the +pa option. The -g and +pa options are incompatible with the -s option or the strip utility (refer to the strip(1) man page).
S1	Message	Cannot seek data in process memory. Errno: <*errno*>; address: <*address*>
	Type	ERROR
	Explanation	Operation could not be performed. There is no recovery.
S2	Message	Cannot read data from process memory. Errno <*errno*>; address <*address*>.
	Type	ERROR
	Explanation	Operation could not be performed. There is no recovery.
S3	Message	Partially read data from process memory
	Type	WARNING
	Explanation	Operation could not be performed. There is no recovery.
S4	Message	Cannot write data to process memory. Errno <*errno*>; address <*address*>.
	Type	ERROR
	Explanation	Operation could not be performed. There is no recovery.
S5	Message	Partially written data to process memory
	Type	WARNING
	Explanation	Operation could not be performed. There is no recovery.
S6	Message	Process is executing.
	Type	ERROR
	Explanation	Operation could not be performed. There is no recovery.
S7	Message	Process is not executing.
	Type	ERROR
	Explanation	Operation could not be performed. There is no recovery.

ID		Description
S8	Message	Process is not paused.
	Type	ERROR
	Explanation	Operation could not be performed. There is no recovery.
S9	Message	Process not detached.
	Type	ERROR
	Explanation	Operation could not be performed. There is no recovery.
S10	Message	Process is detached.
	Type	ERROR
	Explanation	Operation could not be performed. There is no recovery.
S11	Message	Process no longer exists.
	Type	ERROR
	Explanation	Operation could not be performed. There is no recovery.
S12	Message	Breakpoint already exists at addr <*address*>.
	Type	ERROR
	Explanation	Operation could not be performed. There is no recovery.
S13	Message	Cannot access executable <*filename*>. <*errno description*>.
	Type	ERROR
	Explanation	Check the permissions on this file.
S14	Message	fork() system call failed. <*errno description*>.
	Type	ERROR
	Explanation	Operation could not be performed. There is no recovery.
S15	Message	exec() system call failed.
	Type	ERROR
	Explanation	Operation could not be performed. There is no recovery.
S16	Message	Process cannot be terminated. <*errno description*>.
	Type	ERROR
	Explanation	Operation could not be performed. There is no recovery.
S17	Message	Process cannot be attached to. <*errno description*>.
	Type	ERROR
	Explanation	Operation could not be performed. There is no recovery.

ID		Description
S18	Message	Process cannot be detached. *<errno description>*.
	Type	ERROR
	Explanation	Operation could not be performed. There is no recovery.
S19	Message	Process cannot be continued. *<errno description>*.
	Type	ERROR
	Explanation	Operation could not be performed. There is no recovery.
S20	Message	Process cannot be closed. *<errno description>*.
	Type	ERROR
	Explanation	Operation could not be performed. There is no recovery.

ID	Description

Glossary

A

ALL
Acronym for Assembler, Loader, and Libraries.

B

bank conflict
An attempt to access a particular memory bank before a previous access to the bank is complete.

basic block
A linear sequence of machine instructions with a single entry and a single exit.

blocking factor, loop
Integer representing the stride of the outer strip of a pair of loops created by blocking.

C

cache
A small, high-speed buffer memory used in modern computer systems to hold temporarily those portions of the contents of the main memory that are, or are believed to be, currently in use. Cache memory is physically separate from main memory and can be accessed with substantially less latency. Exemplar S-Class and X-Class servers and SPP Series systems employ separate data and instruction cache memories.

cache, direct mapped
A form of cache memory which addresses encached data by a function of the data's virtual address.

On SPP1600 systems and S2000/X2000 servers, the cache address is identical to the least-significant 20 bits of the data's virtual address. This means cache thrashing can occur when the virtual addresses of two data items are an exact multiple of 1 Mbyte 920 bits) apart.

On SPP1200 Series computers, the main cache address is identical to the least significant 18 bits of the data's virtual address. Cache thrashing can therefore occur on SPP1200 machines when the virtual addresses of two data items are an exact multiple of 256 kbytes (18 bits) apart.

cache, fully associative

A form of cache memory that attempts to prevent encached data from being overwritten by storing an incoming element in a cache location that is determined using a hashing algorithm. The incoming element's virtual address is irrelevant. Unlike a direct mapped cache, a fully associative cache will never displace a cache line unless the cache is full. SPP1200 and SPP1600 Series machines use 2-kbyte, on-chip, fully associative assist caches to mitigate cache thrashing.

cache hit

A cache hit occurs if data to be loaded resides in the cache.

cache line

A chunk of contiguous data that is copied into a cache in one operation.

On S2000 servers, processor cache lines are 32 bytes (S2000 servers do not employ CTIcache lines). On X2000 servers, both processor cache lines and CTIcache lines consist of 32 bytes of data. When a processor cache miss occurs and data must be fetched from outside the processor cache, the requested data is brought in as part of a 32-byte cache line. When a CTIcache miss occurs, the requested data is brought into the CTIcache as part of a 32-byte cache line.

On SPP1600 systems, processor cache lines are 32 bytes, and CTIcache lines are 64 bytes (two contiguous processor cache lines).

cache misses, data

Number of times requested data was not found in the processor's data cache.

cache misses, instruction

Number of times that referenced instructions were not found in the processor's instruction cache.

cache thrashing

Cache thrashing occurs when two or more data items that are frequently needed by the program map to the same cache address. In this case, each time one of the items is encached, it overwrites another needed item, causing constant cache misses and impairing data reuse.

chunk
A unit of work in a loop that has been parallelized by the compiler consisting of a number of loop iterations.

chunk count
Number of chunks (packets of loop iterations) assigned to execute on a particular thread in a parallel loop.

clock cycle
The duration of the square wave pulse sent throughout a computer system to synchronize operations. The clock cycle time for SPP1600 Series systems, S2000 servers, and X2000 servers is 8.33 ns.

clock cycles per instruction
A measure of the efficiency of instruction scheduling. CXpa calculates this metric during analysis if instruction counts and clock cycles are collected. On SPP1200 Series systems, the maximum number of instructions that can be executed in a single clock cycle is 2. This means that the theoretical peak value for the clock cycles per instruction metric on this architecture is 0.5.

coherency (cache)
A term frequently applied to caches. If a data item is referenced by a particular processor on a multiprocessor system, the data is copied into that processor's cache and is updated there if the processor modifies the data. If another processor references the data while a copy is still in the first processor's cache, a mechanism is needed to ensure that the second processor does not use an outdated copy of the data from memory. The state that is achieved when both processors' caches always have the latest value for the data is called cache coherency. On Exemplar and SPP Series computers, an item of data may reside concurrently in several processors' caches.

compiler
Software that converts high-level language programs into machine code. Compilers for parallel systems improve performance of applications by automatically determining the best way to execute program procedures when several processors are available.

compiler parallel support library (CPSlib)
A library of thread management and synchronization routines that can be used to control parallelism on Exemplar systems. Using CPSlib requires you to manually control all aspects of parallelism, synchronization, and data partitioning.

concurrency factor, parallel

The ratio of CPU time to wall clock time. For parallel regions, this metric is an indicator of the speed-up achieved through parallelism. Values that approach *n*, where *n* is the number of processors used by your program, indicate good parallel concurrency.

CPU agent, processor agent chip (PAC)

The gate array on Exemplar and SPP Series systems that provides a high-speed interface between the pairs of PA-RISC processors in a functional block and the crossbar. Also called the processor agent chip (PAC).

CPU time

Time the CPU spent in the program (user CPU time), not including time waiting for I/O or running other programs. If a program can use multiple processors, the CPU time may be greater than the wall clock time.

CPU/wall clock

Ratio of CPU time to wall clock time. For serial regions, if the CPU/Wall clock ratio is high (approaches 1.0), the region is compute-bound. For parallel regions, this ratio corresponds to the concurrency factor, which is an indicator of the speed-up achieved through parallelization. Values that approach *n*, where *n* is the number of processors used by your program, indicate good parallel concurrency. For both parallel and serial regions, if the CPU/Wall clock ratio is low, this could indicate a performance bottleneck caused by I/O calls, system calls, or memory accesses.

crossbar

A switching device that connects the CPUs, banks of memory, and I/O controller on a single hypernode of an Exemplar or SPP Series system. Because the crossbar is nonblocking, all ports can run at full bandwidth simultaneously unless there is contention for a particular port.

CTIcache

A partition of physical memory that exists on each hypernode and is used to store copies of global data fetched from other hypernodes.

CTI (Coherent Toroidal Interconnect) ring

The ring interconnect that connects all the hypernodes of a multihypernode Exemplar or SPP Series system together in a ring topology. While the CTI ring is derived from the IEEE SCI standard, complete compatibility is sacrificed to provide lower latencies.

cxoi utility

The Exemplar object file instrumentor (`/opt/cxpa/bin/cxoi`, a separate utility shipped with CXpa) is used to instrument object and archive library files produced by any PA-RISC targeting compiler. This includes executables compiled with HP C, C++, or Fortran 77 compilers. Only routines are exposed for profiling with this utility. The `cxoi` utility only *enables* selection of routine regions for profiling in object and archive library files; it does not automatically select these regions for profiling. Region selection is done using CXpa.

D

data cache

A small cache memory with a one-clock cycle time. This cache holds prefetched and current data. On SPP1600 systems, processors have a 2-kbyte on-chip cache and a 1-Mbyte off-chip cache. On Exemplar S2000 and X2000 servers, processors have 1-Mbyte off-chip caches.

data cache hit

A data cache hit occurs if data to be loaded resides in the processor's data cache.

data cache hit rate

The percentage of total data cache accesses that were data cache hits. If the data cache hit rate is low, this indicates cache thrashing. CXpa uses the following formula to calculate the data cache hit rate:

$$data\_cache\_hit\_rate = \frac{total\_data\_cache\_accesses - data\_cache\_misses}{total\_data\_cache\_accesses} \times 100$$

data cache miss

A data cache miss occurs if data to be loaded does not reside in the processor's data cache.

data TLB misses, DTLB misses

Data translation lookaside buffer misses. This metric represents the number of times the address translation from virtual to physical memory for data to be referenced was not found in the translation lookaside buffer (TLB). The TLB is a cache of virtual-to-physical memory address translations for the most recently referenced page table entries. For SPP1600 Series systems, the TLB has 128 entries; on Exemplar S2000/X2000 servers, the TLB has 96 entries.

dynamic selection
The process by which the compiler chooses the appropriate clone of a code section (typically, a loop) at runtime, in order to achieve the lowest execution time.

E

event
Hardware events such as locally and/or remotely resolved cache misses during reads and/or writes; data and instruction cache accesses and misses; instruction counts and clock cycles, and data and instruction TLB (translation lookaside buffer) misses. Using CXpa, you can configure on-processor and off-processor hardware event counters on SPP Series systems to collect hardware event metrics. Hardware events that can be collected vary according to machine architecture.

event counts
The number of times an event, such as a data or instruction cache miss, occurred.

exclusive times and metrics
Exclusive times and metrics reported by CXpa do not include time spent in or metrics collected for called (child) routines.

G

globally shared memory (GSM)
A feature of some parallel systems that lets all processors transparently access any location in main system memory. This lets a single, scalable operating system control the resources of the system. The arrangement presents a familiar environment to developers and users. On Exemplar Servers and SPP Series systems, globally shared memory is distributed among hypernodes, but any hypernode's memory is accessible from any processor on any hypernode. This type of memory is sometimes referred to as shared virtual memory or global virtual memory.

H

hypernode

In Exemplar and SPP Series systems, a set of up to 16 processors and physical memory organized as a symmetric multiprocessor (SMP) running a single image of the operating system microkernel. An Exemplar or SPP Series system consists of one or more hypernodes, with a high speed CTI ring connecting the hypernodes. When discussing multidimensional parallelism or memory classes, hypernodes are generally called nodes.

I

inclusive times and metrics

Inclusive times and metrics reported by CXpa include time spent in or metrics collected for called (child) routines.

instruction cache

On SPP 1600 systems and S2000/X2000 servers, a 1-Mbyte cache memory with a 1-clock cycle access time. This cache holds prefetched instructions and permits the simultaneous decoding of one instruction with the execution of a previous instruction. On SPP1200 systems, it is 256 kbytes in size with a 1-clock cycle access time.

instruction cache miss

An instruction cache miss occurs when a memory reference for an instruction cannot be resolved in the processor cache.

instrumenting, instrumentation

Before you can profile your program with CXpa, it must be prepared, or instrumented for profiling. When you compile your program with a CXpa profiling option or instrument your program with the `cxoi` utility, instructions are inserted into the executable file that enable CXpa to gather performance data during execution of your program. When you select regions and metrics for profiling, CXpa inserts calls to the timing and profile data collection routines for the selected regions and metrics.

ITLB misses, instruction TLB misses
Instruction translation lookaside buffer misses. This metric represents the number of times the address translation from virtual to physical memory for an instruction was not found in the translation lookaside buffer (TLB). The TLB is a cache of virtual-to-physical memory address translations for the most recently referenced page table entries. On SPP1600 Series systems, the TLB has 120 entries; on Exemplar S2000/X2000, the TLB has 96 entries.

J

join
The synchronized termination of parallel execution by spawned tasks or threads.

L

latency time

As measured by CXpa, the amount of time spent accessing memory to locate data or instructions not found in the processor's data or instruction cache.

latency/counts
Average latency time per event for a code region. For example, if total data cache miss counts are collected, then the event latency/counts value computed by CXpa during analysis represents the average amount of time it took to resolve a data cache miss.

latency/CPU
Ratio of event latency to CPU time for a code region, expressed as a percentage. CXpa calculates this metric during analysis if data or instruction cache miss events, latency, and CPU time are collected. When this percentage is high, it means that the program spent the majority of its CPU time in the region "reaching" for data or instructions that were not encached rather than computing with encached data or instructions.

load
An instruction used to move the contents of a memory location into a register.

locality of reference
An attribute of a memory reference pattern that refers to the likelihood of an address of a memory reference being physically close, in terms of the number of clock cycles necessary to access it, to the CPU making the reference.

localization
Data localization. Optimizations designed to keep frequently used data in the processor data cache, thus eliminating the need for more costly CTIcache or main memory accesses.

locally resolved cache misses
Cache misses that are resolved by using the hypernode crossbar to access memory found on the same hypernode as the processor; the CTI rings are not used.

loop blocking
A loop transformation that strip-mines and interchanges a loop to provide optimal reuse of the encachable loop data.

loop distribution
The restructuring of a loop nest to create simple loop nests. Loop distribution creates two or more loops, called distributed parts, which can serve to make parallelization more efficient by increasing the opportunities for loop interchange and isolating code that must run serially from code that can be parallelized. It can also improve data localization and other optimizations.

loop interchange
The reordering of nested loops. Loop interchange is generally done to increase the granularity of the parallelizable loop(s) present or to allow more efficient access to loop data.

loop replication
The process of transforming one loop into more than one loop to facilitate an optimization. The optimizations that replicate loops are `IF-DO` and `if-for` optimizations, dynamic selection, loop unrolling, unroll and jam, and loop blocking.

loop strip mining
The transformation of a single loop into two nested loops. Conceptually, this is how parallel loops are created by default. A conceptual outer loop advances the initial value of the inner loop's induction variable by the parallel strip length. The parallel strip length is based on the trip count of the loop and the amount of code in the loop body. Strip mining is also used by the data localization optimization.

loop unrolling

A loop transformation performed by the compiler that involves increasing a loop's step value and replicating the loop body, with each replication appropriately offset from the induction variable so that all iterations are performed, given the new step. Unrolling can be total or partial.

Total unrolling involves eliminating the loop structure completely, replicating the loop body a number of times equal to the iteration count, and replacing the iteration variable with constants. Total unrolling is generally performed on loops with small iteration counts.

Partial unrolling is performed on loops with larger or unknown iteration counts. It retains the loop structure, but replicates the body a number of times equal to the unrolling factor, and adjusts references to the iteration variable accordingly.

M

memory bank conflict

An attempt to access a particular memory bank before a previous access to the bank is complete.

message passing

A type of programming in which program modules (often running on different processors or different hosts) communicate with each other by means of system library calls that package, transmit, and receive data. All message passing library calls must be explicitly coded by the programmer.

MIPS

Millions of instructions per second.

MIPS rate

CXpa uses the following formula to calculate the MIPS rate:

$$MIPS\_rate = \frac{Number\_of\_instructions\_completed}{Wall\_clock\_time\_in\_sec}$$

The theoretical peak MIPS rate for Hewlett-Packard PA-RISC 7100/7200 processors with a 10-ns clock cycle at a clock rate of 100 MHz is 200 MIPS.

MPI (message-passing interface)

A message-passing and process control library. For information on the Hewlett-Packard implementation of MPI, refer to the *HP MPI User's Guide*.

N

NUMA
Nonuniform memory access. This term describes systems in which accessing different types of memory (for example, memory on the current hypernode or memory remote to the current hypernode) results in nonuniform access times.

O

off-processor events
Off-processor events are monitored by event counters (when available) on the processor agent chip. On Exemplar S2000, X2000, and SPP1600 Series systems, these event counters can be configured to monitor the number of data cache miss events that are resolved locally and/or remotely (remote miss events imply use of the CTI rings; local miss events do not use the CTI rings) and whether those events occurred due to reads (loads) or writes (stores).

on-processor events
On-processor events are monitored by event counters (when available) located on the HP PA-RISC processors used in SPP1600 Series systems. These event counters monitor events from the processor's point of view only.

optimization
The refining of application software programs to minimize processing time. Optimization takes maximum advantage of a computer's hardware features and minimizes input/output traffic and idle processor time.

optimization level
The degree to which source code is optimized by the compiler. The Exemplar Fortran 77, C, and C++ compilers have five levels of optimization: level `+O0`, `+O1`, `+O2`, `+O3`, and `+O4`. The default level for Exemplar compilers is `+O0`.

oversubscription
In the context of parallel threads, a process attribute that permits the creation of more threads within a process than the number of processors available to the process.

P

parallel optimization
The transformation of source code into parallel code (parallelization) and restructuring of code to enhance parallel performance.

parallelization
The process of transforming serial code to a form of code that can run simultaneously on multiple processors while preserving semantics. At optimization level `+O3 +Oparallel`, the Exemplar compilers automatically parallelize loops in your program and recognize compiler directives and pragmas with which you can manually specify parallelization of loops, tasks, and regions.

parallelization, loop
The process of splitting a loop into several smaller loops, each of which operates on a subset of the data of the original loop, and generating code to run these loops on separate processors in parallel.

PDF (performance data file)
A binary file CXpa creates that contains the performance data for a single run of your program. The data in a PDF is used to create 2D and 3D profile graphs and analysis reports.

pre-instrumented executable
You can write profile selection settings (instrumentation) to the current executable file or to a copy of the current executable. This is referred to as *pre-instrumenting* an executable. The resulting executable file can be run outside the control of CXpa and profiling data collected in a performance data file (PDF file) for later analysis.

process
A collection of one or more execution streams within a single logical address space; an executable program. A process is made up of one or more threads.

PVM
Parallel virtual machine. A message-passing and process control library available on Exemplar SPP Series systems.

R

read
A memory operation in which the contents of a memory location are copied and passed to another part of the system.

read miss
A data or instruction cache miss that occurred as a result of a read operation.

reduced instruction set computer (RISC)
An architectural concept that applies to the definition of the instruction set of a processor. A RISC instruction set is an orthogonal instruction set that is easy to decode in hardware and for which a compiler can generate highly optimized code. The PA-RISC processor used in SPP Series computers employs a RISC architecture.

remotely resolved cache misses
Cache misses that were resolved by using the CTI rings to access memory on another hypernode. Accessing memory on other hypernodes using the CTI rings takes significantly longer than accessing memory on the same hypernode via the hypernode crossbar.

S

Scalable Coherent Interface (SCI)
Scalable Coherent Interface. This is defined by IEEE standard 1596-1992. The interface is physically defined as a pair of 18-bit, differential ECL, unidirectional links which are clocked at 250 MHz. Each link provides 16 bits of data with two control signals. Data is sampled on both the rising and falling edges of the clock. This interface provides the basis for the CTI rings used in Exemplar and SPP Series systems; however, total compatibility with the standard has been sacrificed to provide increased performance.

Scalable Parallel Processor (SPP)
Scalable parallel processor systems combine the accessibility and proven technology of single-processor workstations with symmetric multiprocessors' ease of programming and ability to address large problems. SPPs can be scaled or expanded to serve additional users when necessary. SPP systems are capable of handling hundreds of high-performance processors. Compute capacity, memory, and other subsystems also scale when necessary. A key design goal for SPP systems is to enable performance to increase linearly with respect to its number of processors.

spawn
To activate existing threads.

store
An instruction used to move the contents of a register to memory.

subcomplex
In Exemplar and SPP Series systems, a logical entity that provides control over the allocation of processors and physical memory to different applications and users.

symmetric multiprocessor (SMP)
A multiprocessor computer in which all the processors have equal access to all machine resources. Symmetric multiprocessors have no master or slave processors; the operating system runs on any or all of the processors.

system subcomplex
In an Exemplar or SPP Series system, a subcomplex that is automatically created at boot time by the operating system to run system processes, including `init` and processes spawned by `init`. The Subcomplex Manager utility (`scm`) will not allow users to destroy this subcomplex, nor remove the last processor from this subcomplex.

T

thread
An independent execution stream that is executed by a CPU. One or more threads, each of which can execute on a different CPU, make up each process. Memory, files, signals, and other process attributes are generally shared among threads in a given process, enabling the threads to cooperate in solving the common problem. Threads are created and terminated by instructions that can be automatically generated by Exemplar compilers, inserted by adding compiler directives to source code, or coded explicitly using library calls or assembly-language.

thread identifier (thread ID)
An integer identifier associated with a particular thread. See thread identifier, kernel (ktid) and thread identifier, spawn (stid).

thread identifier, kernel (ktid)
A unique integer identifier (not necessarily sequential) assigned when a thread is created.

thread identifier, spawn (stid)
A sequential integer identifier associated with a particular thread that has been spawned. stids are only assigned to spawned threads, and they are assigned within a spawn context; therefore, duplicate stids may be present amongst the threads of a program, but stids are always unique within the scope of their spawn context. stids are assigned sequentially and run from 0 to one less than the number of threads spawned in a particular spawn context.

TLB (translation lookaside buffer)
A hardware structure in each CPU in an SPP Series system that contain information necessary to translate a virtual memory reference to a physical page and to validate memory accesses. The TLB contains entries that represent address translations from virtual to physical memory for the most recently used page table entries. On SPP1600 Series systems, the TLB contains 120 entries; on Exemplar S2000/X2000 servers, the TLB has 96 entries.

TLB data and instruction misses
Number of times the address translation from virtual to physical memory for data or instructions to be referenced was not found in the translation lookaside buffer (TLB).

W

wall clock time
Time to solution or elapsed time for a program, including disk accesses, memory accesses, input/output, and operating system overhead. Compare with CPU time.

write
A memory operation in which a memory location is updated with new data.

write miss
A data or instruction cache miss that occurred as a result of a read operation.

Index

Symbols

% CPU field in events reports 155
  discussed, for parallel loop regions 147
% Hits field in reports 155
+O1 compiler optimization option 17
+O2 compiler optimization option 17
+O3 compiler optimization option 17, 18
+Oparallel compiler optimization option 17, 18
+pa compiler option for profiling with CXpa 17
.cxpainit start-up file 311
=> symbol in Source Code window 33, 280
2D Profile button 32, 191, 207
2D profile graphs
  customizing 255
  displaying associated source code 176
  fixed metric sorting 273
  saving to a file 267
  sorting options 273
  X defaults 289
  zoom (scaling) options 294
2D Profile window 175
3D Profile button 32, 191
3D profile graphs
  customizing 255
  displaying associated source code 182
  fixed metric sorting 273
  rotating 182
  saving to a file 267
  sorting options 273
  X defaults 289
  zoom (scaling) options 294
3D Profile window 181

A

abbreviations, command 297
abbreviations, loop optimization, described 157
Abort button 206
active PDF 52
  selecting in GUI mode 188
  selecting, in line mode 361
add path command 299
adding a directory to CXpa's search path 283
All regions button (Analysis Report window) 195
All/None buttons (Profile Selection dialog) 251
Analysis Control window 51, 187
Analysis Report window 193
analyze command 301
analyzing PDFs only 51
  in GUI mode 51
  in line mode 53
annotations
  in Source Code window 280
  source code region 83
app-defaults file, location of 289
archive libraries
  HP PA-RISC
    instrumenting with cxoi 23
arguments, program
  specifying in GUI mode 204
  specifying on command line 70
assistance, technical xvi
associated documents xiv
audience xi
average clock cycles per instruction
  defined 103
  discussed 167
average data cache miss latency (latency/counts) 105
average MIPS rate, discussed 103
Avg clock cycles field in reports 155
Avg MIPS rate field in reports 155
axis display controls
  2D Profile window 178
  3D Profile window 184

B

B:*n* abbreviation for loop blocking 157
bank conflict, defined (glossary) 395
basic block
  defined (glossary) 395
batch mode
  instructions for using 69
  overview 9
  sample shell script for batch execution 71
blocking factor, loop
  defined 395
Browse buttons (Help window) 74, 218
buttons
  2D Profile 191, 207
  3D Profile 191
  Abort 206
  All/None (Profile Selection dialog) 251
  Call Graph 191
  Change File (in Source Code window) 280
  Continue 206
  Pause 206
  Profile Selection 206
  Report 191

Reset Zoom
2D Profile window 178
3D Profile window 184
Show All
2D Profile window 178
3D Profile window 184
Start 206
SubComplex Selection 206
Zoom-In
2D Profile window 178
3D Profile window 184
Zoom-Out
2D Profile window 178
3D Profile window 184

C

-c compiler option 19
C++ programs
compiling for use with CXpa 17
cache
defined 395
direct-mapped, defined 395
fully associative, defined 396
cache hit, defined 396
cache line, defined 396
cache misses
collected on S2000/X2000/SPP1600 architectures 109
collected on SPP1200/1600 architectures 110
data 104
instruction 106
locally resolved, defined 107
remotely resolved, defined 108
cache thrashing, defined 396
Call Counts report, for routines 162
call graph
creating 198
displaying associated source code 199
displaying for individual threads 287
report, creating 127
searching for routines in 213
Call Graph button 191
Call Graph window 197
Change File button (in Source Code window) 281
changing CXpa's search path
add path command 299
in GUI mode 283
path command 339
chunk counts
defined 144, 155
defined (glossary) 397
chunks
defined 102
defined (glossary) 397
Chunks column in Parallel region reports 155
clock cycle
defined 397
clock cycles per instruction
defined 103
defined (glossary) 397
coherency (cache), defined 397
collect command 307
collecting metrics
events, in line mode (set events command) 357
in GUI mode (Profile Selection dialog) 242
in line mode (collect command) 307
command files
example in batch mode 69
executing at CXpa start-up 313
executing with the source command 369
format 69
input using the -x option 69
command line editing 38
command syntax xiii
commands
abbreviations 297
add path 299
analyze 301
collect 307
continue 309
cxpa 311
deselect 319
help 321
info 325
introduction 297
list 327
list selectable 331
merge 335
path 339
quit 341
rerun 343
run 345
save executable 349
select 353
set events 357
set pdf 361
set subcomplex 363
set visibility 365
source 369
stop 371
version 373
compiler options
-c 19
for profiling 314
-G 18
optimization, related to profiling 17
+O0 17
+O1 17
+O2 17
+O3 17
-p 18

Compiler Parallel Support Library (CPSlib)
defined 397
compiling
and linking in one step 19
and linking separately 19
syntax 17
Computation report
for routines 162
for serial loops 133
concurrency
factor for parallel loop regions (CPU/Wall) 145
factor, defined 398
`contact` utility xvi
Contents button (Help window) 74, 218
Continue button 206
`continue` command 309
controlling execution
`continue` command 309
`rerun` command 343
`run` command 345
`stop` command 371
using buttons on Executable Manager window 206
CPSlib, defined 397
CPU Agent chip, defined 398
CPU time 101
defined (glossary) 398
CPU/Wall clock time 102
defined (glossary) 398
discussed, for parallel loop regions 145
for parallel regions (concurrency factor) 102
for serial regions (CPU utilization) 102
CPU/Wall field in reports 155
crossbar, defined 398
CTI ring, defined 104, 398
CTIcache, defined 398
current list file name, displaying
in GUI mode 225
using the `info` command 326
current PDF name, displaying
in GUI mode 223
using the `info` command 325
current PDF, selecting 188
Current Process State 221
listed in output of `info` command 325
current working directory, listed 225
customizing
2D and 3D profiles 255
reports 195, 255
X defaults 289
`cxoi` utility
defined 399
instructions for using 23
instrumenting object files and archive libraries 23
known problems 26
limitations 25
Cxpa app-defaults file 289
`cxpa` command 311
`CXPA` environment variable 316
.cxpainit start-up file 311

D

`D` optimization abbreviation 157
Data Cache Accesses report (SPP1200/1600 only)
for parallel loop regions 149
for routines 167
for serial loops 137
data cache hit
defined 104, 399
data cache hit rate
defined 104, 399
formula for calculating 167
data cache miss
defined 104, 399
data cache, defined (glossary) 399
data localization, defined 403
`data_cache`
parameter of `set events` command 120, 358
`deselect` command 93, 319
dialogs
Filter Profile 209
Filter Report 211
Find Routine 213
Info Executable 221
Info PDF 223
Info Session 225
Merge PDFs 227
New PDF 231
Open PDF 235
Product Information 239
Profile Selection 86, 242
Region Subset Selection 255
Routine Detail 259
Save Executable As 263
Save Profile 267
Save Report 271
Sort 273
Source Code Selection 277
Source Search Path 283
Subcomplex Selection 285
Thread Selection 287
Zoom 293
Directories field in New PDF dialog 232
directory
adding to CXpa's search path 283, 299
current working directory, listed 225
removing from CXpa's search path 284
replacing CXpa's search path with the `path` command 339
documentation
associated documentation xiv
ordering information xvi
`Ds` optimization abbreviation 157

DTLB miss
  defined 399
dynamic call graph
  Call Graph window 197, 198
  creating 127, 198
  problems with uninstrumented routines 198
  report 127
Dynamic Call Graph report
  described 127
  displaying 129
  displaying for individual threads 129
  displaying in line mode (analyze command) 303
dynamic selection, defined 400

E

-e CXpa start-up option 70, 311
e profiling status 157
editing the command line 38
Elapsed Wall Time field 206
enabling event collection
  in line mode (collect command) 307
environment variables
  CXPA 316
  PROFDIR 56
ERROR message type 375
error messages
  displaying explanations for (in line mode) 322
event counts 105
events
  collected on Exemplar S2000/X2000 architectures 109
  collected on SPP1200/1600 architectures 110
  collected on SPP1600 Series architectures 109
  enabling event collection
    in GUI mode (Profile Selection dialog) 242
    in line mode using the collect command 307
  memory access events 107
  off-processor (S2000/X2000/SPP1600) 119
  on-processor (SPP1200/1600) 110, 120
  reports
    for parallel loop regions 146
    for routines 164
    for serial loops 134
  selecting
    in GUI mode 113
    in line mode using set events command 357
    using Profile Selection dialog 113
    using the collect and set events commands 117
  terms and definitions 103
Exclude Child/Inner Metrics button 209
exclusive, defined 400
Executable field
  on the Analysis Control window 191
  on the Executable Manager window 206
executable file
  compiling for profiling 18
  creation date, displaying in GUI mode 221
  creation date, displaying in line mode 325
  current, displaying in GUI mode 221
  displaying information about
    in GUI mode 221
    in line mode (info command) 325
  pre-instrumenting in line mode 349
  save executable command 349
Executable Manager window 203
Execution counts 102
execution, controlling
  continue 309
  rerun command 343
  run 345
  stop command 371
exiting CXpa
  from the X/Motif GUI 35
  quit command 341

F

FATAL message type 375
fields
  Elapsed Wall Time 206
  Executable 191, 206
  Filter 228, 232, 264, 268
  Finished State 191
  in CXpa reports 155
  Open PDF 191
  PDF 206
  Process State 206
  Program Arguments 206
  SubComplex 206
files
  .cxpainit (start-up file) 311
  .pdf 361
  executable
    compiling for profiling 18
    creation data, displaying 221
    displaying information in GUI mode 221
    displaying information in line mode 325
    pre-instrumenting 55
    pre-instrumenting in line mode 349
    save executable command 349
  object 19
    instrumenting with cxoi 23
  PDFs (performance data files) 49
    setting in GUI mode 49
    setting in line mode 50, 361
  search path for, setting in line mode 299
  source code
    list command 327
    selecting (GUI mode) 277
    Source Code window 279

viewing (in GUI mode) 279
viewing (in line mode) 327
Filter button 228, 232, 264, 268
Filter Profile dialog 209
Filter Report dialog 211
Find Routine dialog 213
Finished State field 191
fixed loop nesting levels 90, 366
fixed metric sorting in 2D/3D profile graphs 273
format of PDF listed
in GUI mode 223
in line mode 325

G

-G compiler option 18
g profiling status 157
getting help online (in line mode) 321
Go Back button (Help window) 74, 218
graphs
2D Profile 176
3D Profile 182
customizing 255
customizing X defaults 289
dynamic call graph 197
rotation in 3D Profile window 182
GUI interface 9
overview 9
profiling a program 204
guidelines for using CXpa 6

H

help command 321
Help window 217
buttons 218
creating 219
help, online
displaying
in GUI mode 218
in line mode (help command) 321
printing in ASCII 77
printing online help text 218
searching 219
using in GUI mode 73
Hs optimization abbreviation 157
hypernode, defined 106, 401

I

I optimization abbreviation 157
Include Child/Inner Metrics button 209
inclusive, defined 401
info command 325
Info Executable dialog 221
INFO message type 375
Info PDF dialog 223
Info Session dialog 225
instruction cache miss, defined 401
instruction cache, defined 401
instruction_cache
parameter of set events command 120, 358
instruction_counts
parameter of set events command 120, 358
Instructions Completed and Clock Cycles report (SPP1200/1600 only)
for parallel loop regions 150
for routines 167
for serial loops 138
instrumentation 401
Instrumentation Version
displaying in line mode (info command) 325
of executable 221
of PDF 224
Instrumenting Executable dialog 32
interfaces to CXpa 8
introduction
learning CXpa quickly 29
to CXpa and profilers 3
intrusion
discussed 41
minimizing 42
invoking CXpa
cxpa command 311
start-up options 311
with a PDF only (analysis-only mode) 187
ITLB miss
defined 402

J

join, defined 402

K

key bindings for command-line editing 38

L

l annotation for deselected loop regions 280
L annotation for selected loop regions 280
latency
average latency (event latency/counts) 105
cache miss resolution, characterized on SPPs 105
defined 107, 402
latency field in events reports 156
latency/counts, defined 402
latency/CPU, defined 402
learning CXpa quickly 29

libraries, archive
preparing for profiling 23
limitations, of CXpa 13
line mode
changing the PDF name (set pdf command) 361
combining multiple PDFs (merge command) 335
creating reports with analyze command 301
description in man page 315
deselecting regions for profiling 319
displaying
information for a CXpa session 325
online help (help command) 321
source code (list command) 327
invoking CXpa 311
key bindings (command line editing) 38
learning CXpa quickly 35
-nw (no windows) option 311
overview 9
pre-instrumenting executables 56
quitting CXpa 341
selecting events (set events command) 357
selecting metrics to collect (collect command) 307
selecting regions to profile (select command) 353
linking 19
list command 327
list file name, displaying
in GUI mode 225
using the info command 326
list selectable command 331
listing
regions available for profiling in line mode 331
source files in line mode 327
load, defined 402
local memory 107
local_and_remote_misses
parameter of set events command 119, 358
local_misses
parameter of set events command 119, 357
locally and remotely resolved data cache misses 109
locally resolved data cache misses 109
defined 403
on S2000/X2000/SPP1600 109
loop blocking, defined 403
loop distribution, defined 403
loop interchange, defined 403
loop nesting levels 89, 366
and set visibility command 365
fixed 90, 366
default settings 90
relative 91, 366
specifying in line mode 365
loop replication, defined 403
Loop reports
discussed 131
displaying in GUI mode 195
displaying in line mode (analyze command) 303
loop unrolling, defined 404

M

m profiling status 158
memory bank conflict, defined 404
merge command 335
Merge PDFs dialog 227
merging data from multiple PDF files 63
message passing, defined 404
message-passing programs, profiling with CXpa 61
messages, CXpa
displaying online help for 322, 375
introduction 375
metrics
available on all architectures 101
average clock cycles per instruction 103
average MIPS rate 103
chunk counts for parallel loops 102
CPU time 101
data and instruction TLB misses (SPP1200/1600) 110
data cache miss counts and latency (S2000/X2000/SPP1600) 109
data cache miss counts and latency (SPP1200/1600) 110
described 101
discussed 101
dynamic call graph 101
event latency/counts (average latency per event) 105
event latency/CPU 106
events
off-processor (S2000/X2000/SPP1600) 109
on-processor (SPP1200/1600) 107
selecting in GUI mode 113
terminology 103
execution counts 102
instruction cache miss counts and latency (SPP1200/1600) 110
instruction counts and clock cycles (SPP1200/1600) 110
memory access events 102, 107
selecting
in GUI mode 249
in line mode 117
selecting with the collect command 307
sorting in 2D and 3D profile graphs 273
wall clock time 101
MIPS rate, average
defined 103
defined (glossary) 404
field in reports 155
mpa utility
using with CXpa 67
MPI programs
profiling with CXpa 61

N

-nap CXpa start-up option 311
New PDF dialog 231
New PDF menu option 49
NL column in loop reports 156
notational conventions xiii
-nw (no windows) CXpa start-up option 311

O

object files 19
  preparing for profiling 23
Off-Processor Event Counter(s) option menu 252
off-processor events (S2000/X2000/SPP1600) 119
  discussed 107
  defined 405
online help
  printing in ASCII 77
online help, using
  in GUI mode 73
  in line mode 321
On-Processor Event Counter(s) option menu 252
on-processor events (SPP1200/1600) 120
  defined 405
  discussed 107
Open PDF dialog 235
Open PDF field 191
opening PDFs 188
optimization
  abbreviations for loop transformations used in reports 157
  associated compiler documentation xiv
  defined 405
  levels and profiling 17
  levels, defined 405
  parallel, defined 406
Optimized Loops (by thread)
  section in parallel region reports 144, 157
Optimized Loops (cumulative)
  section in parallel region reports 144, 157
options
  compiler
    optimization levels and profiling 17
  CXpa start-up
    -e 311
    -help 311
    listed 311
    -nap 311
    -nw 311
    -nx 311
    -path 312
    -pdf 312
    -pid 312
    -stack bytes 312
    -tm 312
    -w 312
    -win 312
    -x 313
  specifying with CXPA environment variable 316
ordering documentation xvi
organization xi, xii
oversubscription of threads, defined 405
overview
  available metrics 7
  batch mode interface 9
  CXpa 5
  interfaces to CXpa 8
  line mode interface 9
  performance reports 11
  profiles and graphs 10
  X/Motif GUI interface 9

P

p annotation for deselected parallel region 280
P annotation for selected parallel regions 280
-p compiler option 18
P optimization abbreviation 157
p profiling status 158
parallel performance graph 182
Parallel Region performance reports 143
  creating 154
  displaying in GUI mode 195
  displaying in line mode (analyze command) 303
parallel regions (parallel loops)
  as a region type 18
parallelization
  defined 406
  loop, defined 406
path command 339
-path CXpa start-up option 311
Pause button 206
pausing a program in line mode 309
PDF (performance data file)
  active 188
    selecting in GUI mode 52
  analyzing multiple PDFs 51
  changing name to prevent overwriting of existing PDF 49, 361
  controlling from the Analysis Control window 187
  creating with the New PDF dialog 231
  CXpa version created with listed
    in GUI mode 223
    in line mode 325
  default name (<*executable*>.pdf) 49
  defined (glossary) 406
  described 49
  displaying information about
    in GUI mode 223
    in line mode (info command) 325
  executable creation date 223

executable listed 223
format version listed
in GUI mode 223
in line mode 325
Info PDF dialog 223
instrumentation version listed 224
invoking CXpa with a PDF only (analysis-only mode) 187
listing creation date
in GUI mode 223
in line mode 325
listing the current PDF
in GUI mode 223
using the `info` command 325
merging data from multiple PDFs 63
in GUI mode 189, 227
in line mode (`merge` command) 335
opening 188
Open PDF dialog 235
other PDFs (GUI mode) 51
other PDFs (line mode) 54
process state listed (in GUI mode) 223
setting
at CXpa start-up (`-pdf` option) 312
in GUI mode 49
in line mode 50
with the `set pdf` command 361
`-pdf` CXpa start-up option 311
PDF field (Executable Manager window) 206
PDF file
specifying a new PDF name 49
`-pid` CXpa start-up option 312
preface xi
pre-instrumented executables
creating with `save executable` command 349
defined 406
Save Executable As dialog 263
using 55
in GUI mode 58
in line mode 56
printing online help 77
Process ID 221
Process State field 206
process state of PDF 223
process, current state
displaying in GUI mode 221
displaying in line mode 325
process, defined 406
processor agent chip, defined 108
Product Information dialog 239
product number
displaying in GUI mode 239
displaying with the `version` command 373
`PROFDIR` environment variable 62
and pre-instrumented executables 56
Profile Selection button 206
Profile Selection dialog 41, 241
selecting metrics to collect 114, 249
selecting regions to profile 86, 242
profilers
`prof` 5
profiling
by statistical sampling 5
learning CXpa quickly 29
MPI/PVM applications 61
regions 18
strategy 41
profiling a program 204
profiling intrusion
discussed 41
minimizing 42
profiling status (PS) field in reports 157
program
executing with the `run` command 345
reexecuting, with the `rerun` command 343
stopping execution in line mode 371
program arguments
specifying on command line with `-e` option 70
Program Arguments field 206
PS (profiling status) field in performance reports 157
`pU:`*n* optimization abbreviation 157
PVM programs
analyzing profiling data from 63
profiling with CXpa 61

Q

`quit` command 341
quitting CXpa
from the X/Motif GUI 35
with the `quit` command 341

R

`r` annotation for deselected routine regions 280
`R` annotation for selected routine region 280
read misses, defined 107, 407
read, defined 407
`read_only`
parameter of `set events` command 119, 358
redirection
of program I/O 345
reports 304
redirection operators
using with `analyze` command 301
using with `rerun` command 343
using with `run` command 345
Region Subset Selection dialog 255
regions, source code
annotations 83
described 81
deselecting with `deselect` command 319
selecting for profiling 81

in GUI mode 85, 86, 242
in line mode 93
using the select command 353
types of 18
Related Commands, defined 297
relative loop nesting levels 91, 366
release notice, CXpa
displaying in GUI mode 13
displaying in line mode 321
remote_misses
parameter of set events command 119, 357
remotely resolved data cache misses 109
defined 407
on S2000/X2000/SPP1600 systems 109
removing a directory from CXpa's search path 284
Report button 191
reporting problems xvi
reports
Analysis Report window 193
creating or viewing 125
customizing 255
Data Cache Accesses (SPP1200/1600 only) 167
displaying
in GUI mode 195
in line mode with analyze command 302
Dynamic Call Graph 127
fields in, listed and described 155
header information 126
Instructions Completed and Clock Cycles (SPP1200/1600 only) 167
Loop 131
Computation 133
creating 141
Events 134
Optimized Loops section 132
Time to Solution 133
Optimized Loops (by threads) section 144, 157
Optimized Loops (cumulative) section 144, 157
Optimized Loops section 132
overview 123
Parallel Region 143
displaying in GUI mode 195
Events 146
Time to Solution 144
redirecting to a file (in line mode) 304
Routine 161
cache misses and latency 165
Call Counts 162
Computation 162
Events 164
Time to Solution 163
saving to a file (in GUI mode) 271
thread/process visibility setting 124
rerun command 343
Reset Zoom button
2D Profile window 178
3D Profile window 184
resuming
data collection 309
execution 309
RISC (reduced instruction set computer), defined 407
Routine Detail dialog 259
creating 261
Routine reports
Call Counts 162
Computation 162
described 161
displaying in GUI mode 195
displaying in line mode (analyze command) 302
Events 164
Time to Solution 163
routines
as a region type 18
calling uninstrumented routines 13, 21
run command 345
running a program under CXpa
in line mode 345
using the GUI 29

S

S2000 (Exemplar)
off-processor event metrics 109
Save Executable As dialog 263
save executable command 349
Save Profile dialog 267
Save Report dialog 271
Scalable Parallel Processor (SPP), defined 408
SCI (Scalable Coherent Interface), defined 407
script, batch mode execution example 71
Search button (Help window) 74, 219
Search Field (Help window) 74
search path
adding to with the add path command 299
changing (in GUI mode) 283
current, listed 225
-path CXpa start-up option 312
setting with the path command 339
searches
online help searches (titles or all text) 219
select command 93, 353
select pregion example 355
Select Regions options menu 177, 195
selecting
events to collect
in GUI mode 115
in line mode 357
metrics to collect
in GUI mode 113, 242
in line mode (collect command) 307
regions to profile
in GUI mode 242

in line mode 93
session information, displaying 225
set events command 357
parameters (S2000/X2000/SPP1600) 119
parameters (SPP1200 and SPP1600) 120
set pdf command 361
set subcomplex command 363
set visibility command 365
using to filter report data 124
setting the PDF 49
Show All button
2D Profile window 178
3D Profile window 184
Sort dialog 273
source code
annotations 83, 280
correlation in 2D profile graph 176
correlation in 3D profile graph 182
correlation in Call Graph window 199
displaying (in line mode) 327
displaying associated code from profile windows 33
displaying in GUI mode 83
list command 327
list selectable command 331
search path, modifying (GUI mode) 283
selecting files to display (GUI mode) 277, 280
Source Code window 279
source code regions
annotations for 83
described 81
deselecting in line mode 93
deselecting with the deselect command 319
selecting for profiling 81
in GUI mode 85
in line mode 93
Profile Selection dialog 242
Source Code Selection dialog 277
Source Code window 279
source command 369
source files
listing in line mode 327
replacing search path with the path command 339
Source Search Path dialog 283
spawn, defined 408
SPP (Scalable Parallel Processor), defined 408
SPP1200
data and instruction cache misses and latency reports 165
Data Cache Accesses report 167
Instructions Completed and Clock Cycles report 167
on-processor event metrics 110
SPP1600
data and instruction cache misses and latency reports 165
Data Cache Accesses report 167
Instructions Completed and Clock Cycles report 167
local and remote cache misses and latency reports 165
off-processor event metrics 109
on-processor event metrics 110
-stack bytes CXpa start-up option 311
stack size needed for CXpa 312
Start button 206
starting CXpa 311
in line mode (tty interface) 35
using the GUI interface 30
with a PDF only (analysis mode) 51
stop command 371
stopping a program
using the stop command 371
store, defined 408
strip mining (loop transformation), defined 403
subcomplex
defined 408
listing available, in line mode 326
selecting in GUI mode 285
selecting in line mode 363
system, defined 408
SubComplex field 206
SubComplex Selection button 206
Subcomplex Selection dialog 285
symmetric multiprocessor (SMP), defined 408
syntax
conventions xiii
defined 297

T

t profiling status 158
Technical Assistance Center (TAC) xvi
thread
defined 408
thread ID, defined 409
thread ID, kernel 409
thread ID, spawn 409
Thread Selection dialog 287
thread visibility
setting in GUI mode 211
setting in line mode 367
Time to Solution report
for parallel loops 144
for routines 163
for serial loops 133
Titles/All Text button (Help window) 74
Titles/All Text searches in Help window 74, 219
TLB (translation lookaside buffer)
defined 409
TLB data and instruction misses
defined 409
tlb_misses
parameter of set events command 120, 359
-tm CXpa start-up option 312
and pre-instrumented executables 55

translation lookaside buffer (TLB)
defined 409

U

u profiling status 158
using CXpa
in line mode (tty interface) 35
X/Motif GUI interface 29
using this book xi

V

`version` command 373
version number of CXpa
displaying in GUI mode 239
displaying with the `version` command 373
visibility
process/thread
setting in GUI mode 211
setting in line mode 367

W

wall clock time 101
defined 409
`WARNING` message type 375
`-win` CXpa start-up option 312
windows
2D Profile 176
3D Profile 182
Analysis Control 187
Analysis Report 193
Call Graph 197
Executable Manager 203
Filter Profile dialog 209
Filter Report dialog 211
Find Routine dialog 213
Help 217
Info Executable dialog 221
Info PDF dialog 223
Info Session dialog 225
introduction 173
Merge PDFs dialog 227
New PDF dialog 231
Open PDF dialog 235
Product Information dialog 239
Profile Selection dialog 242
Region Subset Selection dialog 255
Routine Detail dialog 259
Save Executable As dialog 263
Save Profile dialog 267
Save Report dialog 271
Sort dialog 273
Source Code 279
Source Code Selection dialog 277
Source Search Path dialog 283
Subcomplex Selection dialog 285
Thread Selection dialog 287
X defaults 289
working directory, listed 225
write misses, defined 108, 410
write, defined 409
`write_only`
parameter of `set events` command 119, 358

X

`-x` CXpa start-up option 69, 311
X defaults 289
x profiling status 158
X resource settings 289
X resource, for automatic window creation 188
X Toolkit options 313
X/Motif GUI 9
X2000
off-processor event metrics 109

Y

y profiling status 158

Z

z profiling status 158
Zoom dialog 293
2D graph zoom (scaling) options 294
3D graph zoom (scaling) options 294
Zoom-In button
2D Profile window 178
3D Profile window 184
Zoom-Out button
2D Profile window 178
3D Profile window 184

B5639-90002