

# IBM System Blue Gene Solution: High Performance Computing Toolkit for Blue Gene/P



Tools to visualize and analyze your  
performance data

Instructions for the Xprofiler and High  
Performance Computing Toolkit GUIs

Tips to optimize your  
application's performance

Gary Lakner  
I-Hsin Chung  
Dr. Guojing Cong  
David Klepacki  
Dr. Christoph Pospiech  
Seetharami R. Seelam  
Hui-Fang Wen





International Technical Support Organization

**IBM System Blue Gene Solution: High Performance  
Computing Toolkit for Blue Gene/P**

December 2007

**Note:** Before using this information and the product it supports, read the information in “Notices” on page v.

**First Edition (December 2007)**

This edition applies to Version 1, Release 1, Modification 1 of IBM System Blue Gene/P Solution (product number 5733-BGP).

**© Copyright International Business Machines Corporation 2007. All rights reserved.**

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

# Contents

<b>Notices</b> .....	v
Trademarks .....	vi
 <b>Preface</b> .....	vii
The team that wrote this paper .....	vii
Become a published author .....	viii
Comments welcome .....	ix
 <b>Chapter 1. MPI Profiler and Tracer</b> .....	1
1.1 System and software requirements .....	2
1.2 Compiling and linking .....	2
1.3 Environment variables .....	3
1.3.1 TRACE_ALL_EVENTS .....	3
1.3.2 TRACE_ALL_TASKS .....	4
1.3.3 TRACE_MAX_RANK .....	4
1.3.4 TRACEBACK_LEVEL .....	4
1.3.5 SWAP_BYTES .....	4
1.3.6 TRACE_SEND_PATTERN (Blue Gene/L and Blue Gene/P only) .....	5
1.4 Output .....	5
1.4.1 Plain text file .....	6
1.4.2 The VIZ file .....	9
1.4.3 Trace file .....	10
1.5 Configuration .....	10
1.5.1 Configuration functions .....	10
1.5.2 Data structure .....	11
1.5.3 Utility functions .....	13
1.6 Related issues .....	16
1.6.1 Overhead .....	16
1.6.2 Multithreading .....	16
 <b>Chapter 2. CPU profiling using Xprofiler</b> .....	17
2.1 Starting Xprofiler .....	18
2.2 Understanding the Xprofiler display .....	20
2.2.1 Xprofiler main menus .....	21
2.2.2 Elements of the function call tree .....	22
2.2.3 Manipulating the Xprofiler display .....	24
2.3 Getting performance data for your application .....	31
 <b>Chapter 3. Hardware Performance Monitoring</b> .....	39
3.1 HPM .....	40
3.2 Events and groups .....	41
3.3 Derived metrics .....	83
3.4 Inheritance .....	83
3.5 Inclusive and exclusive values .....	84
3.5.1 Parent-child relations .....	84
3.5.2 Handling overlap issues .....	85
3.5.3 Computation of exclusive values for derived metrics .....	85
3.6 Function reference .....	85
3.7 Measurement overhead .....	87

3.8	Output	87
3.9	Examples of libhpm for C and C++	88
3.10	Multithreaded program instrumentation issues	89
3.11	Considerations for MPI parallel programs	89
3.11.1	Distributors	90
3.11.2	Aggregators	90
3.11.3	Plug-ins shipped with HPCT	90
3.11.4	User-defined plug-ins	91
3.11.5	Detailed interface description	91
3.11.6	Getting the plug-ins to work	93
<b>Chapter 4. High Performance Computing Toolkit GUI</b>		95
4.1	Starting the HPCT GUI	96
4.2	HPCT GUI Main window (Visualization)	96
4.3	HPCT GUI Main Window with instrumentation	104
4.4	HPCT GUI simple IDE	109
<b>Chapter 5. I/O performance</b>		111
5.1	Design summary	112
5.2	Runtime control of MIO	112
5.2.1	MIO_STATS	112
5.2.2	MIO_FILES	112
5.2.3	MIO_DEFAULTS	114
5.2.4	MIO_DEBUG	114
5.3	Module descriptions and options	114
5.4	Library implementation	116
5.5	Sample implementation	117
<b>Related publications</b>		125
	IBM Redbooks	125
	How to get IBM Redbooks	125
	Help from IBM	125

# Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

*IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.*

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:** INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

## COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

## Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

AIX 5L™

AIX®

Blue Gene/L™

Blue Gene/P™

Blue Gene®

IBM®

PowerPC®

POWER™


POWER4™

POWER5™

POWER5+™

POWER6™

Redbooks®

Redbooks (logo) ®

System i™

System p™

Tracer™

The following terms are trademarks of other companies:

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.



# Preface

This IBM® Redpaper publication is one in a series of IBM documents written specifically for the IBM System Blue Gene/P Solution. The Blue Gene/P system is the second generation of a massively parallel supercomputer from IBM in the IBM System Blue Gene Solution series. This paper provides an overview of the IBM High Performance Computing Toolkit for the Blue Gene/P™ system.

We begin by describing the Message Passing Interface (MPI) Profiler and Tracer™ tool, which collects profiling and tracing data for MPI programs. We explain the system requirements as well as configuration, compiling, linking, environment variables, and output.

Next we discuss how to use Xprofiler for CPU profiling. We then move on to discuss Hardware Performance Monitoring (HPM), including the use and behavior of the libhpm library. Afterward, we describe the GUI of the High Performance Computing Toolkit (HPCT). This single interface provides a means to execute the application and visualize and analyze the collected performance data.

Finally we address I/O performance. Specifically, we discuss the features of the Modular I/O (MIO) library that was developed to assist in optimizing an application's I/O.

## The team that wrote this paper

This paper was produced by a team of specialists from around the world working at the International Technical Support Organization (ITSO), Rochester Center.

**Gary Lakner** is a Staff Software Engineer for IBM Rochester on assignment in the ITSO. He is a member of the Blue Gene/L™ Support Team in the IBM Rochester Support Center, where he specializes in both Blue Gene® hardware and software, as well as performs customer installations. Prior to joining the Blue Gene team, Gary supported TCP/IP communications on the IBM System i™ platform. Gary has been with IBM since 1998.

**I-Hsin Chung** is a Research Staff Member at the IBM Thomas J. Watson Research Center. His research interests include performance tuning, performance analysis, and performance tools. His experience includes designing and developing performance tools on IBM platforms such as IBM Power Systems on AIX® and Linux®, and the Blue Gene/L and Blue Gene/P systems. Prior to joining IBM Research, he received his Ph.D. in Computer Science from the University of Maryland, College Park in 2004.

**Dr. Guojing Cong** is a research staff member at the IBM Thomas J. Watson Research Center. His research interests include automation of performance analysis and tuning for scientific applications, as well as the design and analysis of parallel algorithms for large-scale graph problems. He is an expert in solving irregular combinatorial problems on parallel systems. He received his Ph.D. in computer engineering from the University of New Mexico in 2004 and soon after joined IBM Research.

**David Klepacki** is a senior staff member of IBM Research and has more than 20 years of experience in high performance computing (HPC). He currently manages the Advanced Computing Technology department at the IBM Thomas J. Watson Research Center. He earned a Ph.D. in theoretical nuclear physics from Purdue University and has since worked in a variety of technical areas within IBM including high-performance processor design,

numerically intensive computation, computational physics, parallel computing, and performance modeling.

**Dr. Christoph Pospiech** joined IBM in 1988 as a member of the IBM Scientific Center Heidelberg, after completing his Ph.D. in applied mathematics at Heidelberg University. He has nearly 20 years of experience in the area of vector and parallel computing. Currently he is part of the Systems Technology Group, working on customer applications, mainly from the weather and climate area. He collaborated with the Advanced Computing Technology department at the IBM Thomas J. Watson Research Center on developing and maintaining tool components.

**Seetharami R. Seelam** is a post-doctoral research staff member at the IBM Thomas J. Watson Research Center in Yorktown Heights, NY, since 2007. He is a member of the IBM Advanced Computing Technologies Center, where he works on performance analysis tools and technologies for AIX on the IBM System p™ platform and Blue Gene systems as well as on next-generation automatic performance analysis, bottleneck detection, and solution determination technologies. He received a Ph.D. in Computer Science from University of Texas at El Paso in 2006. His areas of interest are in HPC, operating systems, I/O, performance tools, and resource management.

**Hui-Fang Wen** is an Advisory Software Engineer at the IBM Thomas J. Watson Research Center. She is a member of the IBM Advanced Computing Technology Center, where she works on performance tools and the GUI design. She holds a Masters in Computer Science degree from University of Maryland, College Park. She has been with IBM since 2005.

Tom Gray  
Todd Kelsey  
Gary Mullen-Schultz  
Craig Schmitz  
Jenifer Servais  
ITSO, Rochester Center

## Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbooks® publication dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You will have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you will develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

[ibm.com/redbooks/residencies.html](http://ibm.com/redbooks/residencies.html)

## Comments welcome

Your comments are important to us!

We want our papers to be as helpful as possible. Send us your comments about this paper or other Redbooks publications in one of the following ways:

- ▶ Use the online **Contact us** review redbook form found at:

[ibm.com/redbooks](http://ibm.com/redbooks)

- ▶ Send your comments in an e-mail to:

[redbooks@us.ibm.com](mailto:redbooks@us.ibm.com)

- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization  
Dept. HYTD Mail Station P099  
2455 South Road  
Poughkeepsie, NY 12601-5400





# MPI Profiler and Tracer

In this chapter, we provide documentation for the Message Passing Interface (MPI) profiling and tracing library of the IBM High Performance Computing Toolkit. The MPI profiling and tracing library collects profiling and tracing data for MPI programs. Table 1-1 provides the library file names and their usage.

*Table 1-1 Library file names and usage*

Library name	Usage
libmpitrace.a	Library for both the C and Fortran applications
mpt.h	Header files

**Note:** The C header file is used when it is necessary to configure the library.

## 1.1 System and software requirements

The following systems and software are required for the currently supported architecture:

- ▶ AIX on Power: IBM Parallel Environment (PE) for AIX program product and its Parallel Operating Environment (POE), 32 bit and 64 bit
- ▶ Linux on Power: IBM PE for Linux program product and its POE, 32 bit and 64 bit
- ▶ Blue Gene/L: System software V1R3M0 or later
- ▶ Blue Gene/P: System software V1R1M1 or later

## 1.2 Compiling and linking

The trace library uses the debugging information that is stored within the binary to map the performance information back to the source code. To use the library, the application must be compiled with the `-g` option.

You might consider turning off or having a lower level of optimization (`-O2`, `-O1`,...) for the application when linking with the MPI profiling and tracing library. High level optimization affects the correctness of the debugging information and can also affect the call stack behavior.

To link the application with the library, add the following options to your command line:

- ▶ The option `-L/path/to/libraries`, where `/path/to/libraries` is the path where the libraries are located
- ▶ The option `-lmpitrace`, which should be before the MPI library `-lmpich`, in the linking order
- ▶ The option `-lllicense` to link the license library

For some platforms, if the shared library `liblicense.so` is used, you might need to set the environment variable `LD_LIBRARY_PATH` to `$IHPCT_BASE/lib(lib64)` to make sure that the application finds the correct library during runtime.

Example 1-1 shows how to compile and link in C, using `mpicc`, which currently is based on the GNU compiler.

*Example 1-1 Compiling and linking in C*

---

```
BGPHOME=/bgsys/drivers/ppcfloor
CC=$(BGPHOME)/comm/bin/mpicc
CFLAGS = -I$(BGPHOME)/comm/include -g -O
TRACE_LIB = -L</path/to/libmpitrace.a> -lmpitrace -lllicense
LIB1 = -L$(BGPHOME)/comm/lib -lmpich.cnk -ldcmfcoll.cnk -ldcmf.cnk
LIB2 = -L$(BGPHOME)/runtime/SPI -lSPI.cna -lpthread -lrt
LIB3 = -lgfortranbegin -lgfortran # please read the NOTE
mpitrace: mpi_test.c
    $(CC) -o $@ $< $(CFLAGS) $(TRACE_LIB) $(LIB1) $(LIB2) -lm
```

---

In order to accommodate part of the MPI profiling and tracing library that is written in Fortran, it is necessary to link the two GNU Fortran libraries. Example 1-2 shows how to compile and link a program in Fortran.

*Example 1-2 Compiling and linking in Fortran*

---

```
BGPHOME=/bgsys/drivers/ppcfloor
CC=$(BGPHOME)/comm/bin/mpif77
FFLAGS = -I$(BGPHOME)/comm/include -g -O
TRACE_LIB = -L</path/to/libmpitrace.a> -lmpitrace -llicense
LIB1 = -L$(BGPHOME)/comm/lib -lmpich.cnk -ldcmfcoll.cnk -ldcmf.cnk
LIB2 = -L$(BGPHOME)/runtime/SPI -lSPI.cna -lpthread -lrt
statusesf: statusesf.f
    $(CC) -o $@ $< $(FFLAGS) $(TRACE_LIB) $(LIB1) $(LIB2)
```

---

## 1.3 Environment variables

In this section, we list and describe the environment variables that are used by the toolkit.

### 1.3.1 TRACE\_ALL\_EVENTS

The wrappers can be used in two modes. The default value is set to *yes* and collects both a timing summary and a time history of MPI calls that are suitable for graphical display. If this environment variable is set to *yes*, it saves a record of all MPI events one after MPI Init(), until the application completes or until the trace buffer is full. By default, for MPI ranks 0-255, or for all MPI ranks, if there are 256 or fewer processes in MPI\_COMM\_WORLD, you can change this setting by using TRACE\_ALL\_TASKS or the configuration that is described in 1.5, “Configuration” on page 10.

Another method is to control the time-history measurement within the application by calling routines to start or stop tracing. The following examples show these routines for Fortran, C, and C++:

► Fortran syntax

```
call trace_start()
do work + mpi ...
call trace_stop()
```

► C syntax

```
void trace_start(void);
void trace_start(void);
trace_start();
do work + mpi ...
trace_stop();
```

► C++ syntax

```
extern "C" void trace_start(void);
extern "C" void trace_start(void);
trace_start();
do work + mpi ...
trace_stop();
```

To use one of the previous control methods, the `TRACE_ALL_EVENTS` variable must be disabled. Otherwise, it traces all events. You can use one of the following commands, depending on your shell, to disable the variable:

- ▶ `bash`  
`export TRACE_ALL_EVENTS=no`
- ▶ `csh`  
`setenv TRACE_ALL_EVENTS no (csh)`

### 1.3.2 TRACE\_ALL\_TASKS

When saving MPI event records, it is easy to generate trace files that are too large to visualize. To reduce the data volume, when you set `TRACE_ALL_EVENTS=yes`, the default behavior is to save event records from MPI tasks 0-255 or for all MPI processes if there are 256 or fewer processes in MPI COMM WORLD. That should be enough to provide a clear visual record of the communication pattern.

If you want to save data from all tasks, you must set this environment variable to `yes` by using one of the following commands depending on your shell:

- ▶ `bash`  
`export TRACE_ALL_TASKS=yes`
- ▶ `csh`  
`setenv TRACE_ALL_TASKS yes`

### 1.3.3 TRACE\_MAX\_RANK

To provide more control, you can set `MAX_TRACE_RANK=#`. For example, if you set `MAX_TRACE_RANK=2048`, you get trace data from 2048 tasks, 0-2047, provided that you have at least 2048 tasks in your job. By using the time-stamped trace feature selectively, both in time (trace start/trace stop) and by MPI rank, you can gain insight into the MPI performance of large complex parallel applications.

### 1.3.4 TRACEBACK\_LEVEL

In some cases, there might be deeply nested layers on top of MPI and you might need to profile higher up the call chain (functions in the call stack). You can do this by setting this environment variable (default value is 0). For example, setting `TRACEBACK_LEVEL=1` indicates that the library must save addresses starting with the parent in the call chain (level = 1), not with the location of the MPI call (level = 0).

### 1.3.5 SWAP\_BYTES

The event trace file is binary, and therefore, it is sensitive to byte order. For example, Blue Gene/L is big endian, and your visualization workstation is probably little endian (for example, x86). The trace files are written in little endian format by default.

If you use a big endian system for graphical display, such as Apple OS/X, AIX on the System p workstation, and so on, you can set an environment variable by using one of the following commands depending on your shell:



- ▶ bash
 

```
export SWAP_BYTES=no
```
- ▶ csh
 

```
setenv SWAP_BYTES no
```

Setting this variable results in a trace file in big endian format when you run your job.

### 1.3.6 TRACE\_SEND\_PATTERN (Blue Gene/L and Blue Gene/P only)

In either profiling or tracing mode, there is an option to collect information about the number of hops for point-to-point communication on the torus network. This feature can be enabled by setting the TRACE\_SEND\_PATTERN environment variable as follows depending on your shell:

- ▶ bash
 

```
export TRACE_SEND_PATTERN=yes
```
- ▶ csh
 

```
setenv TRACE_SEND_PATTERN yes
```

When you set this variable, the wrappers keep track of the number of bytes that are sent to each task, and a binary file *send bytes.matrix* is written during MPI Finalize, which lists the number of bytes that were sent from each task to all other tasks. The binary file has the following format:

$D_{00}, D_{01}, \dots, D_{0n}, D_{10}, \dots, D_{ij}, \dots, D_{nn}$

In this format, the data type  $D_{ij}$  is double (in C), and it represents the size of MPI data that is sent from rank  $i$  to rank  $j$ . This matrix can be used as input to external utilities that can generate efficient mappings of MPI tasks onto torus coordinates. The wrappers also provide the average number of hops for all flavors of MPI Send. The wrappers do not track the message-traffic patterns in collective calls, such as MPI Alltoall. Only point-to-point send operations are tracked. AverageHops for all communications on a given processor is measured as follows:

$$\text{AverageHops} = \text{sum}(\text{Hops}_i \times \text{Bytes}_i) / \text{sum}(\text{Bytes}_i)$$

$\text{Hops}_i$  is the distance between the processors for MPI communication, and  $\text{Bytes}_i$  is the size of the data that is transferred in this communication. The logical concept behind this performance metric is to measure how far each byte has to travel for the communication (in average). If the communication processor pair is close to each other in the coordinate, the AverageHops value tends to be small.

## 1.4 Output

After building the binary executable and setting the environment, run the application as you normally would do. To have better control for the performance data collected and output, refer to 1.5, “Configuration” on page 10.

## 1.4.1 Plain text file

The wrapper for MPI Finalize() writes the timing summaries in *mpi profile.taskid* files. The *mpi profile.0* file contains a timing summary from each task. Currently, for scalability reasons, only four ranks, rank 0 and rank with (min,med,max) MPI communication time, generate a plain text file by default. To change this default setting, one simple function can be implemented and linked into compilation. Example 1-3 provides the function.

*Example 1-3 Function to change the default*

---

```
control.c:
int MT_output_trace(int rank) {
    return 1;
}
mpitrace: mpi_test.c
$(CC) $(CFLAGS) control.o mpi_test.o $(TRACE_LIB) -lm -o $@
```

---

Example 1-4 shows an example of *mpi profile.0*.

*Example 1-4 mpi profile.0*

---

```
elapsed time from clock-cycles using freq = 700.0 MHz
-----
MPI Routine #calls avg. bytes time(sec)
-----
MPI_Comm_size 1 0.0 0.000
MPI_Comm_rank 1 0.0 0.000
MPI_Isend 21 99864.3 0.000
MPI_Irecv 21 99864.3 0.000
MPI_Waitall 21 0.0 0.014
MPI_Barrier 47 0.0 0.000
-----
total communication time = 0.015 seconds.
total elapsed time = 4.039 seconds.
-----
Message size distributions:
MPI_Isend #calls avg. bytes time(sec)
          3      2.3      0.000
          1      8.0      0.000
          1     16.0      0.000
          1     32.0      0.000
          1     64.0      0.000
          1    128.0      0.000
          1    256.0      0.000
          1    512.0      0.000
          1   1024.0      0.000
          1   2048.0      0.000
          1   4096.0      0.000
          1   8192.0      0.000
          1  16384.0      0.000
          1  32768.0      0.000
          1  65536.0      0.000
          1 131072.0      0.000
          1 262144.0      0.000
          1 524288.0      0.000
          11048576.0      0.000
```

MPI_Irecv	#calls	avg. bytes	time(sec)
3	2.3	0.000	
1	8.0	0.000	
1	16.0	0.000	
1	32.0	0.000	
1	64.0	0.000	
1	128.0	0.000	
1	256.0	0.000	
1	512.0	0.000	
1	1024.0	0.000	
1	2048.0	0.000	
1	4096.0	0.000	
1	8192.0	0.000	
1	16384.0	0.000	
1	32768.0	0.000	
1	65536.0	0.000	
1	131072.0	0.000	
1	262144.0	0.000	
1	524288.0	0.000	
1	1048576.0	0.000	

-----

Communication summary for all tasks:

minimum communication time = 0.015 sec for task 0

median communication time = 4.039 sec for task 20

maximum communication time = 4.039 sec for task 30

taskid xcoord ycoord zcoord procid total\_comm(sec) avg\_hops

0	0	0	0	0	0.015	1.00
1	1	0	0	0	4.039	1.00
2	2	0	0	0	4.039	1.00
3	3	0	0	0	4.039	4.00
4	0	1	0	0	4.039	1.00
5	1	1	0	0	4.039	1.00
6	2	1	0	0	4.039	1.00
7	3	1	0	0	4.039	4.00
8	0	2	0	0	4.039	1.00
9	1	2	0	0	4.039	1.00
10	2	2	0	0	4.039	1.00
11	3	2	0	0	4.039	4.00
12	0	3	0	0	4.039	1.00
13	1	3	0	0	4.039	1.00
14	2	3	0	0	4.039	1.00
15	3	3	0	0	4.039	7.00
16	0	0	1	0	4.039	1.00
17	1	0	1	0	4.039	1.00
18	2	0	1	0	4.039	1.00
19	3	0	1	0	4.039	4.00
20	0	1	1	0	4.039	1.00
21	1	1	1	0	4.039	1.00
22	2	1	1	0	4.039	1.00
23	3	1	1	0	4.039	4.00
24	0	2	1	0	4.039	1.00
25	1	2	1	0	4.039	1.00
26	2	2	1	0	4.039	1.00
27	3	2	1	0	4.039	4.00

28	0	3	1	0	4.039	1.00
29	1	3	1	0	4.039	1.00
30	2	3	1	0	4.039	1.00
31	3	3	1	0	4.039	7.00
MPI tasks sorted by communication time:						
taskid	xcoord	ycoord	zcoord	procid	total_comm(sec)	avg_hops
0	0	0	0	0	0.015	1.00
9	1	2	0	0	4.039	1.00
26	2	2	1	0	4.039	1.00
10	2	2	0	0	4.039	1.00
2	2	0	0	0	4.039	1.00
1	1	0	0	0	4.039	1.00
17	1	0	1	0	4.039	1.00
5	1	1	0	0	4.039	1.00
23	3	1	1	0	4.039	4.00
4	0	1	0	0	4.039	1.00
29	1	3	1	0	4.039	1.00
21	1	1	1	0	4.039	1.00
15	3	3	0	0	4.039	7.00
19	3	0	1	0	4.039	4.00
31	3	3	1	0	4.039	7.00
20	0	1	1	0	4.039	1.00
6	2	1	0	0	4.039	1.00
7	3	1	0	0	4.039	4.00
8	0	2	0	0	4.039	1.00
3	3	0	0	0	4.039	4.00
16	0	0	1	0	4.039	1.00
11	3	2	0	0	4.039	4.00
13	1	3	0	0	4.039	1.00
14	2	3	0	0	4.039	1.00
24	0	2	1	0	4.039	1.00
27	3	2	1	0	4.039	4.00
22	2	1	1	0	4.039	1.00
25	1	2	1	0	4.039	1.00
28	0	3	1	0	4.039	1.00
12	0	3	0	0	4.039	1.00
18	2	0	1	0	4.039	1.00
30	2	3	1	0	4.039	1.00

---

### 1.4.2 The VIZ file

In addition to the mpi profile.taskid files, the library might also generate mpi profile taskid.viz XML format files that you can view by using the High Performance Computing Toolkit (HPCT) GUI as shown in Figure 1-1.

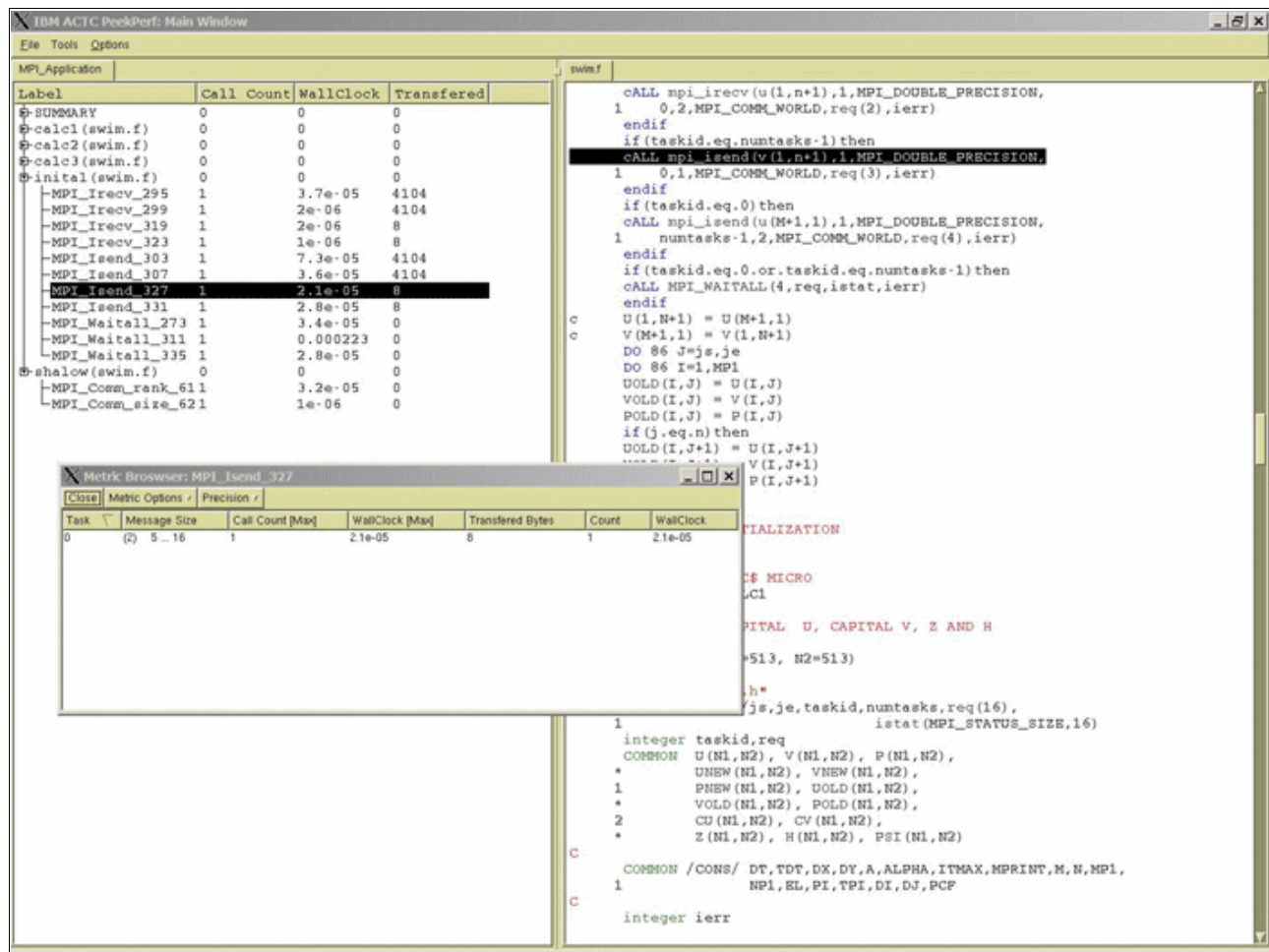


Figure 1-1 HPCT GUI

### 1.4.3 Trace file

The library also generates a file called *single trace*. The Peekview utility can (inside the HPCT GUI or independently) display this trace file as shown in Figure 1-2.

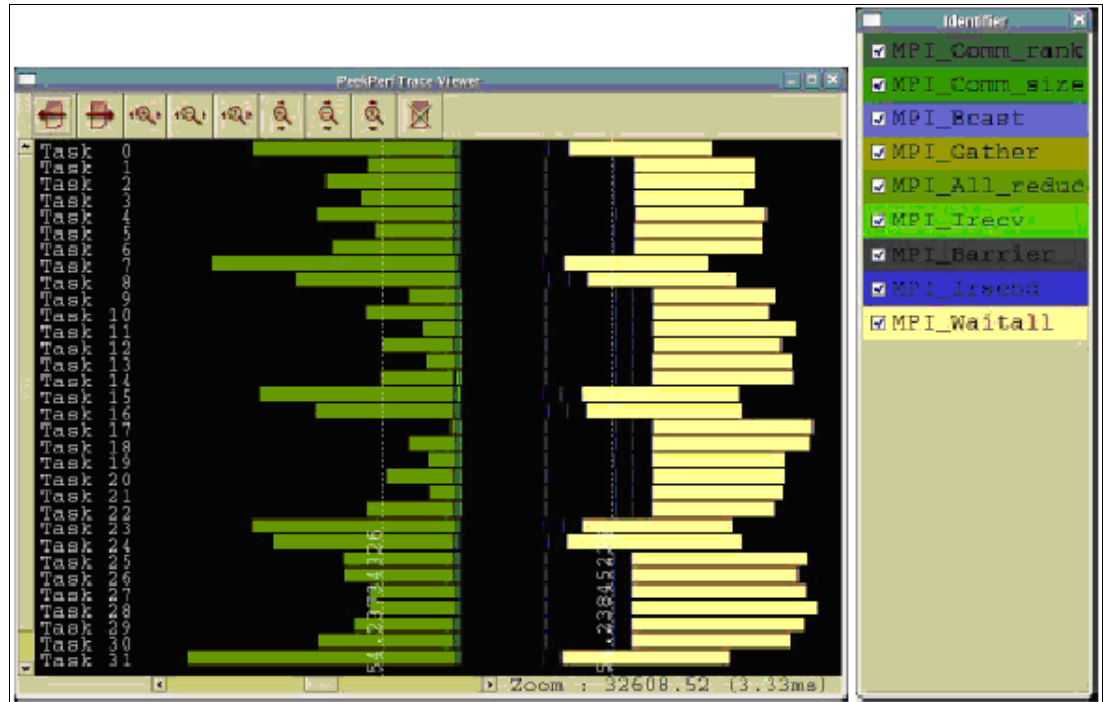


Figure 1-2 Single trace file in the Peekview utility

## 1.5 Configuration

In this section, we describe a more general way to make the tracing tool configurable, and thereafter, to allow users to focus on performance points of interest. By providing a flexible mechanism to control the events that are recorded, the library can remain useful for large-scale parallel applications.

### 1.5.1 Configuration functions

Three functions can be rewritten to configure the library. During run time, the return values of those three functions determine which performance information to store, which process (MPI rank) will output the performance information, and which performance information to output to files.

► `int MT_trace_event(int);`

Whenever an MPI function (profiled or traced) is called, this function is invoked. The integer passed into this function is the ID number for the MPI function. The return value is 1 if the performance information should be stored in the buffer. Otherwise, the return value is 0.

► `int MT_output_trace(int);`

This function is called once in `MPI_Finalize()`. The integer passed into this function is the MPI rank. The return value is 1 if it will output performance information. Otherwise, the return value is 0.

► `int MT_output_text(void);`

This function is called inside the `MPI_Finalize()` function once. The user can rewrite this function to customize the performance data output, for example as user-defined performance metrics or data layout.

## 1.5.2 Data structure

Each data structure that is described in this section is usually used with an associated utility function to provide user information when implementing configuration functions.

### **MT\_summarystruct**

The `MT_summarystruct` data structure, shown in Example 1-5, holds statistics results, which include MPI ranks and statistical values, such as Min, Max, Median, Average, and Sum. The data structure is used together with the `MT_get_allresults()` utility function.

*Example 1-5 MT\_summarystruct data structure*

---

```
struct MT_summarystruct {
    int min_rank;
    int max_rank;
    int med_rank;
    void *min_result;
    void *max_result;
    void *med_result;
    void *avg_result;
    void *sum_result;
    void *all_result;
    void *sorted_all_result;
    int *sorted_rank;
};
```

---

### **MT\_envstruct**

The `MT_envstruct` data structure (Example 1-6) is used with the `MT_get_environment()` utility function. It holds MPI process self information, including MPI rank (`mpirank`), total number of MPI tasks (`ntasks`), and total number of MPI function types (profiled, traced, or `nmpi`). For the Blue Gene/L system, it also provides the process self environment information including x, y, and z coordinates in the torus, dimension of the torus (`xSize`, `ySize`, `zSize`), the processor ID (`procid`), and the CPU clock frequency (`clockHz`).

*Example 1-6 MT\_envstruct data structure*

---

```
struct MT_envstruct {
    int mpirank;
    int xCoord;
    int yCoord;
    int zCoord;
    int xSize;
    int ySize;
    int zSize;
    int procid;
    int ntasks;
    double clockHz;
    int nmpi;
};
```

---

## MT\_tracebufferstruct

The MT\_tracebufferstruct data structure (Example 1-7) is used together with the MT get\_tracebufferinfo() utility function. It holds information about the number of events that are recorded (number events) and information about memory space (in total, used, or available in MB) for tracing.

*Example 1-7 MT\_tracebufferstruct data structure*

---

```
struct MT_tracebufferstruct {
    int number_events;
    double total_buffer; /* in terms of MBytes */
    double used_buffer;
    double free_buffer;
};
```

---

## MT\_callerstruct

The MT\_callerstruct data structure, shown in Example 1-8, holds the caller's information for the MPI function. It is used with the MT get\_callerinfo() utility function. The information includes the source file path, source file name, function name, and line number in the source file.

*Example 1-8 MT\_callerstruct data structure*

---

```
struct MT_callerstruct {
    char *filepath;
    char *filename;
    char *funcname;
    int lineno;
};
```

---

## MT\_memorystruct (Blue Gene/L only)

Since the memory space per compute node on the Blue Gene/L system is limited, the MT\_memorystruct data structure (Example 1-9) is used with the MT get\_memoryinfo() utility function to provide memory usage information.

*Example 1-9 MT\_memorystruct data structure*

---

```
struct MT_memorystruct {
    unsigned int max_stack_address;
    unsigned int min_stack_address;
    unsigned int max_heap_address;
};
```

---



### 1.5.3 Utility functions

The utility functions provide information for program execution to help the user easily customize the MPI Profiler and Tracer. In this section, we describe the interface for the utility functions.

#### **long long MT\_get\_mpi\_counts(int)**

The integer passed in is the MPI ID, and the number of call counts for this MPI function is returned. The MPI ID can be one of the following IDs:

▶ ALLGATHER_ID	▶ IPROBE_ID	▶ SEND_INIT_ID
▶ ALLGATHERV_ID	▶ IRECV_ID	▶ SENDRECV_ID
▶ ALLREDUCE_ID	▶ IRSEND_ID	▶ SENDRECV_REPLACE_ID
▶ ALLTOALL_ID	▶ ISEND_ID	▶ SSEND_ID
▶ ALLTOALLV_ID	▶ ISSEND_ID	▶ SSEND_INIT_ID
▶ BARRIER_ID	▶ PROBE_ID	▶ START_ID
▶ BCAST_ID	▶ RECV_ID	▶ STARTALL_ID
▶ BSEND_ID	▶ RECV_INIT_ID	▶ TEST_ID
▶ BSEND_INIT_ID	▶ REDUCE_ID	▶ TESTALL_ID
▶ BUFFER_ATTACH_ID	▶ REDUCE_SCATTER_ID	▶ TESTANY_ID
▶ BUFFER_DETACH_ID	▶ RSEND_ID	▶ TESTSOME_ID
▶ COMM_RANK_ID	▶ RSEND_INIT_ID	▶ WAIT_ID
▶ COMM_SIZE_ID	▶ SCAN_ID	▶ WAITALL_ID
▶ GATHER_ID	▶ SCATTER_ID	▶ WAITANY_ID
▶ GATHERV_ID	▶ SCATTERV_ID	▶ WAITSOME_ID
▶ IBSEND_ID	▶ SEND_ID	

#### **double MT\_get\_mpi\_counts(int)**

Similar to the `MT_get_mpi_counts()` function, the `double MT_get_mpi_counts(int)` function returns the accumulated size of the data that is transferred by the MPI function.

#### **double MT\_get\_mpi\_time(int)**

Similar to the `MT_get_mpi_counts()` function, the `double MT_get_mpi_time(int)` function returns the accumulated time that is spent in the MPI function.

#### **double MT\_get\_avg\_hops(void)**

The distance between two processors  $p, q$  with physical coordinates  $(x_p, y_p, z_p)$  and  $(x_q, y_q, z_q)$ , is calculated as:

$$\text{Hops}(p, q) = |x_p - x_q| + |y_p - y_q| + |z_p - z_q|$$

We measure the AverageHops for all communications on a given processor as follows:

$$\text{AverageHops} = \text{sum}(\text{Hops}_i \times \text{Bytes}_i) / \text{sum}(\text{Bytes}_i)$$

In this equation,  $Hops_i$  is the distance between the processors for MPI communication, and  $Bytes_i$  is the size of the data transferred in this communication. The logical concept behind this performance metric is to measure how far each byte must travel for the communication (in average). If the communication processor pair is close to each other in the coordinate, the AverageHops value will tend to be small.

### **double MT\_get\_time(void)**

The double `MT_get_time(void)` function returns the time since `MPI_Init()` is called.

### **double MT\_get\_elapsed\_time(void)**

The double `MT_get_elapsed_time(void)` function returns the time between which `MPI_Init()` and `MPI_Finalize()` are called.

### **char \*MT\_get\_mpi\_name(int)**

The char `*MT_get_mpi_name(int)` function returns the name of an MPI ID in a string.

### **int MT\_get\_tracebufferinfo(struct MT\_tracebufferstruct \*)**

The int `MT_get_tracebufferinfo(struct MT_tracebufferstruct *)` function returns the size of a buffer used or free by the MPI Profiler or Tracer tool at the moment.

### **unsigned long MT\_get\_calleraddress(int level)**

The unsigned long `MT_get_calleraddress(int level)` function returns the caller's address in memory.

### **int MT\_get\_callerinfo(unsigned long caller memory address, struct MT\_callerstruct \*)**

This function takes the caller memory address (from `MT_get_calleraddress()`) and returns detailed caller information including the path, the source file name, the function name, and the line number of the caller in the source file.

### **void MT\_get\_environment(struct MT\_envstruct \*)**

The void `MT_get_environment(struct MT_envstruct *)` function returns its own environment information including MPI rank, physical coordinates, dimension of the block, number of total tasks, and CPU clock frequency.

### **int MT\_get\_allresults(int data type, int mpi id, struct MT\_summarystruct \*)**

This function returns statistical results, such as min, max, median, and average, on primitive performance data, for example call counts, size of data transferred, time, and so on, for specific or all MPI functions. The data type can be one of the following data types, and `mpi id` can be one of the MPI IDs listed in 1.5.3, "Utility functions" on page 13, or ALLMPI ID for all MPI functions:

- ▶ COUNTS
- ▶ BYTES
- ▶ COMMUNICATIONTIME
- ▶ STACK
- ▶ HEAP
- ▶ MAXSTACKFUNC
- ▶ ELAPSEDTIME
- ▶ AVGHOPS

## **int MT\_get\_memoryinfo(struct MT\_memorystruct \*)**

The `int MT_get_memoryinfo(struct MT_memorystruct *)` function returns information for memory usage on the compute node and is only available with the Blue Gene/L system.

In Example 1-10, we re-write the `MT_trace_event()` and `MT_output_trace()` routines with about 50 lines of code (and use the default version of `MT_output_text()`) on the Blue Gene/L system. The function automatically detects the communication pattern and shuts off the recording of trace events after the first instance of the pattern. Also only MPI ranks of less than 32 will output performance data at the end of program execution. As shown in the example, such utility functions as `MT_get_time()` and `MT_get_environment()` help the user easily obtain information that is necessary to configure the library. In this example, `MT_get_time()` returns the execution time spent so far, and `MT_get_environment()` returns the process personality including its physical coordinates and MPI rank.

*Example 1-10 Sample code for the MPI tracing configuration*

---

```
int MT_trace_event(int id) {
    ...
    now=MT_get_time();
    MT_get_environment(&env);
    ...
    /* get MPI function call distribution */
    current_event_count=MT_get_mpi_counts();
    /* compare MPI function call distribution */
    comparison_result
    =compare_dist(prev_event_count,current_event_count);
    prev_event_count=current_event_count;
    /* compare MPI function call distribution */
    if(comparison_result==1)
    return 0; /* stop tracing */
    else
    return 1; /* start tracing */
}
int MT_output_trace(int rank) {
    if (rank < 32)
    return 1; /* output performance data */
    else
    return 0; /* no output */
}
```

---

## 1.6 Related issues

In this section, we describe related issues for the MPI Profiler and Tracer.

### 1.6.1 Overhead

The library implements wrappers that use the MPI profiling interface and have the following form:

```
int MPI_Send(...) {  
    start_timing();  
    PMPI_Send(...);  
    stop_timing();  
    log_the_event();  
}
```

When event tracing is enabled, the wrappers save a time-stamped record of every MPI call for graphical display. This record adds some overhead, about 1-2 microseconds per call. The event-tracing method uses a small buffer in memory, up to  $3 \times 10^4$  events per task. Therefore, this method is best suited for short-running applications or time-stepping codes for a few steps. To further trace or profile a large scale application, configuration might be required to improve the scalability. Refer to 1.5, “Configuration” on page 10, for details.

### 1.6.2 Multithreading

The current version of the MPI profiling and tracing library is not thread-safe. Therefore, use it in single-threaded applications or when only one thread makes MPI calls. The wrappers can be made thread-safe by adding mutex locks around updates of static data. These locks can add additional overhead.



## CPU profiling using Xprofiler

Xprofiler for the Blue Gene/P system is a tool that helps you analyze your application performance. It uses data collected by the `-pg` compiler option to construct a graphical display of the functions within your application. Xprofiler provides quick access to the profiled data, which lets you identify the functions that are the most CPU intensive. The GUI also lets you manipulate the display in order to focus on the critical areas of the application.

## 2.1 Starting Xprofiler

You start Xprofiler by issuing the **Xprofiler** command from the command line. You must also specify the executable, profile data file or files, and options, which you can do in one of two ways. You can either specify them on the command line, with the **Xprofiler** command, or you can issue the **Xprofiler** command alone and then specify the options from within the GUI.

To start Xprofiler and specify the executable, profile data file or files, and options from the command line, enter:

```
Xprofiler a.out gmon.out... [options]
```

*a.out* is the name of your binary executable file, and *gmon.out* is the name of your profile data file or files. *options* can be one or more of the options listed in Table 2-1.

Table 2-1 Xprofiler options

Option	Syntax	Description
-b	Xprofiler -b a.out gmon.out	This option suppresses the printing of the field descriptions for the Flat Profile, Call Graph Profile, and Function Index reports when they are written to a file with the Save As option of the File menu.
-s	Xprofiler -s a.out gmon.out.1 gmon.out.2 gmon.out.3	If multiple gmon.out files are specified when Xprofiler is started, this option produces the gmon.sum profile data file. The gmon.sum file represents the sum of the profile information in all the specified profile files. Note that if you specify a single gmon.out file, the gmon.sum file contains the same data as the gmon.out file.
-z	Xprofiler -z a.out gmon.out	This option includes functions that have both zero CPU usage and no call counts in the Flat Profile, Call Graph Profile, and Function Index reports. A function will not have a call count if the file that contains its definition was not compiled with the -pg option, which is common with system library files.
-a	Xprofiler -a pathA:@:pathB	This option adds alternative paths to search for source code and library files, or changes the current path search order. When using this command line option, you can use the at sign (@) to represent the default file path, in order to specify that other paths be searched before the default path.
-c	Xprofiler a.out gmon.out -c config_file_name	This option loads the specified configuration file. If the -c option is used on the command line, the configuration file name specified with it is displayed in the Configuration File (-c): text field, in the Loads Files window, and the Selection field of the Load Configuration File window. When both the -c and -disp_max options are specified on the command line, the -disp_max option is ignored. However, the value that was specified with it is displayed in the Initial Display (-disp_max): field in the Load Files window the next time it is opened.
-disp_max	Xprofiler -disp_max 50 a.out gmon.out	This option sets the number of function boxes that Xprofiler initially displays in the function call tree. The value that is supplied with this flag can be any integer between 0 and 5,000. Xprofiler displays the function boxes for the most CPU-intensive functions through the number that you specify. For instance, if you specify 50, Xprofiler displays the function boxes for the 50 functions in your program that consume the most CPU. After this, you can change the number of function boxes that are displayed via the Filter menu options. This flag has no effect on the content of any of the Xprofiler reports.

Option	Syntax	Description
-e	Xprofiler -e foo -e bar a.out gmon.out	<p>This option de-emphasizes the general appearance of the function box or boxes for the specified function or functions in the function call tree. This option also limits the number of entries for these function in the Call Graph Profile report. This also applies to the specified function's descendants, as long as they have not been called by non-specified functions.</p> <p>In the function call tree, the function box or boxes for the specified function or functions appears to be unavailable. Its size and the content of the label remain the same. This also applies to descendant functions, as long as they have not been called by non-specified functions.</p> <p>In the Call Graph Profile report, an entry for the specified function only appears where it is a child of another function or as a parent of a function that also has at least one non-specified function as its parent. The information for this entry remains unchanged. Entries for descendants of the specified function do not appear unless they have been called by at least one non-specified function in the program.</p>
-E	Xprofiler -E foo -E bar a.out gmon.out	<p>This option changes the general appearance and label information of the function box or boxes for the specified function or functions in the function call tree. In addition, this option limits the number of entries for these functions in the Call Graph Profile report and changes the CPU data that is associated with them. These results also apply to the specified function's descendants, as long as they have not been called by non-specified functions in the program.</p> <p>In the function call tree, the function box for the specified function appears to be unavailable, and its size and shape also change so that it appears as a square of the smallest allowable size. In addition, the CPU time shown in the function box label appears as zero. The same applies to function boxes for descendant functions, as long as they have not been called by non-specified functions. This option also causes the CPU time spent by the specified function to be deducted from the left side CPU total in the label of the function box for each of the specified ancestors of the function.</p> <p>In the Call Graph Profile report, an entry for the specified function only appears where it is a child of another function or as a parent of a function that also has at least one non-specified function as its parent. When this is the case, the time in the self and descendants columns for this entry is set to zero. In addition, the amount of time that was in the descendants column for the specified function is subtracted from the time listed under the descendants column for the profiled function. As a result, be aware that the value listed in the % time column for most profiled functions in this report will change.</p>

Option	Syntax	Description
-f	Xprofiler -f foo -f bar a.out gmon.out	<p>This option de-emphasizes the general appearance of all function boxes in the function call tree, except for that of the specified function or functions and its descendant or descendants. In addition, the number of entries in the Call Graph Profile report for the non-specified functions and non-descendant functions is limited. The -f flag overrides the -e flag.</p> <p>In the function call tree, all function boxes, except for that of the specified function or functions and its descendant or descendants, appear to be unavailable. The size of these boxes and the content of their labels remain the same. For the specified function or functions, and its descendant or descendants, the appearance of the function boxes and labels remains the same.</p> <p>In the Call Graph Profile report, an entry for a non-specified or non-descendant function only appears where it is a parent or child of a specified function or one of its descendants. All information for this entry remains the same.</p>
-F	Xprofiler -F foo -F bar a.out gmon.out	<p>This option changes the general appearance and label information of all function boxes in the function call tree, except for that of the specified function or functions and its descendants. In addition, the number of entries in the Call Graph Profile report for the non-specified and non-descendant functions is limited, and the CPU data associated with them is changed. The -F flag overrides the -E flag.</p> <p>In the function call tree, all function boxes, except for that of the specified function or functions and its descendant or descendants, appear to be unavailable. The size and shape of these boxes change so that they are displayed as squares of the smallest allowable size. In addition, the CPU time shown in the function box label appears as zero.</p> <p>In the Call Graph Profile report, an entry for a non-specified or non-descendant function only is displayed where it is a parent or child of a specified function or one of its descendants. When this is the case, the time in the self and descendants columns for this entry is set to zero. As a result, be aware that the value listed in the % time column for most profiled functions in this report will change.</p>
-L	Xprofiler -L /lib/profiled	<p>This option sets the path name for locating shared libraries. If you plan to specify multiple paths, use the Set File Search Paths option of the File menu on the Xprofiler GUI.</p>

## 2.2 Understanding the Xprofiler display

The primary difference between Xprofiler and the UNIX® **gprof** command is that Xprofiler gives a graphical picture of the CPU consumption of your application in addition to textual data. This information allows you to focus quickly on the areas of your application that consume a disproportionate amount of CPU.

Xprofiler displays your profiled program in a single main window. It uses several types of graphic images to represent the relevant parts of your program. Functions are displayed as solid green boxes, called *function boxes*, and the calls between them are displayed as blue arrows, called *call arcs*. The function boxes and call arcs that belong to each library within your application are displayed within a fenced-in area called a *cluster box*.



The Xprofiler main window contains a graphical representation of the functions and calls within your application as well as their inter-relationships. In the main window, Xprofiler shows the function call tree. The function call tree shows the function boxes, call arcs, and cluster boxes that represent the functions within your application.

When Xprofiler first opens, by default, the function boxes for your application are clustered by library. This type of clustering means that a cluster box appears around each library, and the function boxes and call arcs within the cluster box are reduced in size. If you want to see more detail, you must uncluster the functions by selecting **File → Uncluster Functions**.

## 2.2.1 Xprofiler main menus

Along the upper portion of the main window is the menu bar. The left side of the menu bar contains the Xprofiler menus that let you work with your profiled data. In this section, we describe each of the menus:

- ▶ File menu

With the File menu, you specify the executable (a.out) files and profile data (gmon.out) files that Xprofiler will use. You also use this menu to control how your files are accessed and saved.

- ▶ View menu

You use the View menu to help you focus on portions of the function call tree, in the Xprofiler main window, in order to have a better view of the application's critical areas.

- ▶ Filter menu

Using the Filter menu, you can add, remove, and change specific parts of the function call tree. By controlling what Xprofiler displays, you can focus on the objects that are most important to you.

- ▶ Report menu

The Report menu provides several types of profiled data in a textual and tabular format. With the options of the Report menu, you can display textual data, save it to a file, view the corresponding source code, or locate the corresponding function box or call arc in the function call tree, in addition to presenting the profiled data.

- ▶ Utility menu

The Utility menu contains one option, *Locate Function By Name*, with which you can highlight a particular function box in the function call tree.

- ▶ Function menu

You can perform a number of operations for any of the functions shown in the function call tree by using the Function menu. You can access statistical data, look at source code, and control which functions are displayed.

The Function menu is not visible from the Xprofiler window. To access it, you right-click the function box of the function in which you are interested. By doing this, you only open the Function menu and also select this function. Then, when you select actions from the Function menu, they are applied to this function.

- ▶ Arc menu

With the Arc menu, you can locate the caller and callee functions for a particular call arc. A call arc represents a call between two functions within the function call tree.

The Arc menu is not visible from the Xprofiler window. You access it by right-clicking the call arc in which you are interested. By doing this, you open the Arc menu and also select

that call arc. Then, when you perform actions with the Arc menu, they are applied to that call arc.

► **Cluster Node menu**

Using the Cluster Node menu, you can control the way your libraries are displayed by Xprofiler. In order to access the Cluster Node Menu, the function boxes, in the function call tree, must first be clustered by library. When the function call tree is clustered, all the function boxes within each library are displayed within a cluster box.

The Cluster Node menu is not visible from the Xprofiler window. You access it by right-clicking the edge of the cluster box in which you are interested. By doing this, you open the Cluster Node menu and also select that cluster. Then, when you perform actions with the Cluster Node menu, they are applied to the functions within that library cluster. Display Status Field at the bottom of the Xprofiler window is a single field that tells you:

- The name of your application.
- The number of gmon.out files used in this session.
- The total amount of CPU used by the application.
- The number of functions and calls in your application and how many are currently displayed.

## 2.2.2 Elements of the function call tree

The graphical representation of the functions within a program are displayed in the main window of Xprofiler. Each function can be viewed individually or grouped into cluster boxes. In this section, we describe how to interpret and manipulate the functions contained in the display.

### Functions

Functions are represented by green, solid-filled boxes in the function call tree:

- The *size and shape* of each function box indicates its CPU usage.
- The *height* of each function box represents the amount of CPU time it spent on executing itself.
- The *width* of each function box represents the amount of CPU time it spent on executing itself, plus its descendant functions.

As a result, a function box that is wide and flat represents a function that uses a relatively small amount of CPU on itself. That is, it spends most of its time on its descendants. However, the function box for a function that spends most of its time executing only itself is roughly square shaped.

Under each function box in the function call tree is a label that contains the name of the function and related CPU usage data. For information about the function box labels, see 2.3, “Getting performance data for your application” on page 31.

Figure 2-1 shows the function boxes for two functions, sub1 and printf, as they might appear in the Xprofiler display. Each function box has its own menu. To access it, move your mouse pointer over the function box of the function in which you are interested, and right-click. Each function also has an information box from which you can get basic performance information quickly. To access the information box, move your mouse pointer over the function box of the function in which you are interested, and click.

The calls made between each of the functions in the function call tree are represented by blue arrows that extend between their corresponding function boxes. These lines are called *call arcs*. Each call arc appears as a solid blue line between two function boxes. The arrowhead

indicates the direction of the call. The function represented by the function box it points to is the one that receives the call. The function that makes the call is known as the *caller*, while the function receiving the call is known as the *callee*.

Each call arc includes a numerical label that tells you the number of calls that were exchanged between the two corresponding functions.

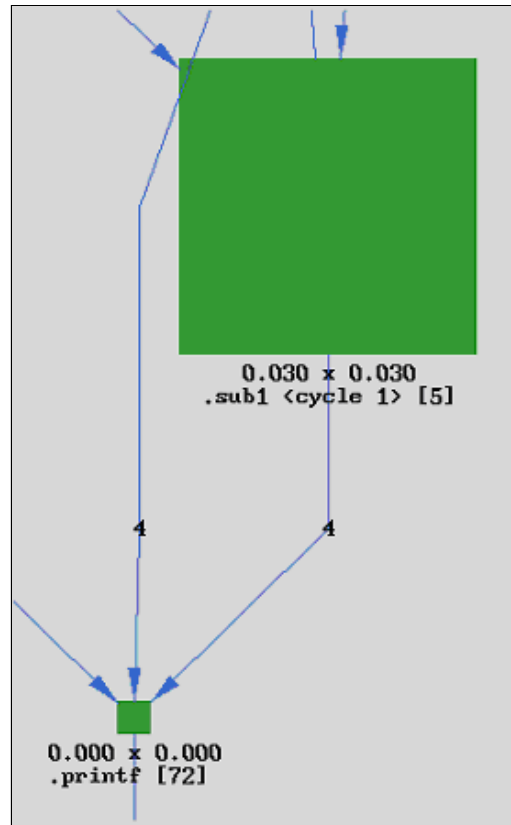


Figure 2-1 Sample functions

## Library clusters

With Xprofiler, you can collect the function boxes and call arcs that belong to each of your shared libraries into cluster boxes. Since there is a box around each library, the individual function boxes and call arcs are difficult to see. If you want to see more detail, you must uncluster the function boxes by selecting **Filter** → **Unclassify Functions**.

When viewing function boxes within a cluster box, notice that the size of each function box is relative to those of the other functions within the same library cluster. However, when all the libraries are unclustered, the size of each function box is relative to all the functions in the application, as shown in the function call tree.

Each library cluster has its own menu with which you can manipulate the cluster box. To access it, move your mouse pointer over the edge of the cluster box in which you are interested and right-click. Each cluster also has an information box that shows you the name of the library and the total CPU usage (in seconds) for the functions within it. To access the information box, move your mouse pointer over the edge of the cluster box in which you are interested and click.

### 2.2.3 Manipulating the Xprofiler display

You can look at your profiled data a number of ways by using Xprofiler, depending on what you want to see. Xprofiler provides the following functions:

- ▶ Navigation that lets you move around the display and zoom in on specific areas
- ▶ Display options, based on your personal viewing preferences
- ▶ Filtering capability so that you can include and exclude certain objects from the display
- ▶ Zooming in on the function call tree
- ▶ Filtering capabilities on what you see
- ▶ Clustering together of libraries
- ▶ Location of specific objects in the function call tree
- ▶ Customization of Xprofiler resources

We explain some of these functions in the sections that follow.

#### Zooming in on the function call tree

With Xprofiler, you can magnify specific areas of the window to gain a better view of your profiled data. The View menu includes the following options for you to do this:

- ▶ Overview
- ▶ Zoom In
- ▶ Zoom Out

To resize a specific area of the display, select **View** → **Overview**. In the Overview Window is a miniature view of the function call tree, just as it is displayed in the Xprofiler main window. When you open the Overview window, the highlighted area represents the current view of the main window.

You control the size and placement of the highlighted area with your mouse. Depending on where you place your cursor over the highlighted area, your cursor changes to indicate the operation that you can perform:

- ▶ Two crossed arrows

When your cursor appears as two crossed arrows, you can control where the box is placed by clicking and holding down your left mouse button.

- ▶ Line with perpendicular arrow

When your cursor appears as a line with an arrow perpendicular to it, your mouse pointer has grabbed the edge of the highlighted area, and you can now resize it.

By pressing and holding down your left mouse button, and then dragging it in or out, you can increase or decrease the size of the box. Notice that, as you move the edge in or out, the size of the entire highlighted area changes.

- ▶ Right angle with pointed arrow

When your cursor appears as a right angle with an arrow pointing into it, your mouse pointer has grabbed the corner of the highlighted area and you can now resize it.

By pressing and holding down your left mouse button, and dragging it diagonally up or down, you can increase or decrease the size of the box. Notice that, as you move the corner up or down, the size of the entire highlighted area changes.

You can also zoom in or out on a specific area of the function call tree:

1. Place your cursor within the light blue highlighted area. Notice that the cursor changes to four crossed arrows. This indicates that your cursor has control over the placement of the box.
2. Move your cursor over one of the four corners of the highlighted area. Notice that the cursor changes to a right angle with an arrow pointing into it. This indicates that you now have control over the corner of the highlighted area.
3. Press and hold down your left mouse button, and drag the corner of the box diagonally inward. The box shrinks as you move it.
4. When the highlighted area is as small as you want it, release the mouse button. The Xprofiler main display redraws itself to contain only the functions within the highlighted area, and in the same proportions. This function has the effect of magnifying the items within the highlighted area.
5. Move your cursor over the highlighted area. Again it changes to four crossed arrows to indicate that you have control over placement of the highlighted area. Press and hold down the left mouse button and drag the highlighted area to the area of the Xprofiler display that you want to magnify.
6. Release the mouse button. The main Xprofiler display now contains the items in which you are interested.

## **Filtering your view**

When Xprofiler first opens, the entire function call tree is displayed in the main window. This includes the function boxes and call arcs that belong to your executable as well as the shared libraries that it uses. At times, you might want to simplify what you see in the main window. There are a number of ways to do this.

Using the Filtering options of the Filter menu, you can change the appearance of the function call tree only. The performance data contained in the reports, via the Reports menu, is not affected.

### ***Displaying the entire function call tree***

When you first open Xprofiler, all the function boxes and call arcs of your executable and its shared libraries appear in the main window (Figure 2-2 on page 26). After that, you can choose to filter out specific items from the window. However, there might be times when you want to see the entire function call tree again without reloading your application. To display the entire tree, select **Filter** → **Show Entire Call Tree**. Xprofiler erases whatever is currently displayed in the main window and replaces it with the entire function call tree.

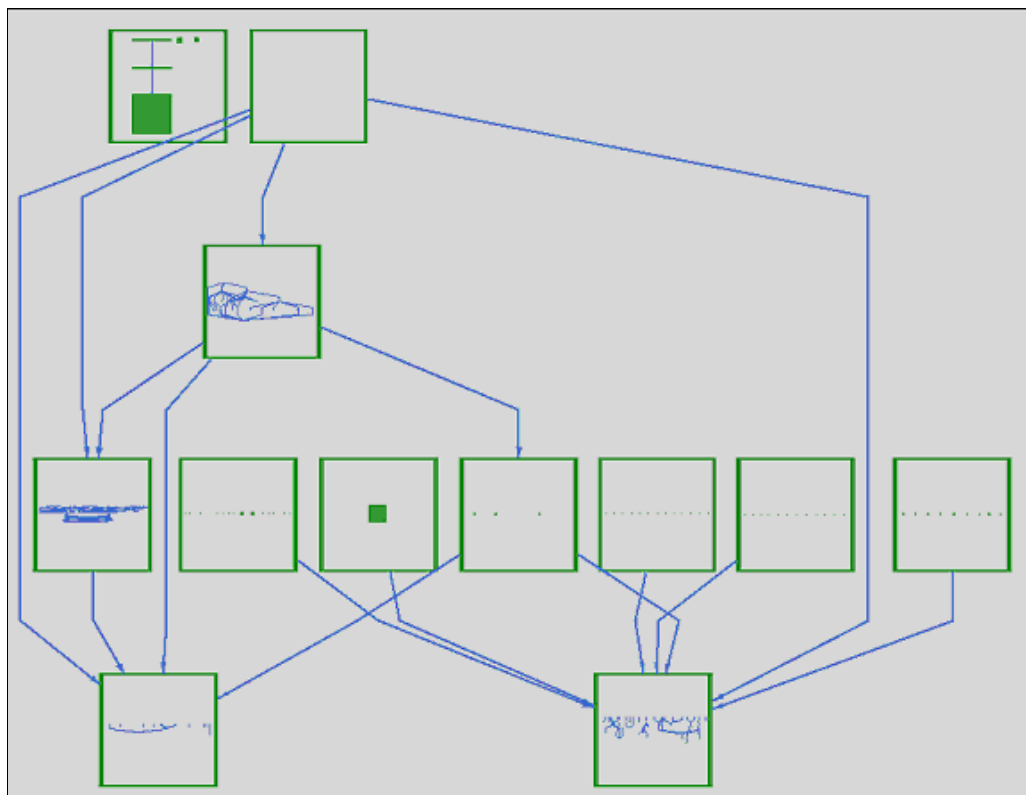


Figure 2-2 Xprofiler before filters

### ***Filtering shared library functions***

In most cases, your application will call functions that are within shared libraries. This means that these shared libraries will appear in the Xprofiler window along with your executable. As a result, the window can become crowded and obscure the items that you really want to see. If this is the case, you might want to filter the shared libraries from the display.

To filter the shared libraries, select **Filter** → **Remove All Library Calls**. The shared library function boxes disappear from the function call tree, leaving only the function boxes of your executable file visible (see Figure 2-3).

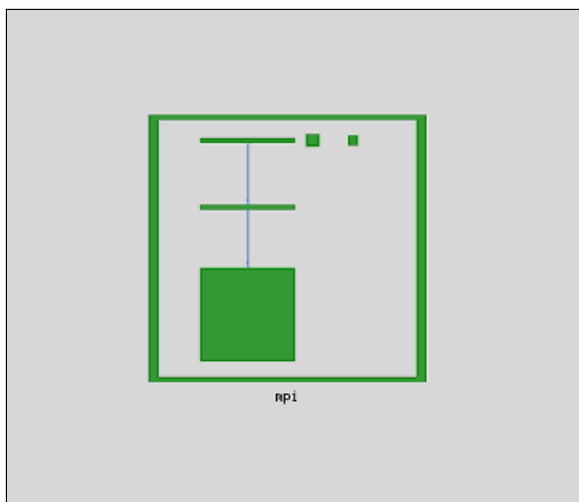


Figure 2-3 Xprofiler with library calls filtered

If you removed the library calls from the display, you might want to add all or some of the library calls back. To add library calls to the display, select **File** → **Add Library Calls**. Once again, the function boxes are displayed with the function call tree. Note, however, that all of the shared library calls that were in the initial function call tree might not be added back. The Add Library Calls option only adds back the function boxes for the library functions that were called by functions that are currently displayed in the Xprofiler window.

If you only want to add specific function boxes back to the display, select **Filter** → **Filter by Function Names** (see Figure 2-4). When the window opens, click the **Add these functions to graph** button. Then in the Enter Function name field, type the name of the function. You can add more than one function by using this method. Each of the functions that you enter must be separated by a space.

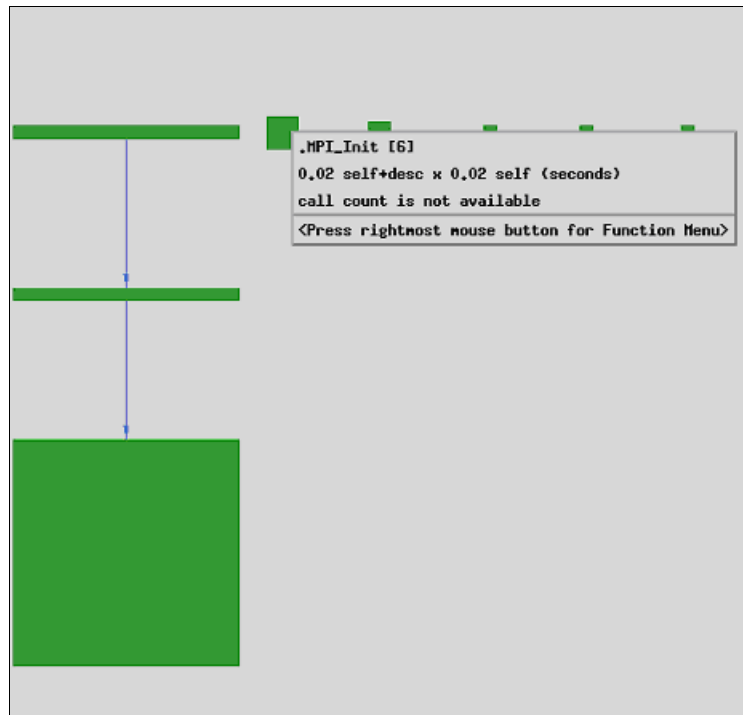


Figure 2-4 Xprofiler with the function box added

If multiple functions in your program include the strings that you enter in their names, the filter applies to each one. For example, say that you specified `sub` and `print`, and your program also included the functions named `sub1`, `psub1`, and `printf`. The `sub`, `sub1`, `psub1`, `print`, and `printf` functions will all be added to the graph.

### Filtering by function characteristics

The Filter menu of Xprofiler offers three options to add or subtract function boxes from the main window, based on specific characteristics:

- ▶ Filter by function name
- ▶ Filter by CPU time
- ▶ Filter by call counts

Each one of these options uses a different window so that you can specify the criteria by which you want to include or exclude function boxes from the window.

### ***Filter by function name***

To filter by function name, select **Filter** → **Filter by Function Names**. A window opens that presents three options:

- ▶ Add these functions to graph
- ▶ Remove these functions from the graph
- ▶ Display only these functions

Click the button for the option you want, and then, in the Enter function name field, type the name of the function to which you want the filter applied to. For example, if you want to remove the function box for a function called `fprint`, from the main window, you click the **Remove this function from the graph** button.

Then in the Enter function name field, type `fprint`. You can enter more than one function name in this field. If there are multiple functions in your program that include the strings that you enter in their names, the filter applies to each one. For example, you specified `sub` and `print`, and your program also included the functions named `sub1`, `psub1`, and `printf`. The `sub`, `sub1`, `psub1`, `print`, and `printf` functions will all be added to the graph.

### ***Filter by CPU time***

To Filter by CPU time, select **Filter** → **Filter by CPU Time**. The Filter by CPU Time window that opens presents two options:

- ▶ Show functions consuming the most CPU time
- ▶ Show functions consuming the least CPU time

Click the button for the option that you want. Then use the slider to specify the number of function boxes that you want displayed. For example, if you want to display the function boxes for the 10 functions in your application that consumed the most CPU, you click the **Show functions consuming the most CPU** button. Then you specify 10 with the slider and click the **OK** button. The function call tree then updates to reflect the options that you selected.

### ***Filter by call counts***

To filter by call counts, select **Filter** → **Filter by Call Counts**. A window opens that provides two choices:

- ▶ Show arcs with the most call counts
- ▶ Show arcs with the least call counts

Click the button for the option that you want. Then use the slider to specify the number of arcs that you want displayed. For example, if you want to display the 10 call arcs in your application that represented the least number of calls, you click the **Show arcs with the least call counts** button and specify 10 with the slider.

## **Including and excluding parent and child functions**

When tuning the performance of your application, you will want to know which functions consumed the most CPU time. Then you will need to ask several questions in order to understand their behavior:

- ▶ Where did each function spend most of the CPU time?
- ▶ What other functions *called* this function?
- ▶ Were the calls made directly or indirectly?
- ▶ What other functions *did* this function *call*?
- ▶ Were the calls made directly or indirectly?



When you understand how these functions behave and can improve their performance, you can move on to analyzing the functions that consume less CPU.

When your application is large, the function call tree will also be large. As a result, the functions that are the most CPU-intensive might be difficult to see in the function call tree. To work around this, select **Filter** → **Filter by CPU**, which lets you display only the function boxes for the functions that consume the most CPU time. After you have done this, the Function menu for each function lets you add the parent and descendant function boxes to the function call tree. By doing this, you create a smaller, simpler function call tree that displays the function boxes associated with the most CPU-intensive area of the application.

A *child function* is one that is directly called by the function of interest. To see only the function boxes for the function of interest and its child functions:

1. Move your mouse pointer over the function box in which you are interested.
2. Right-click to access the **Function** menu, and select **Immediate Children** → **Show Child Functions Only**.

Xprofiler erases the current display and replaces it with only the function boxes for the function you chose, plus its child functions.

A *parent function* is one that directly calls the function of interest. To see only the function boxes for the function of interest and its parent functions:

1. Move your mouse pointer over the function box in which you are interested.
2. Right-click to access the **Function** menu, and select **Immediate Parents** → **Show Parent Functions Only**.

Xprofiler erases the current display and replaces it with only the function boxes for the function that you chose plus its parent functions.

There might be times when you want to see the function boxes for both the parent and child functions of the function in which you are interested, without erasing the rest of the function call tree. This is especially true if you chose to display the function boxes for two or more of the most CPU-intensive functions with the Filter by CPU option of the Filter menu. You suspect that more than one function is consuming too much CPU. To see these function boxes for both the parent and child functions:

1. Move your mouse pointer over the function box in which you are interested.
2. Right-click to access the **Function** menu, and select **Immediate Parents** → **Add Parent Functions to Tree**. Xprofiler leaves the current display as it is, but adds the parent function boxes.
3. Move your mouse pointer over the same function box.
4. Right-click to access the **Function** menu, and select **Immediate Children** → **Add Child functions to Tree**. Xprofiler leaves the current display as it is, but now adds the child function boxes in addition to the parent function boxes.

## Locating specific objects in the function call tree

If you are interested in one or more specific functions in a complex program, you might need help locating their corresponding function boxes in the function call tree. If you want to locate a single function and you know its name, you can use the Locate Function By Name option of the Utility menu.

To locate a function by name:

1. Select the **Utility** → **Locate Function By Name** option.
2. In the Search By Function Name Dialog window, in the Enter Function Name field, type the name of the function you want to locate. The function name you type here must be a continuous string. It cannot include blanks. Then click either the **OK** or **Apply** button.

The corresponding function box is highlighted (its color changes to red) in the function call tree and Xprofiler zooms in on its location. To display the function call tree in full detail again, select **View** → **Overview**.

There might also be times when you want to see only the function boxes for the functions that you are concerned with, plus other specific functions that are related to it. For instance, suppose you want to see all the functions that directly called the function in which you are interested. It is not easy to select these function boxes when you view the entire call tree, so you want to display them, plus the function of interest alone. Each function has its own menu, called a *Function menu*. Via the Function menu, you can choose to see the following functions for the function in which you are interested:

- ▶ Parent functions: Functions that directly call the function of interest
- ▶ Child functions: Functions that are directly called by the function of interest
- ▶ Ancestor functions: Functions that can call, directly or indirectly, the function of interest
- ▶ Descendant functions: Functions that can be called, directly or indirectly, by the function of interest
- ▶ Functions that belong to the same cycle

When you use these options, Xprofiler erases the current display and replaces it with only the function boxes for the function of interest and all the functions of the type that you specified.

### Locating and displaying parent functions

A *parent* is any function that directly calls the function in which you are interested. To locate the parent function boxes of the function in which you are interested:

1. Right-click the function box of interest to open the Function menu.
2. From the **Function** menu, select **Immediate Parents** → **Show Parent Functions Only**. Xprofiler redraws the display to show you only the function boxes for the function of interest and its parent functions.

### Locating and displaying child functions

A *child* is any function that is directly called by the function in which you are interested. To locate the function boxes for the children of the function in which you are interested:

1. Right-click the function box of interest to open the Function menu.
2. From the **Function** menu, select **Immediate Children** → **Show Child Functions Only**. Xprofiler redraws the display to show you only the function boxes for the function of interest and its child functions.

## Locating and displaying ancestor functions

An *ancestor* is any function that can call, directly or indirectly, the function in which you are interested. To locate ancestor functions:

1. Right-click the function box of interest to open the Function menu.
2. From the **Function** menu, select **All Paths To → Show Ancestor Functions Only**. Xprofiler redraws the display to show you only the function boxes for the function of interest and its ancestor functions.

## Locating and displaying descendant functions

A *descendant* is any function that can be called, directly or indirectly, by the function in which you are interested. To locate the descendant functions:

1. Right-click the function box of interest to open the Function menu.
2. From the **Function** menu, select **All Paths From → Show Descendant Functions Only**. Xprofiler redraws the display to show you only the function boxes for the function of interest and its descendant functions.

## Locating and displaying functions on a cycle

To locate the functions that are on the same cycle as the function in which you are interested:

1. Right-click the function box of interest to open the Function menu.
2. From the **Function** menu, select **All Functions on the Cycle → Show Cycle Functions Only**. Xprofiler redraws the display to show you only the function of interest and all the other functions on its cycle.

## 2.3 Getting performance data for your application

With Xprofiler, you can obtain performance data for your application on a number of levels and in a number of ways. You can easily view data that pertains to a single function, or you can use the reports that are provided to obtain information about your application as a whole.

### Getting basic data

Xprofiler makes it easy to obtain data on specific items in the function call tree. After you locate the item in which you are interested, you can gather function data, call data, or cluster data.

#### **Basic function data**

Below each function box in the function call tree is a label that contains basic performance data. Figure 2-5 shows the function box for the function `main` and its label.

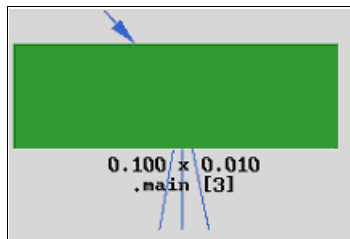


Figure 2-5 Function box

The label contains the following information:

- ▶ The name of the function, its associated cycle, if any, and its index  
In the example in Figure 2-5, the name of the function is main, and its index is [3]. It is not associated with a cycle.
- ▶ The total amount of CPU time (in seconds) this function spent on itself plus the amount of CPU time it spent on its descendants (the number to the left of the x)  
In Figure 2-5, the function main spent 0.100 seconds on itself, plus its descendants.
- ▶ The amount of CPU time (in seconds) this function spent only on itself (the number to the right of the x)  
In Figure 2-5, the function main spent 0.010 seconds on itself.

Since labels are not always visible in the Xprofiler window when it is fully zoomed out, you might need to zoom in to see the labels.

### **Basic call data**

Call arc labels are displayed over each call arc as shown in Figure 2-6. The label shows the number of calls that were made between the two functions (from caller to callee).

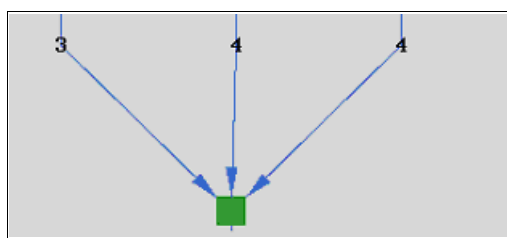


Figure 2-6 Call arc labels

### **Basic cluster data**

Cluster box labels indicate the name of the library that is represented by that cluster. If it is a shared library, the label shows its full path name.

### **Information boxes**

For each function box, call arc, and cluster box, there is a corresponding information box that you can access with your mouse. It provides the same basic data that is displayed on the label. This is useful when the Xprofiler display is fully zoomed out and the labels are not visible. To access the information box, click the function box, call arc, or cluster box (place it over the edge of the box).

For a function, the information box contains the following details:

- ▶ The name of the function, its associated cycle, if any, and its index
- ▶ The amount of CPU used by this function  
Two values are supplied in this field:
  - The first value is the amount of CPU time that is spent on this function plus the time spent on its descendants.
  - The second value represents the amount of CPU time this function spent only on itself.
- ▶ The number of times this function was called (by itself or any other function in the application)

For a call, the information box contains the following details:

- ▶ The caller and callee functions (their names) and their corresponding indexes
- ▶ The number of times the caller function called the callee

For a cluster, the information box contains the following details:

- ▶ The name of the library
- ▶ The total CPU usage (in seconds) used by the functions within it

## Statistics Report option of the Function menu

You can obtain performance statistics for a single function via the Statistics Report option of the Function menu. With this option, you can see data on the CPU usage and call counts of the selected function. If you are using more than one gmon.out file, this option breaks down the statistics per each gmon.out file that you use.

When you select the Statistics Report menu option, the Function Level Statistics Report window opens and contains the following information:

- ▶ Function name

The name of the function you selected.

- ▶ Summary data

The total amount of CPU used by this function and the number of times it was called. If you used multiple gmon.out files, the value shown here represents their sum.

- ▶ CPU usage

The amount of CPU used by this function. Two values are supplied in this field:

- The first value is the amount of CPU time spent on this function plus the time spent on its descendants.
- The second value represents the amount of CPU time this function spent only on itself.

- ▶ Call counts

The number of times this function called itself, plus the number of times it was called by other functions:

- The average (Average) number of calls made to this function, or by this function, per gmon.out file.
- Standard Deviation (Std Dev), which is the value that represents the difference in call count sampling, per function, from one gmon.out file to another. A small standard deviation value in this field means that the function was almost always called the same number of times in each gmon.out file.
- The maximum (Maximum) number of calls made to this function or by this function in a single gmon.out file. The corresponding gmon.out file is displayed to the right.
- The minimum (Minimum) number of calls made to this function or by this function in a single gmon.out file. The corresponding gmon.out file is displayed to the right.

- ▶ Statistics data

The CPU usage and calls made to or by this function, broken down by gmon.out file:

- The average (Average) CPU time used by the data in each gmon.out file.
- Standard deviation (Std Dev) which is the value that represents the difference in CPU usage samplings, per function, from one gmon.out file to another. The smaller the standard deviation is, the more balanced the workload is.

- Of all the gmon.out files, the maximum (Maximum) amount of CPU time used by all calls to this function. The corresponding gmon.out file is displayed to the right.
- Of all the gmon.out files, the minimum (Minimum) amount of CPU time used by all calls to this function. The corresponding gmon.out file is displayed to the right.

## Getting detailed data via reports

Xprofiler provides CPU usage data in textual and tabular format. This data is provided in various tables called *reports*. If you are a **gprof** user, you are familiar with the Flat Profile, Call Graph Profile, and Function Index reports. Xprofiler generates these same reports, in the same format, plus two others.

You can access the Xprofiler reports from the Report menu. With the Report menu, you can see the following reports:

- ▶ Flat Profile
- ▶ Call Graph Profile
- ▶ Function Index
- ▶ Function Call Summary
- ▶ Library Statistics

Each report window also includes a Search Engine field, which is located at the bottom of the window. Using the Search Engine field, you can search for a specific string in the report. Each of the Xprofiler reports is explained in the sections that follow.

### ***Flat Profile report***

When you select the Flat Profile menu option, the Flat Profile window opens. The Flat Profile report shows the total execution times and call counts for each function, including shared library calls, within the application. The entries for the functions that use the greatest percentage of the total CPU usage are displayed at the top of the list, while the remaining functions appear in descending order, based on the amount of time used.

Note that the data that is presented in the Flat Profile window is the same data that is generated with the UNIX **gprof** command.

### ***Call Graph Profile report***

Using the Call Graph Profile menu option, you can view the functions of your application, sorted by the percentage of total CPU usage that each function, and its descendants, consumed. When you select this option, a new Call Graph Profile window opens.

### ***Function Index report***

With the Function Index menu option, you can view a list of the function names that are included in the function call tree. When you select this option, the Function Index window opens and shows the function names in alphabetical order. To the left of each function name is its index, enclosed in brackets, for instance [2].

### ***Fields of the Function Call Summary window***

The Function Call Summary window contains the following fields:

- ▶ %total  
The percentage of the total number of calls generated by this caller-callee pair.
- ▶ calls  
The number of calls attributed to this caller-callee pair.

- function

The name of the caller function and callee function.

### ***Library Statistics report***

When you select the Library Statistics menu option, a window opens that shows the CPU time consumed and call counts of each library within your application.

## **Viewing the source and disassembler code**

Xprofiler provides several ways for you to view your code. You can view the source or disassembler code for your application on a per-function basis. This ability also applies to any included code your application may use.

When you view source or included function call code, you use the Source Code window. When you view disassembler code, you use the Disassembler Code window. You can access these windows through the Report menu of the Xprofiler GUI or the Function menu of the function that you have selected.

### ***Viewing source code***

Both the Function menu and Report menu provide the means for you to access the Source Code window, from which you view your code.

To access the Source Code window via the Function menu:

1. Right-click the function box to open the Function menu.
2. Select the **Show Source Code** option.

To access the Source Code window via the Report menu:

1. Select **Report → Flat Profile**.
2. In the Flat Profile window, click the entry of the function that you want to view. The entry is highlighted to show that it is selected.
3. Select **Code Display → Show Source Code**.

The Source Code window opens and shows the source code for the function that you selected.

### ***Using the Source Code window***

The Source Code window shows only the source code file for the function that you specified from the Flat Profile window or Function menu. It contains information in the following fields:

- Line

This is the source code line number.

- No. ticks per line

Each tick represents .01 seconds of CPU time used. The number that is displayed in this field represents the number of ticks that are used by the corresponding line of code. For instance, if the number 3 is displayed in this field, for a source statement, this source statement has used .03 seconds of CPU time. Note that the CPU usage data is displayed only in this field if you used the `-g` option when you compiled your application. Otherwise, this field is blank.

- Source code

This is the application's source code.

- Search Engine

Using the Search Engine field at the bottom of the Source Code window, you can search for a specific string in your source code.

The Source Code window contains the following menus:

- File menu

With the Save As option, you can save the annotated source code to a file. When you select this option, the Save File Dialog window opens. Click **Close** if you want to close the Source Code window.

- Utility menu

The Utility menu contains only one option, Show Included Functions. Using the Show Included Functions option, you can view the source code of files that are included by the application's source code. If a selected function does not have an included function file associated with it, or does not have the function file information available because the -g option was not used for compiling, the Utility menu will be unavailable.

The availability of the Utility menu serves as an indication of whether there is any included function file information associated with the selected function. Be aware that, when you display a selected function's source code, the function name shown in the Search Engine area of the Source Code window does not match any function shown in the code display if the selected function is an included function that is called by more than one function. In this case, the selected function resides in one of the included function files of the caller function. Therefore, if you cannot find the function that you selected in the Source Code window, and the Utility menu is activated, use the Utility menu to select the proper included function file. If you cannot find the selected function in the Source Code window and the Utility menu is unavailable, you can assume that the program was not compiled with the -g flag.

When you select the Show Included Functions option, the Included Functions Dialog window opens, which lists all of the included function files. Specify a file by either clicking one of the entries in the list or by typing the file name in the Selection field. Then click either the **OK** or **Apply** button. After selecting a file from the Included Functions Dialog window, the Included Function File window opens, displaying the source code for the file that you specified.

### ***Viewing disassembler code***

Both the Function and Report menus provide the means for you to access the Disassembler Code window, from which you can view your code.

To access the Disassembler Code window via the Function menu:

1. Right-click the function in which you are interested to open the Function menu.
2. From the **Function** menu, select **Show Disassembler Code** to open the Disassembler Code window.

To access the Disassembler Code window via the Report menu:

1. Select **Report** → **Flat Profile**.
2. In the Flat Profile window, click the entry of the function that you want to view. The entry highlights to show that it is selected.
3. Select **Code Display** → **Show Disassembler Code**. The Disassembler Code window opens and shows the disassembler code for the function that you selected.



### ***Using the Disassembler Code window***

The Disassembler Code window shows only the disassembler code for the function that you specified in the Flat Profile window. The Disassembler Code window contains information in the following fields:

- ▶ Address

The address of each instruction in the function that you selected from either the Flat Profile window or the function call tree.

- ▶ No. ticks per instruction

Each tick represents .01 seconds of CPU time used. The number that is displayed in this field represents the number of ticks used by the corresponding instruction. For instance, if the number 3 is in this field, this instruction uses .03 seconds of CPU time.

- ▶ Instruction

The execution instruction.

- ▶ Assembler code

The corresponding assembler code of the execution instruction.

- ▶ Source code

The line in your application's source code that corresponds to the execution instruction and assembler code.

Using the Search Engine field, at the bottom of the Disassembler Code window, you can search for a specific string in your disassembler code.





# Hardware Performance Monitoring

Hardware Performance Monitoring (HPM) was developed for performance measurement of applications running on IBM systems that support IBM PowerPC® 970, POWER4™, POWER5™, and POWER6™ processors with the AIX 5L™, Linux, or Blue Gene operating system.

## 3.1 HPM

The HPM capability on the Blue Gene/P system consists of an instrumentation library, called *libhpm*. Libhpm provides instrumented programs with a summary output for each instrumented region in a program. This library supports serial and parallel (Message Passing Interface (MPI), threaded, and mixed mode) applications, written in Fortran, C, and C++.

Libhpm is a library that provides a programming interface to start and stop performance counting for an application program. The part of the application program between the start and stop of performance counting is called an *instrumentation section*. Any such instrumentation section is assigned a unique integer number as a section identifier. Example 3-1 shows a simple case of how an instrumented program might look.

*Example 3-1 Sample instrumented program*

---

```
hpmInit( tasked, "my program" );
hpmStart( 1, "outer call" );
do_work();
hpmStart( 2, "computing meaning of life" );
do_more_work();
hpmStop( 2 );
hpmStop( 1 );
hpmTerminate( taskID );
```

---

Calls to `hpmInit()` and `hpmTerminate()` embrace the instrumented part. Every instrumentation section starts with `hpmStart()` and ends with `hpmStop()`. The section identifier is the first parameter to the latter two functions. As shown in the example, libhpm supports multiple instrumentation sections, overlapping instrumentation sections. Each instrumented section can be called multiple times. When `hpmTerminate()` is encountered, the counted values are collected and printed.

The program in Example 3-1 also shows a sample of two properly nested instrumentation sections. For section 1, we can consider the *exclusive* time and *exclusive* counter values. By that, we mean the difference of the values for section 1 and section 2. The original values for section 1 are called *inclusive values* for a matter of distinction. The terms *inclusive* and *exclusive* for the embracing instrumentation section are chosen to indicate whether counter values and times for the contained sections are included or excluded. For more details, see 3.5, “Inclusive and exclusive values” on page 84.

Libhpm supports OpenMP and threaded applications. There is only a thread-safe version of libhpm. Either a thread-safe linker invocation, such as `xlc_r` and `xlfr`, should be used or `libpthread.a` must be included in the list of libraries.

Notice that libhpm collects information and performs summarization during run time. Thus, there can be considerable overhead if instrumentation sections are inserted inside inner loops.

## 3.2 Events and groups

The hardware performance counters information is the value of special CPU registers that are incremented at certain events. The number of such registers is different for each architecture as shown in Table 3-1.

Table 3-1 Registers per architecture

Processor architecture	Number of performance counter registers
PowerPC 970	8
POWER4	8
POWER5	6
POWER5+	6
POWER6	6
Blue Gene/L	52
Blue Gene/P	256

On both AIX and Linux, kernel extensions provide *counter virtualization*. That is, the user sees private counter values for the application. On a technical side, the counting of the special CPU registers is frozen, and the values are saved whenever the application process is taken off the CPU and another process is scheduled. The counting is resumed when the user application is scheduled on the CPU.

The special CPU registers can count different events. On the IBM POWER™ CPUs, there are restrictions on which registers can count which events. Table 3-3 lists the events for the Blue Gene/P system.

Furthermore, there are many rules that restrict the concurrent use of different events. Each valid combination of assignments of events to hardware counting registers is called a *group*. To make handling easier, a list of valid groups is provided. Table 3-2 lists the valid groups. The group or event set to be used can be selected via the environment variable `HPM_EVENT_SET`. If the environment variable `HPM_EVENT_SET` is not specified, a default group is taken as indicated in Table 3-2.

Table 3-2 Default group

Processor architecture	Number of groups	Default group
PowerPC 970	41	23
POWER4	64	60
POWER5	148 (140)	137
POWER5+™	152	145
POWER6	127	195
Blue Gene/L	16	0
Blue Gene/P	4	0

The number of groups for POWER5 is 140 for AIX 5.2 and 148 for Linux and AIX 5.3. The reason for this difference is the different versions of bos.pmapi. The last group (139) was changed and eight new groups were appended.

Table 3-3 contains a list of the event names and groups on the Blue Gene/P system.

Table 3-3 Event names

Group	Counter	Event name	Event description
0	0	BGP_PU0_JPIPE_INSTRUCTIONS	PU0: Number of J-pipe instructions
0	1	BGP_PU0_JPIPE_ADD_SUB	PU0: PowerPC Add/Sub in J-pipe
0	2	BGP_PU0_JPIPE_LOGICAL_OPS	PU0: PowerPC logical operations in J-pipe
0	3	BGP_PU0_JPIPE_SHROTMK	PU0: Shift, rotate, mask instructions
0	4	BGP_PU0_IPIPE_INSTRUCTIONS	PU0: Number of I-pipe instructions
0	5	BGP_PU0_IPIPE_MULT_DIV	PU0: PowerPC Mul/Div in I-pipe
0	6	BGP_PU0_IPIPE_ADD_SUB	PU0: PowerPC Add/Sub in I-pipe
0	7	BGP_PU0_IPIPE_LOGICAL_OPS	PU0: PowerPC logical operations in I-pipe
0	8	BGP_PU0_IPIPE_SHROTMK	PU0: Shift, rotate, mask instructions
0	9	BGP_PU0_IPIPE_BRANCHES	PU0: PowerPC branches
0	10	BGP_PU0_IPIPE_TLB_OPS	PU0: PowerPC TLB operations
0	11	BGP_PU0_IPIPE_PROCESS_CONTROL	PU0: PowerPC process control
0	12	BGP_PU0_IPIPE_OTHER	PU0: PowerPC other I-pipe operations
0	13	BGP_PU0_DCACHE_LINEFILLINPROG	PU0: Number of cycles D-cache LineFillInProgress
0	14	BGP_PU0_ICACHE_LINEFILLINPROG	PU0: Number of cycles I-cache LineFillInProgress
0	15	BGP_PU0_DCACHE_MISS	PU0: Accesses to D cache that miss in D Cache
0	16	BGP_PU0_DCACHE_HIT	PU0: Accesses to D cache that hit in D Cache
0	17	BGP_PU0_DATA_LOADS	PU0: PowerPC data loads
0	18	BGP_PU0_DATA_STORES	PU0: PowerPC data stores
0	19	BGP_PU0_DCACHE_OPS	PU0: D cache operations
0	20	BGP_PU0_ICACHE_MISS	PU0: Accesses to I cache that miss in I Cache
0	21	BGP_PU0_ICACHE_HIT	PU0: Accesses to I cache that hit in I Cache
0	22	BGP_PU0_FPU_ADD_SUB_1	PU0: PowerPC FP Add/Sub (fadd, fadds, fsub, fsubs)
0	23	BGP_PU0_FPU_MULT_1	PU0: PowerPC FP Mult (fmul, fmuls)

Group	Counter	Event name	Event description
0	24	BGP_PU0_FPU_FMA_2	PU0: PowerPC FP FMA (fmadd, fmadds, fmsub, fmsubs, fnmadd, fnmadds, fnmsub, fnmsubs; one result generated per instruction, two flops)
0	25	BGP_PU0_FPU_DIV_1	PU0: PowerPC FP Div (fdiv, fdivs; Single Pipe Divide)
0	26	BGP_PU0_FPU_OTHER_NON_STORAGE_OPS	PU0: PowerPC FP remaining non-storage instructions (fabs, fnabs, frsp, fctiw, fctiwz, fres, frsqte, fsel, fmr, fneg, fcmpl, fcmpl, mffs, mcrfs, mtfsfi, mtfsf, mtfsb0, mtfsb1)
0	27	BGP_PU0_FPU_ADD_SUB_2	PU0: Dual pipe Add/Sub (fpadd, fpsub)
0	28	BGP_PU0_FPU_MULT_2	PU0: Dual pipe Mult (fpmul, fxmul, fxpmul, fxsmul)
0	29	BGP_PU0_FPU_FMA_4	PU0: Dual pipe FMAs (fpmadd, fpmnadd, fpmsub, fpnmnsub, fxmadd, fxnmadd, fxmsub, fxnmnsub, fxcpmadd, fxcsmadd, fxcpmnadd, fxcsmnadd, fxcpsub, fxcsmnsub, fxcpmnsub, fxcsmnsub, fxcnpma, fxcnpsma, fxcnpsma, fxcnsma, fxcnpsma, fxcnpsma, fxcnpsma; two results are generated per instruction, four flops)
0	30	BGP_PU0_FPU_DUAL_PIPE_OTHER_NON_STORAGE_OPS	PU0: Dual pipe remaining non-storage instructions (fpmr, fpneg, fsmr, fsneg, fxmr, fsmfp, fsmtp, fpabs, fpnabs, fsabs, fsnabs, frsp, fpctiw, fpctiwz, fpre, frsqte, fpsel, fscmp)
0	31	BGP_PU0_FPU_QUADWORD_LOADS	PU0: Quad Word Loads (ffpdx, lfpdx, lfxdx, lfxdux)
0	32	BGP_PU0_FPU_OTHER_LOADS	PU0: All other Loads (lfs, lfsx, lfsu, lfsux, lfpsx, fpsux, lfdsx, lfdsux, lfssx, lfssux, lfd, lfdx, lfdx, lfdx, lfxsx, lfxsux)
0	33	BGP_PU0_FPU_QUADWORD_STORES	PU0: Quad Word Stores (stfpdx, stfpdx, stfxdx, stfxdux)
0	34	BGP_PU0_FPU_OTHER_STORES	PU0: All other FPU Stores (stfs, stfsx, stfsu, stfsux, stfd, stfdx, stfdu, stfdx, stfiwx, stfpx, stfpx, stfpiwx, stfsdx, stfsdx, stfssx, stfssux, stfxsx, stfxsux)
0	35	BGP_PU1_JPIPE_INSTRUCTIONS	PU1: Number of J-pipe instructions
0	36	BGP_PU1_JPIPE_ADD_SUB	PU1: PowerPC Add/Sub in J-pipe
0	37	BGP_PU1_JPIPE_LOGICAL_OPS	PU1: PowerPC logical operations in J-pipe
0	38	BGP_PU1_JPIPE_SHROTMK	PU1: Shift, rotate, mask instructions
0	39	BGP_PU1_IPIPE_INSTRUCTIONS	PU1: Number of I-pipe instructions
0	40	BGP_PU1_IPIPE_MULT_DIV	PU1: PowerPC Mul/Div in I-pipe

Group	Counter	Event name	Event description
0	41	BGP_PU1_IPIPE_ADD_SUB	PU1: PowerPC Add/Sub in I-pipe
0	42	BGP_PU1_IPIPE_LOGICAL_OPS	PU1: PowerPC logical operations in I-pipe
0	43	BGP_PU1_IPIPE_SHROTMK	PU1: Shift, rotate, mask instructions
0	44	BGP_PU1_IPIPE_BRANCHES	PU1: PowerPC branches
0	45	BGP_PU1_IPIPE_TLB_OPS	PU1: PowerPC TLB operations
0	46	BGP_PU1_IPIPE_PROCESS_CONTROL	PU1: PowerPC process control
0	47	BGP_PU1_IPIPE_OTHER	PU1: PowerPC other I-pipe operations
0	48	BGP_PU1_DCACHE_LINEFILLINPROG	PU1: Number of cycles D-cache LineFillInProgress
0	49	BGP_PU1_ICACHE_LINEFILLINPROG	PU1: Number of cycles I-cache LineFillInProgress
0	50	BGP_PU1_DCACHE_MISS	PU1: Accesses to D cache that miss in D Cache
0	51	BGP_PU1_DCACHE_HIT	PU1: Accesses to D cache that hit in D Cache
0	52	BGP_PU1_DATA_LOADS	PU1: PowerPC data loads
0	53	BGP_PU1_DATA_STORES	PU1: PowerPC data stores
0	54	BGP_PU1_DCACHE_OPS	PU1: D cache operations
0	55	BGP_PU1_ICACHE_MISS	PU1: Accesses to I cache that miss in I Cache
0	56	BGP_PU1_ICACHE_HIT	PU1: Accesses to I cache that hit in I Cache
0	57	BGP_PU1_FPU_ADD_SUB_1	PU1: PowerPC FP Add/Sub (fadd, fadds, fsub, fsubs)
0	58	BGP_PU1_FPU_MULT_1	PU1: PowerPC FP Mult (fmul, fmuls)
0	59	BGP_PU1_FPU_FMA_2	PU1: PowerPC FP FMA (fmadd, fmadds, fmsub, fmsubs, fnmadd, fnmadds, fnmsub, fnmsubs; one result generated per instruction, two flops)
0	60	BGP_PU1_FPU_DIV_1	PU1: PowerPC FP Div (fddiv, fddivs; Single Pipe Divide)
0	61	BGP_PU1_FPU_OTHER_NON_STORAGE_OPS	PU1: PowerPC FP remaining non-storage instructions (fabs, fnabs, frsp, fctiw, fctiwz, fres, frsqrt, fsel, fmr, fneg, fcmpl, fcmpl, mffs, mcrfs, mtfshi, mtfsh, mtfshb0, mtfshb1)
0	62	BGP_PU1_FPU_ADD_SUB_2	PU1: Dual pipe Add/Sub (fpadd, fpsub)
0	63	BGP_PU1_FPU_MULT_2	PU1: Dual pipe Mult (fpmul, fxmul, fxpmul, fxsmul)



Group	Counter	Event name	Event description
0	64	BGP_PU1_FPU_FMA_4	PU1: Dual pipe FMA's (fpmadd, fpnmadd, fpmsub, fpnmsub, fxmadd, fxnmadd, fxmsub, fxnmsub, fxcpmadd, fxcsmadd, fxcpnadd, fxcsmadd, fxcpsub, fxcmsub, fxcpsnsub, fxcsnmsub, fxcnpma, fxcnpma, fxcpsma, fxcnsma, fxcnpma, fxcxnsma, fxcxma, fxcxns; two results generated per instruction, four flops)
0	65	BGP_PU1_FPU_DUAL_PIPE_OTHER_NON_STORAGE_OPS	PU1: Dual pipe remaining non-storage instructions (fpmr, fpneg, fsmr, fsneg, fxmr, fsmfp, fsmtp, fpabs, fpnabs, fsabs, fsnabs, fprsp, fpctiw, fpctiwz, fpre, fprsqte, ftsel, fscmp)
0	66	BGP_PU1_FPU_QUADWORD_LOADS	PU1: Quad Word Loads (ffpdx, lfpdux, lfxdx, lfxdux)
0	67	BGP_PU1_FPU_OTHER_LOADS	PU1: All other Loads (lfs, lfsx, lfsu, lfsux, lfpsx, fpsux, lfdsx, lfdsux, lfssx, lfssux, lfd, lfdx, lfdx, lfdx, lfxsx, lfxsux)
0	68	BGP_PU1_FPU_QUADWORD_STORES	PU1: Quad Word Stores (stfpdx, stfpdux, stfxdx, stfxdux)
0	69	BGP_PU1_FPU_OTHER_STORES	PU1: All other FPU Stores (stfs, stfsx, stfsu, stfsux, stfd, stfdx, stfdu, stfdx, stfiwx, stfpx, stfpx, stfpiwx, stfsdx, stfsdux, stfssx, stfssux, stfxsx, stfxsux)
0	70	BGP_PU0_L1_INVALIDATION_REQUESTS	PU0 L1: Invalidation requested
0	71	BGP_PU1_L1_INVALIDATION_REQUESTS	PU1 L1: Invalidation requested
0	72	BGP_PU0_L2_VALID_PREFETCH_REQUESTS	PU0 L2: Prefetch request valid
0	73	BGP_PU0_L2_PREFETCH_HITS_IN_FILTER	PU0 L2: Prefetch hits in filter
0	74	BGP_PU0_L2_PREFETCH_HITS_IN_STREAM	PU0 L2: Prefetch hits in active stream
0	75	BGP_PU0_L2_CYCLES_PREFETCH_PENDING	PU0 L2: Number of cycles for which L2-prefetch is pending
0	76	BGP_PU0_L2_PAGE_ALREADY_IN_L2	PU0 L2: requested PF is already in L2
0	77	BGP_PU0_L2_PREFETCH_SNOOP_HIT_SAME_CORE	PU0 L2: Prefetch snoop hit from the same core (write)
0	78	BGP_PU0_L2_PREFETCH_SNOOP_HIT_OTHER_CORE	PU0 L2: Prefetch snoop hit from the other core
0	79	BGP_PU0_L2_PREFETCH_SNOOP_HIT_PLB	PU0 L2: Prefetch snoop hit PLB (write)
0	80	BGP_PU0_L2_CYCLES_READ_REQUEST_PENDING	PU0 L2: Number of cycles for which a read request is pending
0	81	BGP_PU0_L2_READ_REQUESTS	PU0 L2: Read requests
0	82	BGP_PU0_L2_DEVBUS_READ_REQUESTS	PU0 L2: Devbus read requests (for SRAM, LOCK and UPC)
0	83	BGP_PU0_L2_L3_READ_REQUESTS	PU0 L2: L3 read request

Group	Counter	Event name	Event description
0	84	BGP_PU0_L2_NETBUS_READ_REQUESTS	PU0 L2: Netbus read requests (for tree and torus)
0	85	BGP_PU0_L2_BLIND_DEV_READ_REQUESTS	PU0 L2: BLIND device read request
0	86	BGP_PU0_L2_PREFETCHABLE_REQUESTS	PU0 L2: Prefetchable requests
0	87	BGP_PU0_L2_HIT	PU0 L2: L2 hit
0	88	BGP_PU0_L2_SAME_CORE_SNOOPS	PU0 L2: Same core snoops
0	89	BGP_PU0_L2_OTHER_CORE_SNOOPS	PU0 L2: Other core snops
0	90	BGP_PU0_L2_OTHER_DP_PU0_SNOOPS	PU0 L2: Other DP PU0 snoops
0	91	BGP_PU0_L2_OTHER_DP_PU1_SNOOPS	PU0 L2: Other DP PU1 snoops
0	92	BGP_PU0_L2_RESERVED_1	PU0 L2: Reserved
0	93	BGP_PU0_L2_RESERVED_2	PU0 L2: Reserved
0	94	BGP_PU0_L2_RESERVED_3	PU0 L2: Reserved
0	95	BGP_PU0_L2_RESERVED_4	PU0 L2: Reserved
0	96	BGP_PU0_L2_RESERVED_5	PU0 L2: Reserved
0	97	BGP_PU0_L2_RESERVED_6	PU0 L2: Reserved
0	98	BGP_PU0_L2_RESERVED_7	PU0 L2: Reserved
0	99	BGP_PU0_L2_RESERVED_8	PU0 L2: Reserved
0	100	BGP_PU0_L2_RESERVED_9	PU0 L2: Reserved
0	101	BGP_PU0_L2_RESERVED_10	PU0 L2: Number of writes to L3
0	102	BGP_PU0_L2_RESERVED_11	PU0 L2: Number of writes to network
0	103	BGP_PU0_L2_RESERVED_12	PU0 L2: Number of writes to devbus
0	104	BGP_PU1_L2_VALID_PREFETCH_REQUESTS	PU1 L2: Prefetch request valid
0	105	BGP_PU1_L2_PREFETCH_HITS_IN_FILTER	PU1 L2: Prefetch hits in filter
0	106	BGP_PU1_L2_PREFETCH_HITS_IN_STREAM	PU1 L2: Prefetch hits in active stream
0	107	BGP_PU1_L2_CYCLES_PREFETCH_PENDING	PU1 L2: Number of cycles for which L2-prefetch is pending
0	108	BGP_PU1_L2_PAGE_ALREADY_IN_L2	PU1 L2: requested PF is already in L2
0	109	BGP_PU1_L2_PREFETCH_SNOOP_HIT_SAME_CORE	PU1 L2: Prefetch snoop hit from the same core (write)
0	110	BGP_PU1_L2_PREFETCH_SNOOP_HIT_OTHER_CORE	PU1 L2: Prefetch snoop hit from the other core
0	111	BGP_PU1_L2_PREFETCH_SNOOP_HIT_PLB	PU1 L2: Prefetch snoop hit PLB (write)
0	112	BGP_PU1_L2_CYCLES_READ_REQUEST_PENDING	PU1 L2: Number of cycles for which a read request is pending
0	113	BGP_PU1_L2_READ_REQUESTS	PU1 L2: Read requests

Group	Counter	Event name	Event description
0	114	BGP_PU1_L2_DEVBUS_READ_REQUESTS	PU1 L2: Devbus read requests (for SRAM, LOCK and UPC)
0	115	BGP_PU1_L2_L3_READ_REQUESTS	PU1 L2: L3 read request
0	116	BGP_PU1_L2_NETBUS_READ_REQUESTS	PU1 L2: Netbus read requests (for tree and torus)
0	117	BGP_PU1_L2_BLIND_DEV_READ_REQUESTS	PU1 L2: BLIND device read request
0	118	BGP_PU1_L2_PREFETCHABLE_REQUESTS	PU1 L2: Prefetchable requests
0	119	BGP_PU1_L2_HIT	PU1 L2: L2 hit
0	120	BGP_PU1_L2_SAME_CORE_SNOOPS	PU1 L2: Same core snoops
0	121	BGP_PU1_L2_OTHER_CORE_SNOOPS	PU1 L2: Other core snops
0	122	BGP_PU1_L2_OTHER_DP_PU0_SNOOPS	PU1 L2: Other DP PU0 snoops
0	123	BGP_PU1_L2_OTHER_DP_PU1_SNOOPS	PU1 L2: Other DP PU1 snoops
0	124	BGP_PU1_L2_RESERVED_1	PU1 L2: Reserved
0	125	BGP_PU1_L2_RESERVED_2	PU1 L2: Reserved
0	126	BGP_PU1_L2_RESERVED_3	PU1 L2: Reserved
0	127	BGP_PU1_L2_RESERVED_4	PU1 L2: Reserved
0	128	BGP_PU1_L2_RESERVED_5	PU1 L2: Reserved
0	129	BGP_PU1_L2_RESERVED_6	PU1 L2: Reserved
0	130	BGP_PU1_L2_RESERVED_7	PU1 L2: Reserved
0	131	BGP_PU1_L2_RESERVED_8	PU1 L2: Reserved
0	132	BGP_PU1_L2_RESERVED_9	PU1 L2: Reserved
0	133	BGP_PU1_L2_RESERVED_10	PU1 L2: Number of writes to L3
0	134	BGP_PU1_L2_RESERVED_11	PU1 L2: Number of writes to network
0	135	BGP_PU1_L2_RESERVED_12	PU1 L2: Number of writes to devbus
0	136	BGP_L3_M0_RD0_SINGLE_LINE_DELIVERED_L2	L3 M0: Rd 0: Single line delivered to L2
0	137	BGP_L3_M0_RD0_BURST_DELIVERED_L2	L3 M0: Rd 0: Burst delivered to L2
0	138	BGP_L3_M0_RD0_READ_RETURN_COLLISION	L3 M0: Rd 0: Read return collision
0	139	BGP_L3_M0_RD0_DIR0_HIT_OR_INFLIGHT	L3 M0: Rd 0: dir0 hit or in flight
0	140	BGP_L3_M0_RD0_DIR0_MISS_OR_LOCKDOWN	L3 M0: Rd 0: dir0 miss or lock-down
0	141	BGP_L3_M0_RD0_DIR1_HIT_OR_INFLIGHT	L3 M0: Rd 0: dir1 hit or in flight
0	142	BGP_L3_M0_RD0_DIR1_MISS_OR_LOCKDOWN	L3 M0: Rd 0: dir1 miss or lock-down
0	143	BGP_L3_M0_RD1_SINGLE_LINE_DELIVERED_L2	L3 M0: Rd 1: Single line delivered to L2
0	144	BGP_L3_M0_RD1_BURST_DELIVERED_L2	L3 M0: Rd 1: Burst delivered to L2
0	145	BGP_L3_M0_RD1_READ_RETURN_COLLISION	L3 M0: Rd 1: Read return collision

Group	Counter	Event name	Event description
0	146	BGP_L3_M0_RD1_DIR0_HIT_OR_INFLIGHT	L3 M0: Rd 1: dir0 hit or in flight
0	147	BGP_L3_M0_RD1_DIR0_MISS_OR_LOCKDOWN	L3 M0: Rd 1: dir0 miss or lock-down
0	148	BGP_L3_M0_RD1_DIR1_HIT_OR_INFLIGHT	L3 M0: Rd 1: dir1 hit or in flight
0	149	BGP_L3_M0_RD1_DIR1_MISS_OR_LOCKDOWN	L3 M0: Rd 1: dir1 miss or lock-down
0	150	BGP_L3_M0_DIR0_LOOKUPS	L3 M0: Dir 0: Number of lookups
0	151	BGP_L3_M0_DIR0_CYCLES_REQUESTS_NOT_TAKEN	L3 M0: Dir 0: Number of cycles with requests that are not taken
0	152	BGP_L3_M0_DIR1_LOOKUPS	L3 M0: Dir 1: Number of lookups
0	153	BGP_L3_M0_DIR1_CYCLES_REQUESTS_NOT_TAKEN	L3 M0: Dir 1: Number of cycles with requests that are not taken
0	154	BGP_L3_M0_MH_DDR_STORES	L3 M0: M0-18/MH: Number of stores to DDR
0	155	BGP_L3_M0_MH_DDR_FETCHES	L3 M0: M0-19/MH: Number of fetches from DDR
0	156	BGP_L3_M1_RD0_SINGLE_LINE_DELIVERED_L2	L3 M1: Rd 0: Single line delivered to L2
0	157	BGP_L3_M1_RD0_BURST_DELIVERED_L2	L3 M1: Rd 0: Burst delivered to L2
0	158	BGP_L3_M1_RD0_READ_RETURN_COLLISION	L3 M1: Rd 0: Read return collision
0	159	BGP_L3_M1_RD0_DIR0_HIT_OR_INFLIGHT	L3 M1: Rd 0: dir0 hit or in flight
0	160	BGP_L3_M1_RD0_DIR0_MISS_OR_LOCKDOWN	L3 M1: Rd 0: dir0 miss or lock-down
0	161	BGP_L3_M1_RD0_DIR1_HIT_OR_INFLIGHT	L3 M1: Rd 0: dir1 hit or in flight
0	162	BGP_L3_M1_RD0_DIR1_MISS_OR_LOCKDOWN	L3 M1: Rd 0: dir1 miss or lock-down
0	163	BGP_L3_M1_RD1_SINGLE_LINE_DELIVERED_L2	L3 M1: Rd 1: Single line delivered to L2
0	164	BGP_L3_M1_RD1_BURST_DELIVERED_L2	L3 M1: Rd 1: Burst delivered to L2
0	165	BGP_L3_M1_RD1_READ_RETURN_COLLISION	L3 M1: Rd 1: Read return collision
0	166	BGP_L3_M1_RD1_DIR0_HIT_OR_INFLIGHT	L3 M1: Rd 1: dir0 hit or in flight
0	167	BGP_L3_M1_RD1_DIR0_MISS_OR_LOCKDOWN	L3 M1: Rd 1: dir0 miss or lock-down
0	168	BGP_L3_M1_RD1_DIR1_HIT_OR_INFLIGHT	L3 M1: Rd 1: dir1 hit or in flight
0	169	BGP_L3_M1_RD1_DIR1_MISS_OR_LOCKDOWN	L3 M1: Rd 1: dir1 miss or lock-down
0	170	BGP_L3_M1_DIR0_LOOKUPS	L3 M1: Dir 0: Number of lookups
0	171	BGP_L3_M1_DIR0_CYCLES_REQUESTS_NOT_TAKEN	L3 M1: Dir 0: Number of cycles with requests that are not taken
0	172	BGP_L3_M1_DIR1_LOOKUPS	L3 M1: Dir 1: Number of lookups
0	173	BGP_L3_M1_DIR1_CYCLES_REQUESTS_NOT_TAKEN	L3 M1: Dir 1: Number of cycles with requests that are not taken
0	174	BGP_L3_M1_MH_DDR_STORES	L3 M1: M0-18/MH: Number of stores to DDR

Group	Counter	Event name	Event description
0	175	BGP_L3_M1_MH_DDR_FETCHES	L3 M1: M0-19/MH: Number of fetches from DDR
0	176	BGP_PU0_SNOOP_PORT0_REMOTE_SOURCE_REQUESTS	PU0 snoop: Port 0 received a snoop request from a remote source
0	177	BGP_PU0_SNOOP_PORT1_REMOTE_SOURCE_REQUESTS	PU0 snoop: Port 1 received a snoop request from a remote source
0	178	BGP_PU0_SNOOP_PORT2_REMOTE_SOURCE_REQUESTS	PU0 snoop: Port 2 received a snoop request from a remote source
0	179	BGP_PU0_SNOOP_PORT3_REMOTE_SOURCE_REQUESTS	PU0 snoop: Port 3 received a snoop request from a remote source
0	180	BGP_PU0_SNOOP_PORT0_REJECTED_REQUESTS	PU0 snoop: Port 0 snoop filter rejected a snoop request
0	181	BGP_PU0_SNOOP_PORT1_REJECTED_REQUESTS	PU0 snoop: Port 1 snoop filter rejected a snoop request
0	182	BGP_PU0_SNOOP_PORT2_REJECTED_REQUESTS	PU0 snoop: Port 2 snoop filter rejected a snoop request
0	183	BGP_PU0_SNOOP_PORT3_REJECTED_REQUESTS	PU0 snoop: Port 3 snoop filter rejected a snoop request
0	184	BGP_PU0_SNOOP_L1_CACHE_WRAP	PU0 snoop: Snoop filter detected an L1 cache wrap
0	185	BGP_PU1_SNOOP_PORT0_REMOTE_SOURCE_REQUESTS	PU1 snoop: Port 0 received a snoop request from a remote source
0	186	BGP_PU1_SNOOP_PORT1_REMOTE_SOURCE_REQUESTS	PU1 snoop: Port 1 received a snoop request from a remote source
0	187	BGP_PU1_SNOOP_PORT2_REMOTE_SOURCE_REQUESTS	PU1 snoop: Port 2 received a snoop request from a remote source
0	188	BGP_PU1_SNOOP_PORT3_REMOTE_SOURCE_REQUESTS	PU1 snoop: Port 3 received a snoop request from a remote source
0	189	BGP_PU1_SNOOP_PORT0_REJECTED_REQUESTS	PU1 snoop: Port 0 snoop filter rejected a snoop request
0	190	BGP_PU1_SNOOP_PORT1_REJECTED_REQUESTS	PU1 snoop: Port 1 snoop filter rejected a snoop request
0	191	BGP_PU1_SNOOP_PORT2_REJECTED_REQUESTS	PU1 snoop: Port 2 snoop filter rejected a snoop request
0	192	BGP_PU1_SNOOP_PORT3_REJECTED_REQUESTS	PU1 snoop: Port 3 snoop filter rejected a snoop request
0	193	BGP_PU1_SNOOP_L1_CACHE_WRAP	PU1 snoop: Snoop filter detected an L1 cache wrap
0	194	BGP_TORUS_XP_PACKETS	Torus: Number of packets sent to X+ dimension
0	195	BGP_TORUS_XP_32BCHUNKS	Torus: Number of 32B chunks sent to X+

Group	Counter	Event name	Event description
0	196	BGP_TORUS_XM_PACKETS	Torus: Number of packets sent to X-dimension
0	197	BGP_TORUS_XM_32BCHUNKS	Torus: Number of 32B chunks sent to X-
0	198	BGP_TORUS_YP_PACKETS	Torus: Number of packets sent to Y+ dimension
0	199	BGP_TORUS_YP_32BCHUNKS	Torus: Number of 32B chunks sent to Y+
0	200	BGP_TORUS_YM_PACKETS	Torus: Number of packets sent to Y-dimension
0	201	BGP_TORUS_YM_32BCHUNKS	Torus: Number of 32B chunks sent to Y-
0	202	BGP_TORUS_ZP_PACKETS	Torus: Number of packets sent to Z+ dimension
0	203	BGP_TORUS_ZP_32BCHUNKS	Torus: Number of 32B chunks sent to Z+
0	204	BGP_TORUS_ZM_PACKETS	Torus: Number of packets sent to Z-dimension
0	205	BGP_TORUS_ZM_32BCHUNKS	Torus: Number of 32B chunks sent to Z-
0	206	BGP_DMA_PACKETS_INJECTED	DMA: Number of packets injected
0	207	BGP_DMA_DESCRIPTOR_READ_FROM_L3	DMA: Number of descriptors read from L3
0	208	BGP_DMA_FIFO_PACKETS_RECEIVED	DMA: Number of fifo packets received
0	209	BGP_DMA_COUNTER_PACKETS_RECEIVED	DMA: Number of counter packets received
0	210	BGP_DMA_REMOTE_GET_PACKETS_RECEIVED	DMA: Number of remote get packets received
0	211	BGP_DMA_IDPU_READ_REQUESTS_TO_L3	DMA: Number of read requests to L3 by IDPU
0	212	BGP_DMA_READ_VALID_RETURNED	DMA: Number of read valid returned from L3
0	213	BGP_DMA_ACKED_READ_REQUESTS	DMA: Number of DMA L3 read requests acknowledged by the L3
0	214	BGP_DMA_CYCLES_RDPU_WRITE_ACTIVE	DMA: Number of cycles rdpu wants to write to L3, independent of the write ready
0	215	BGP_DMA_WRITE_REQUESTS_TO_L3	DMA: Number of write requests to L3
0	216	BGP_DMA_RESERVED_1	DMA: Reserved
0	217	BGP_DMA_RESERVED_2	DMA: Reserved
0	218	BGP_DMA_RESERVED_3	DMA: Reserved
0	219	BGP_DMA_RESERVED_4	DMA: Reserved
0	220	BGP_DMA_RESERVED_5	DMA: Reserved
0	221	BGP_DMA_RESERVED_6	DMA: Reserved

Group	Counter	Event name	Event description
0	222	BGP_COL_AC_CH2_VC0_MATURE	Collective: arbiter_core ch2_vc0_mature
0	223	BGP_COL_AC_CH1_VC0_MATURE	Collective: arbiter_core ch1_vc0_mature
0	224	BGP_COL_AC_CH0_VC0_MATURE	Collective: arbiter_core ch0_vc0_mature
0	225	BGP_COL_AC_INJECT_VC0_MATURE	Collective: arbiter_core inj_vc0_mature
0	226	BGP_COL_AC_CH2_VC1_MATURE	Collective: arbiter_core ch2_vc1_mature
0	227	BGP_COL_AC_CH1_VC1_MATURE	Collective: arbiter_core ch1_vc1_mature
0	228	BGP_COL_AC_CH0_VC1_MATURE	Collective: arbiter_core ch0_vc1_mature
0	229	BGP_COL_AC_INJECT_VC1_MATURE	Collective: arbiter_core inj_vc1_mature
0	230	BGP_COL_AC_PENDING_REQUESTS	Collective: arbiter_core requests pending
0	231	BGP_COL_AC_WAITING_REQUESTS	Collective: arbiter_core requests waiting (ready to go)
0	232	BGP_COL_AR2_PACKET_TAKEN	Collective: Arbiter receiver 2 packets taken
0	233	BGP_COL_AR1_PACKET_TAKEN	Collective: Arbiter receiver 1 packets taken
0	234	BGP_COL_AR0_PACKET_TAKEN	Collective: Arbiter receiver 0 packets taken
0	235	BGP_COL_ALC_PACKET_TAKEN	Collective: Arbiter local client packets taken
0	236	BGP_COL_AR0_VC0_DATA_PACKETS_RECEIVED	Collective: Receiver 0 vc0 data packets received
0	237	BGP_COL_AR0_VC1_DATA_PACKETS_RECEIVED	Collective: Receiver 0 vc1 data packets received
0	238	BGP_COL_AR1_VC0_DATA_PACKETS_RECEIVED	Collective: Receiver 1 vc0 data packets received
0	239	BGP_COL_AR1_VC1_DATA_PACKETS_RECEIVED	Collective: Receiver 1 vc1 data packets received
0	240	BGP_COL_AR2_VC0_DATA_PACKETS_RECEIVED	Collective: Receiver 2 vc0 data packets received
0	241	BGP_COL_AR2_VC1_DATA_PACKETS_RECEIVED	Collective: Receiver 2 vc1 data packets received
0	242	BGP_COL_AS0_VC0_DATA_PACKETS_SENT	Collective: Sender 0 vc0 data packets sent
0	243	BGP_COL_AS0_VC1_DATA_PACKETS_SENT	Collective: Sender 0 vc1 data packets sent

Group	Counter	Event name	Event description
0	244	BGP_COL_AS1_VC0_DATA_PACKETS_SENT	Collective: Sender 1 vc0 data packets sent
0	245	BGP_COL_AS1_VC1_DATA_PACKETS_SENT	Collective: Sender 1 vc1 data packets sent
0	246	BGP_COL_AS2_VC0_DATA_PACKETS_SENT	Collective: Sender 2 vc0 data packets sent
0	247	BGP_COL_AS2_VC1_DATA_PACKETS_SENT	Collective: Sender 2 vc1 data packets sent
0	248	BGP_COL_INJECT_VC0_HEADER	Collective: Injection vc0 header
0	249	BGP_COL_INJECT_VC1_HEADER	Collective: Injection vc1 header added
0	250	BGP_COL_RECEPTION_VC0_PACKET_ADDED	Collective: Reception vc0 packet added
0	251	BGP_COL_RECEPTION_VC1_PACKET_ADDED	Collective: Reception vc1 packet added
0	252	BGP_IC_TIMESTAMP	IC: Timestamp
0	253	BGP_MISC_RESERVED_1	Misc: Reserved
0	254	BGP_MISC_RESERVED_2	Misc: Reserved
0	255	BGP_MISC_ELAPSED_TIME	Misc: Elapsed time
1	0	BGP_PU2_JPIPE_INSTRUCTIONS	PU2: Number of J-pipe instructions
1	1	BGP_PU2_JPIPE_ADD_SUB	PU2: PowerPC Add/Sub in J-pipe
1	2	BGP_PU2_JPIPE_LOGICAL_OPS	PU2: PowerPC logical operations in J-pipe
1	3	BGP_PU2_JPIPE_SHROTMK	PU2: Shift, rotate, mask instructions
1	4	BGP_PU2_IPIPE_INSTRUCTIONS	PU2: Number of I-pipe instructions
1	5	BGP_PU2_IPIPE_MULT_DIV	PU2: PowerPC Mul/Div in I-pipe
1	6	BGP_PU2_IPIPE_ADD_SUB	PU2: PowerPC Add/Sub in I-pipe
1	7	BGP_PU2_IPIPE_LOGICAL_OPS	PU2: PowerPC logical operations in I-pipe
1	8	BGP_PU2_IPIPE_SHROTMK	PU2: Shift, rotate, mask instructions
1	9	BGP_PU2_IPIPE_BRANCHES	PU2: PowerPC branches
1	10	BGP_PU2_IPIPE_TLB_OPS	PU2: PowerPC TLB operations
1	11	BGP_PU2_IPIPE_PROCESS_CONTROL	PU2: PowerPC process control
1	12	BGP_PU2_IPIPE_OTHER	PU2: PowerPC other I-pipe operations
1	13	BGP_PU2_DCACHE_LINEFILLINPROG	PU2: Number of cycles D-cache LineFillInProgress
1	14	BGP_PU2_ICACHE_LINEFILLINPROG	PU2: Number of cycles I-cache LineFillInProgress
1	15	BGP_PU2_DCACHE_MISS	PU2: Accesses to D cache that miss in D Cache





Group	Counter	Event name	Event description
1	33	BGP_PU2_FPU_QUADWORD_STORES	PU2: Quad Word Stores (stfpdx, stfpdux, stfxdx, stfxdux)
1	34	BGP_PU2_FPU_OTHER_STORES	PU2: All other FPU stores (stfs, stfsx, stfsu, stfsux, stfd, stfdx, stfdu, stfdx, stfiwx, stfpsx, stfpsux, stfpiwx, stfsdx, stfsdux, stfssx, stfssux, stfxsx, stfxsux)
1	35	BGP_PU3_JPIPE_INSTRUCTIONS	PU3: Number of J-pipe instructions
1	36	BGP_PU3_JPIPE_ADD_SUB	PU3: PowerPC Add/Sub in J-pipe
1	37	BGP_PU3_JPIPE_LOGICAL_OPS	PU3: PowerPC logical operations in J-pipe
1	38	BGP_PU3_JPIPE_SHROTMK	PU3: Shift, rotate, mask instructions
1	39	BGP_PU3_IPIPE_INSTRUCTIONS	PU3: Number of I-pipe instructions
1	40	BGP_PU3_IPIPE_MULT_DIV	PU3: PowerPC Mul/Div in I-pipe
1	41	BGP_PU3_IPIPE_ADD_SUB	PU3: PowerPC Add/Sub in I-pipe
1	42	BGP_PU3_IPIPE_LOGICAL_OPS	PU3: PowerPC logical operations in I-pipe
1	43	BGP_PU3_IPIPE_SHROTMK	PU3: Shift, rotate, mask instructions
1	44	BGP_PU3_IPIPE_BRANCHES	PU3: PowerPC branches
1	45	BGP_PU3_IPIPE_TLB_OPS	PU3: PowerPC TLB operations
1	46	BGP_PU3_IPIPE_PROCESS_CONTROL	PU3: PowerPC process control
1	47	BGP_PU3_IPIPE_OTHER	PU3: PowerPC other I-pipe operations
1	48	BGP_PU3_DCACHE_LINEFILLINPROG	PU3: Number of cycles D-cache LineFillInProgress
1	49	BGP_PU3_ICACHE_LINEFILLINPROG	PU3: Number of cycles I-cache LineFillInProgress
1	50	BGP_PU3_DCACHE_MISS	PU3: Accesses to D cache that miss in D Cache
1	51	BGP_PU3_DCACHE_HIT	PU3: Accesses to D cache that hit in D Cache
1	52	BGP_PU3_DATA_LOADS	PU3: PowerPC data loads
1	53	BGP_PU3_DATA_STORES	PU3: PowerPC data stores
1	54	BGP_PU3_DCACHE_OPS	PU3: D cache operations
1	55	BGP_PU3_ICACHE_MISS	PU3: Accesses to I cache that miss in I Cache
1	56	BGP_PU3_ICACHE_HIT	PU3: Accesses to I cache that hit in I Cache
1	57	BGP_PU3_FPU_ADD_SUB_1	PU3: PowerPC FP Add/Sub (fadd, fadds, fsub, fsubs)
1	58	BGP_PU3_FPU_MULT_1	PU3: PowerPC FP Mult (fmul, fmuls)



Group	Counter	Event name	Event description
1	75	BGP_COL_AC_INJECT_VC0_MATURE_UM1	Collective: arbiter_core inj_vc0_mature
1	76	BGP_COL_AC_CH2_VC1_MATURE_UM1	Collective: arbiter_core ch2_vc1_mature
1	77	BGP_COL_AC_CH1_VC1_MATURE_UM1	Collective: arbiter_core ch1_vc1_mature
1	78	BGP_COL_AC_CH0_VC1_MATURE_UM1	Collective: arbiter_core ch0_vc1_mature
1	79	BGP_COL_AC_INJECT_VC1_MATURE_UM1	Collective: arbiter_core inj_vc1_mature
1	80	BGP_COL_AR0_VC0_EMPTY_PACKET	Collective: Receiver 0 vc0 empty packet
1	81	BGP_COL_AR0_VC1_EMPTY_PACKET	Collective: Receiver 0 vc1 empty packet
1	82	BGP_COL_AR0_IDLE_PACKET	Collective: Receiver 0 IDLE packet
1	83	BGP_COL_AR0_BAD_PACKET_MARKER	Collective: Receiver 0 known-bad-packet marker
1	84	BGP_COL_AR0_VC0_CUT_THROUGH	Collective: Receiver 0 vc0 cut-through
1	85	BGP_COL_AR0_VC1_CUT_THROUGH	Collective: Receiver 0 vc1 cut-through
1	86	BGP_COL_AR0_HEADER_PARITY_ERROR	Collective: Receiver 0 header parity error
1	87	BGP_COL_AR0_UNEXPECTED_HEADER_ERROR	Collective: Receiver 0 unexpected header error
1	88	BGP_COL_AR0_RESYNC	Collective: Receiver 0 resynch-mode (after error)
1	89	BGP_COL_AR1_VC0_EMPTY_PACKET	Collective: Receiver 1 vc0 empty packet
1	90	BGP_COL_AR1_VC1_EMPTY_PACKET	Collective: Receiver 1 vc1 empty packet
1	91	BGP_COL_AR1_IDLE_PACKET	Collective: Receiver 1 IDLE packet
1	92	BGP_COL_AR1_BAD_PACKET_MARKER	Collective: Receiver 1 known-bad-packet marker
1	93	BGP_COL_AR1_VC0_CUT_THROUGH	Collective: Receiver 1 vc0 cut-through
1	94	BGP_COL_AR1_VC1_CUT_THROUGH	Collective: Receiver 1 vc1 cut-through
1	95	BGP_COL_AR1_HEADER_PARITY_ERROR	Collective: Receiver 1 header parity error
1	96	BGP_COL_AR1_UNEXPECTED_HEADER_ERROR	Collective: Receiver 1 unexpected header error
1	97	BGP_COL_AR1_RESYNC	Collective: Receiver 1 resynch-mode (after error)
1	98	BGP_COL_AR2_VC0_EMPTY_PACKET	Collective: Receiver 2 vc0 empty packet
1	99	BGP_COL_AR2_VC1_EMPTY_PACKET	Collective: Receiver 2 vc1 empty packet
1	100	BGP_COL_AR2_IDLE_PACKET	Collective: Receiver 2 IDLE packet
1	101	BGP_COL_AR2_BAD_PACKET_MARKER	Collective: Receiver 2 known-bad-packet marker

Group	Counter	Event name	Event description
1	102	BGP_COL_AR2_VC0_CUT_THROUGH	Collective: Receiver 2 vc0 cut-through
1	103	BGP_COL_AR2_VC1_CUT_THROUGH	Collective: Receiver 2 vc1 cut-through
1	104	BGP_COL_AR2_HEADER_PARITY_ERROR	Collective: Receiver 2 header parity error
1	105	BGP_COL_AR2_UNEXPECTED_HEADER_ERROR	Collective: Receiver 2 unexpected header error
1	106	BGP_COL_AR2_RESYNC	Collective: Receiver 2 resynch-mode (after error)
1	107	BGP_COL_AS0_VC0_CUT_THROUGH	Collective: Sender 0 vc0 cut-through
1	108	BGP_COL_AS0_VC1_CUT_THROUGH	Collective: Sender 0 vc1 cut-through
1	109	BGP_COL_AS0_VC0_PACKETS_SENT	Collective: Sender 0 vc0 packet sent (total)
1	110	BGP_COL_AS0_VC1_PACKETS_SENT	Collective: Sender 0 vc1 packet sent (total)
1	111	BGP_COL_AS0_IDLE_PACKETS_SENT	Collective: Sender 0 IDLE packets sent
1	112	BGP_COL_AS1_VC0_CUT_THROUGH	Collective: Sender 1 vc0 cut-through
1	113	BGP_COL_AS1_VC1_CUT_THROUGH	Collective: Sender 1 vc1 cut-through
1	114	BGP_COL_AS1_VC0_PACKETS_SENT	Collective: Sender 1 vc0 packets sent (total)
1	115	BGP_COL_AS1_VC1_PACKETS_SENT	Collective: Sender 1 vc1 packets sent (total)
1	116	BGP_COL_AS1_IDLE_PACKETS_SENT	Collective: Sender 1 IDLE packets sent
1	117	BGP_COL_AS2_VC0_CUT_THROUGH	Collective: Sender 2 vc0 cut-through
1	118	BGP_COL_AS2_VC1_CUT_THROUGH	Collective: Sender 2 vc1 cut-through
1	119	BGP_COL_AS2_VC0_PACKETS_SENT	Collective: Sender 2 vc0 packets sent (total)
1	120	BGP_COL_AS2_VC1_PACKETS_SENT	Collective: Sender 2 vc1 packets sent (total)
1	121	BGP_COL_AS2_IDLE_PACKETS_SENT	Collective: Sender 2 IDLE packets sent
1	122	BGP_COL_INJECT_VC0_PAYLOAD_ADDED	Collective: Injection vc0 payload added
1	123	BGP_COL_INJECT_VC1_PAYLOAD_ADDED	Collective: Injection vc1 payload added
1	124	BGP_COL_INJECT_VC0_PACKET_TAKEN	Collective: Injection vc0 packet taken
1	125	BGP_COL_INJECT_VC1_PACKET_TAKEN	Collective: Injection vc1 packet taken
1	126	BGP_COL_RECEPTION_VC0_HEADER_TAKEN	Collective: Reception vc0 header taken
1	127	BGP_COL_RECEPTION_VC1_HEADER_TAKEN	Collective: Reception vc1 header taken
1	128	BGP_COL_RECEPTION_VC0_PAYLOAD_TAKEN	Collective: Reception vc0 payload taken
1	129	BGP_COL_RECEPTION_VC1_PAYLOAD_TAKEN	Collective: Reception vc1 payload taken

Group	Counter	Event name	Event description
1	130	BGP_COL_RECEPTION_VC0_PACKET_DISCARDED	Collective: Reception vc0 packet discarded
1	131	BGP_COL_RECEPTION_VC1_PACKET_DISCARDED	Collective: Reception vc1 packet discarded
1	132	BGP_PU2_L2_VALID_PREFETCH_REQUESTS	PU2 L2: Prefetch request valid
1	133	BGP_PU2_L2_PREFETCH_HITS_IN_FILTER	PU2 L2: Prefetch hits in filter
1	134	BGP_PU2_L2_PREFETCH_HITS_IN_STREAM	PU2 L2: Prefetch hits in active stream
1	135	BGP_PU2_L2_CYCLES_PREFETCH_PENDING	PU2 L2: Number of cycles for which an L2-prefetch is pending
1	136	BGP_PU2_L2_PAGE_ALREADY_IN_L2	PU2 L2: Requested PF is already in L2
1	137	BGP_PU2_L2_PREFETCH_SNOOP_HIT_SAME_CORE	PU2 L2: Prefetch snoop hit from same core (write)
1	138	BGP_PU2_L2_PREFETCH_SNOOP_HIT_OTHER_CORE	PU2 L2: Prefetch snoop hit from other core
1	139	BGP_PU2_L2_PREFETCH_SNOOP_HIT_PLB	PU2 L2: Prefetch snoop hit PLB (write)
1	140	BGP_PU2_L2_CYCLES_READ_REQUEST_PENDING	PU2 L2: Number of cycles for which a read request is pending
1	141	BGP_PU2_L2_READ_REQUESTS	PU2 L2: read requests
1	142	BGP_PU2_L2_DEVBUS_READ_REQUESTS	PU2 L2: devbus read requests (for SRAM, LOCK and UPC)
1	143	BGP_PU2_L2_L3_READ_REQUESTS	PU2 L2: L3 read request
1	144	BGP_PU2_L2_NETBUS_READ_REQUESTS	PU2 L2: netbus read requests (for tree and torus)
1	145	BGP_PU2_L2_BLIND_DEV_READ_REQUESTS	PU2 L2: BLIND device read request
1	146	BGP_PU2_L2_PREFETCHABLE_REQUESTS	PU2 L2: Prefetchable requests
1	147	BGP_PU2_L2_HIT	PU2 L2: L2 hit
1	148	BGP_PU2_L2_SAME_CORE_SNOOPS	PU2 L2: Same core snoops
1	149	BGP_PU2_L2_OTHER_CORE_SNOOPS	PU2 L2: Other core snoops
1	150	BGP_PU2_L2_OTHER_DP_PU0_SNOOPS	PU2 L2: Other DP PU2 snoops
1	151	BGP_PU2_L2_OTHER_DP_PU1_SNOOPS	PU2 L2: Other DP PU1 snoops
1	152	BGP_PU2_L2_RESERVED_1	PU2 L2: Reserved
1	153	BGP_PU2_L2_RESERVED_2	PU2 L2: Reserved
1	154	BGP_PU2_L2_RESERVED_3	PU2 L2: Reserved
1	155	BGP_PU2_L2_RESERVED_4	PU2 L2: Reserved
1	156	BGP_PU2_L2_RESERVED_5	PU2 L2: Reserved
1	157	BGP_PU2_L2_RESERVED_6	PU2 L2: Reserved
1	158	BGP_PU2_L2_RESERVED_7	PU2 L2: Reserved

Group	Counter	Event name	Event description
1	159	BGP_PU2_L2_RESERVED_8	PU2 L2: Reserved
1	160	BGP_PU2_L2_RESERVED_9	PU2 L2: Reserved
1	161	BGP_PU2_L2_RESERVED_10	PU2 L2: Number of writes to L3
1	162	BGP_PU2_L2_RESERVED_11	PU2 L2: Number of writes to network
1	163	BGP_PU2_L2_RESERVED_12	PU2 L2: Number of writes to devbus
1	164	BGP_PU3_L2_VALID_PREFETCH_REQUESTS	PU3 L2: Prefetch request valid
1	165	BGP_PU3_L2_PREFETCH_HITS_IN_FILTER	PU3 L2: Prefetch hits in filter
1	166	BGP_PU3_L2_PREFETCH_HITS_IN_STREAM	PU3 L2: Prefetch hits in active stream
1	167	BGP_PU3_L2_CYCLES_PREFETCH_PENDING	PU3 L2: Number of cycles for which an L2-prefetch is pending
1	168	BGP_PU3_L2_PAGE_ALREADY_IN_L2	PU3 L2: requested PF is already in L2
1	169	BGP_PU3_L2_PREFETCH_SNOOP_HIT_SAME_CORE	PU3 L2: Prefetch snoop hit from same core (write)
1	170	BGP_PU3_L2_PREFETCH_SNOOP_HIT_OTHER_CORE	PU3 L2: Prefetch snoop hit from other core
1	171	BGP_PU3_L2_PREFETCH_SNOOP_HIT_PLB	PU3 L2: Prefetch snoop hit PLB (write)
1	172	BGP_PU3_L2_CYCLES_READ_REQUEST_PENDING	PU3 L2: Number of cycles for which a read request is pending
1	173	BGP_PU3_L2_READ_REQUESTS	PU3 L2: read requests
1	174	BGP_PU3_L2_DEVBUS_READ_REQUESTS	PU3 L2: Devbus read requests (for SRAM, LOCK and UPC)
1	175	BGP_PU3_L2_L3_READ_REQUESTS	PU3 L2: L3 read request
1	176	BGP_PU3_L2_NETBUS_READ_REQUESTS	PU3 L2: netbus read requests (for tree and torus)
1	177	BGP_PU3_L2_BLIND_DEV_READ_REQUESTS	PU3 L2: BLIND device read request
1	178	BGP_PU3_L2_PREFETCHABLE_REQUESTS	PU3 L2: Prefetchable requests
1	179	BGP_PU3_L2_HIT	PU3 L2: L2 hit
1	180	BGP_PU3_L2_SAME_CORE_SNOOPS	PU3 L2: Same core snoops
1	181	BGP_PU3_L2_OTHER_CORE_SNOOPS	PU3 L2: Other core snops
1	182	BGP_PU3_L2_OTHER_DP_PU0_SNOOPS	PU3 L2: Other DP PU0 snoops
1	183	BGP_PU3_L2_OTHER_DP_PU1_SNOOPS	PU3 L2: Other DP PU3 snoops
1	184	BGP_PU3_L2_RESERVED_1	PU3 L2: Reserved
1	185	BGP_PU3_L2_RESERVED_2	PU3 L2: Reserved
1	186	BGP_PU3_L2_RESERVED_3	PU3 L2: Reserved
1	187	BGP_PU3_L2_RESERVED_4	PU3 L2: Reserved
1	188	BGP_PU3_L2_RESERVED_5	PU3 L2: Reserved

Group	Counter	Event name	Event description
1	189	BGP_PU3_L2_RESERVED_6	PU3 L2: Reserved
1	190	BGP_PU3_L2_RESERVED_7	PU3 L2: Reserved
1	191	BGP_PU3_L2_RESERVED_8	PU3 L2: Reserved
1	192	BGP_PU3_L2_RESERVED_9	PU3 L2: Reserved
1	193	BGP_PU3_L2_RESERVED_10	PU3 L2: Number of writes to L3
1	194	BGP_PU3_L2_RESERVED_11	PU3 L2: Number of writes to network
1	195	BGP_PU3_L2_RESERVED_12	PU3 L2: Number of writes to devbus
1	196	BGP_L3_M0_R2_SINGLE_LINE_DELIVERED_L2	L3 M0: Rd 2: Single line delivered to L2
1	197	BGP_L3_M0_R2_BURST_DELIVERED_L2	L3 M0: Rd 2: Burst delivered to L2
1	198	BGP_L3_M0_R2_READ_RETURN_COLLISION	L3 M0: Rd 2: Read return collision
1	199	BGP_L3_M0_R2_DIR0_HIT_OR_INFLIGHT	L3 M0: Rd 2: dir0 hit or in flight
1	200	BGP_L3_M0_R2_DIR0_MISS_OR_LOCKDOWN	L3 M0: Rd 2: dir0 miss or lock-down
1	201	BGP_L3_M0_R2_DIR1_HIT_OR_INFLIGHT	L3 M0: Rd 2: dir1 hit or in flight
1	202	BGP_L3_M0_R2_DIR1_MISS_OR_LOCKDOWN	L3 M0: Rd 2: dir1 miss or lock-down
1	203	BGP_L3_M0_W0_DEPOSIT_REQUESTS	L3 M0: WRB 0: total accepted deposit requests from write queues to write buffer
1	204	BGP_L3_M0_W0_CYCLES_REQUESTS_NOT_TAKEN	L3 M0: WRB 0: Number of cycles with requests from queues that are not taken
1	205	BGP_L3_M0_W1_DEPOSIT_REQUESTS	L3 M0: WRB 1: Total accepted deposit requests from write queues to write buffer
1	206	BGP_L3_M0_W1_CYCLES_REQUESTS_NOT_TAKEN	L3 M0: WRB 1: Number of cycles with requests from queues that are not taken
1	207	BGP_L3_M0_MH_ALLOCATION_REQUESTS	L3 M0: MH: Number of allocation requests to write buffer
1	208	BGP_L3_M0_MH_CYCLES_ALLOCATION_REQUESTS_NOT_TAKEN	L3 M0: MH: Number of allocation request cycles to write buffer without being taken
1	209	BGP_L3_M0_PF_PREFETCH_INT0_EDRAM	L3 M0: PF: Number of line prefetches brought into eDRAM
1	210	BGP_L3_M0_RESERVED_1	L3 M0: Reserved
1	211	BGP_L3_M0_RESERVED_2	L3 M0: Reserved
1	212	BGP_L3_M0_RESERVED_3	L3 M0: Reserved
1	213	BGP_L3_M0_RESERVED_4	L3 M0: Reserved
1	214	BGP_L3_M0_RESERVED_5	L3 M0: Reserved
1	215	BGP_L3_M0_RESERVED_6	L3 M0: Reserved
1	216	BGP_L3_M1_R2_SINGLE_LINE_DELIVERED_L2	L3 M1: Rd 2: Single line delivered to L2



Group	Counter	Event name	Event description
1	217	BGP_L3_M1_R2_BURST_DELIVERED_L2	L3 M1: Rd 2: Burst delivered to L2
1	218	BGP_L3_M1_R2_READ_RETURN_COLLISION	L3 M1: Rd 2: Read return collision
1	219	BGP_L3_M1_R2_DIR0_HIT_OR_INFLIGHT	L3 M1: Rd 2: dir0 hit or in flight
1	220	BGP_L3_M1_R2_DIR0_MISS_OR_LOCKDOWN	L3 M1: Rd 2: dir0 miss or lock-down
1	221	BGP_L3_M1_R2_DIR1_HIT_OR_INFLIGHT	L3 M1: Rd 2: dir1 hit or in flight
1	222	BGP_L3_M1_R2_DIR1_MISS_OR_LOCKDOWN	L3 M1: Rd 2: dir1 miss or lock-down
1	223	BGP_L3_M1_W0_DEPOSIT_REQUESTS	L3 M1: WRB 0: Total accepted deposit requests from write queues to write buffer
1	224	BGP_L3_M1_W0_CYCLES_REQUESTS_NOT_TAKEN	L3 M1: WRB 0: Number of cycles with requests from queues that are not taken
1	225	BGP_L3_M1_W1_DEPOSIT_REQUESTS	L3 M1: WRB 1: Total accepted deposit requests from write queues to write buffer
1	226	BGP_L3_M1_W1_CYCLES_REQUESTS_NOT_TAKEN	L3 M1: WRB 1: Number of cycles with requests from queues that are not taken
1	227	BGP_L3_M1_MH_ALLOCATION_REQUESTS	L3 M1: MH: Number of allocation requests to write buffer
1	228	BGP_L3_M1_MH_CYCLES_ALLOCATION_REQUESTS_NOT_TAKEN	L3 M1: MH: Number of allocation request cycles to writebuffer without being taken
1	229	BGP_L3_M1_PF_PREFETCH_INTO_EDRAM	L3 M1: PF: Number of line prefetches brought into eDRAM
1	230	BGP_L3_M1_RESERVED_1	L3 M1: Reserved
1	231	BGP_L3_M1_RESERVED_2	L3 M1: Reserved
1	232	BGP_L3_M1_RESERVED_3	L3 M1: Reserved
1	233	BGP_L3_M1_RESERVED_4	L3 M1: Reserved
1	234	BGP_L3_M1_RESERVED_5	L3 M1: Reserved
1	235	BGP_L3_M1_RESERVED_6	L3 M1: Reserved
1	236	BGP_PU2_SNOOP_PORT0_REMOTE_SOURCE_REQUESTS	PU2 snoop: Port 0 received a snoop request from a remote source
1	237	BGP_PU2_SNOOP_PORT1_REMOTE_SOURCE_REQUESTS	PU2 snoop: Port 1 received a snoop request from a remote source
1	238	BGP_PU2_SNOOP_PORT2_REMOTE_SOURCE_REQUESTS	PU2 snoop: Port 2 received a snoop request from a remote source
1	239	BGP_PU2_SNOOP_PORT3_REMOTE_SOURCE_REQUESTS	PU2 snoop: Port 3 received a snoop request from a remote source
1	240	BGP_PU2_SNOOP_PORT0_REJECTED_REQUESTS	PU2 snoop: Port 0 snoop filter rejected a snoop request

Group	Counter	Event name	Event description
1	241	BGP_PU2_SNOOP_PORT1_REJECTED_REQUESTS	PU2 snoop: Port 1 snoop filter rejected a snoop request
1	242	BGP_PU2_SNOOP_PORT2_REJECTED_REQUESTS	PU2 snoop: Port 2 snoop filter rejected a snoop request
1	243	BGP_PU2_SNOOP_PORT3_REJECTED_REQUESTS	PU2 snoop: Port 3 snoop filter rejected a snoop request
1	244	BGP_PU2_SNOOP_L1_CACHE_WRAP	PU2 snoop: Snoop filter detected an L1 cache wrap
1	245	BGP_PU3_SNOOP_PORT0_REMOTE_SOURCE_REQUESTS	PU3 snoop: Port 0 received a snoop request from a remote source
1	246	BGP_PU3_SNOOP_PORT1_REMOTE_SOURCE_REQUESTS	PU3 snoop: Port 1 received a snoop request from a remote source
1	247	BGP_PU3_SNOOP_PORT2_REMOTE_SOURCE_REQUESTS	PU3 snoop: Port 2 received a snoop request from a remote source
1	248	BGP_PU3_SNOOP_PORT3_REMOTE_SOURCE_REQUESTS	PU3 snoop: Port 3 received a snoop request from a remote source
1	249	BGP_PU3_SNOOP_PORT0_REJECTED_REQUESTS	PU3 snoop: Port 0 snoop filter rejected a snoop request
1	250	BGP_PU3_SNOOP_PORT1_REJECTED_REQUESTS	PU3 snoop: Port 1 snoop filter rejected a snoop request
1	251	BGP_PU3_SNOOP_PORT2_REJECTED_REQUESTS	PU3 snoop: Port 2 snoop filter rejected a snoop request
1	252	BGP_PU3_SNOOP_PORT3_REJECTED_REQUESTS	PU3 snoop: Port 3 snoop filter rejected a snoop request
1	253	BGP_PU3_SNOOP_L1_CACHE_WRAP	PU3 snoop: Snoop filter detected an L1 cache wrap
1	254	BGP_MISC_RESERVED_3	Misc: Reserved
1	255	BGP_MISC_ELAPSED_TIME_UM1	Misc: Elapsed time
2	0	BGP_PU0_JPIPE_INSTRUCTIONS_UM2	PU0: Number of J-pipe instructions
2	1	BGP_PU0_JPIPE_ADD_SUB_UM2	PU0: PowerPC Add/Sub in J-pipe
2	2	BGP_PU0_JPIPE_LOGICAL_OPS_UM2	PU0: PowerPC logical operations in J-pipe
2	3	BGP_PU0_JPIPE_SHROTMK_UM2	PU0: Shift, rotate, mask instructions
2	4	BGP_PU0_IPIPE_INSTRUCTIONS_UM2	PU0: Number of I-pipe instructions
2	5	BGP_PU0_IPIPE_MULT_DIV_UM2	PU0: PowerPC Mul/Div in I-pipe
2	6	BGP_PU0_IPIPE_ADD_SUB_UM2	PU0: PowerPC Add/Sub in I-pipe
2	7	BGP_PU0_IPIPE_LOGICAL_OPS_UM2	PU0: PowerPC logical operations in I-pipe
2	8	BGP_PU0_IPIPE_SHROTMK_UM2	PU0: Shift, rotate, mask instructions
2	9	BGP_PU0_IPIPE_BRANCHES_UM2	PU0: PowerPC branches







Group	Counter	Event name	Event description
2	70	BGP_PU0_L1_INVALIDATION_UM2	PU0 L1: Invalidation requested
2	71	BGP_PU1_L1_INVALIDATION_UM2	PU1 L1: Invalidation requested
2	72	BGP_PU0_SNOOP_PORT0_CACHE_REJECTED_REQUEST	PU0 snoop: Port 0 snoop cache rejected a request
2	73	BGP_PU0_SNOOP_PORT1_CACHE_REJECTED_REQUEST	PU0 snoop: Port 1 snoop cache rejected a request
2	74	BGP_PU0_SNOOP_PORT2_CACHE_REJECTED_REQUEST	PU0 snoop: Port 2 snoop cache rejected a request
2	75	BGP_PU0_SNOOP_PORT3_CACHE_REJECTED_REQUEST	PU0 snoop: Port 3 snoop cache rejected a request
2	76	BGP_PU0_SNOOP_PORT0_HIT_STREAM_REGISTER_IN_ACTIVE_SET	PU0 snoop: Port 0 request hit a stream register in the active set
2	77	BGP_PU0_SNOOP_PORT1_HIT_STREAM_REGISTER_IN_ACTIVE_SET	PU0 snoop: Port 1 request hit a stream register in the active set
2	78	BGP_PU0_SNOOP_PORT2_HIT_STREAM_REGISTER_IN_ACTIVE_SET	PU0 snoop: Port 2 request hit a stream register in the active set
2	79	BGP_PU0_SNOOP_PORT3_HIT_STREAM_REGISTER_IN_ACTIVE_SET	PU0 snoop: Port 3 request hit a stream register in the active set
2	80	BGP_PU0_SNOOP_PORT0_HIT_STREAM_REGISTER_IN_HISTORY_SET	PU0 snoop: Port 0 request hit a stream register in the history set
2	81	BGP_PU0_SNOOP_PORT1_HIT_STREAM_REGISTER_IN_HISTORY_SET	PU0 snoop: Port 1 request hit a stream register in the history set
2	82	BGP_PU0_SNOOP_PORT2_HIT_STREAM_REGISTER_IN_HISTORY_SET	PU0 snoop: Port 2 request hit a stream register in the history set
2	83	BGP_PU0_SNOOP_PORT3_HIT_STREAM_REGISTER_IN_HISTORY_SET	PU0 snoop: Port 3 request hit a stream register in the history set
2	84	BGP_PU0_SNOOP_PORT0_STREAM_REGISTER_REJECTED_REQUEST	PU0 snoop: Port 0 stream register rejected a request
2	85	BGP_PU0_SNOOP_PORT1_STREAM_REGISTER_REJECTED_REQUEST	PU0 snoop: Port 1 stream register rejected a request
2	86	BGP_PU0_SNOOP_PORT2_STREAM_REGISTER_REJECTED_REQUEST	PU0 snoop: Port 2 stream register rejected a request
2	87	BGP_PU0_SNOOP_PORT3_STREAM_REGISTER_REJECTED_REQUEST	PU0 snoop: Port 3 stream register rejected a request
2	88	BGP_PU0_SNOOP_PORT0_RANGE_FILTER_REJECTED_REQUEST	PU0 snoop: Port 0 range filter rejected a request
2	89	BGP_PU0_SNOOP_PORT1_RANGE_FILTER_REJECTED_REQUEST	PU0 snoop: Port 1 range filter rejected a request
2	90	BGP_PU0_SNOOP_PORT2_RANGE_FILTER_REJECTED_REQUEST	PU0 snoop: Port 2 range filter rejected a request
2	91	BGP_PU0_SNOOP_PORT3_RANGE_FILTER_REJECTED_REQUEST	PU0 snoop: Port 3 range filter rejected a request

Group	Counter	Event name	Event description
2	92	BGP_PU0_SNOOP_PORT0_UPDATED_CACHE_LINE	PU0 snoop: Port 0 snoop cache updated cache line
2	93	BGP_PU0_SNOOP_PORT1_UPDATED_CACHE_LINE	PU0 snoop: Port 1 snoop cache updated cache line
2	94	BGP_PU0_SNOOP_PORT2_UPDATED_CACHE_LINE	PU0 snoop: Port 2 snoop cache updated cache line
2	95	BGP_PU0_SNOOP_PORT3_UPDATED_CACHE_LINE	PU0 snoop: Port 3 snoop cache updated cache line
2	96	BGP_PU0_SNOOP_PORT0_FILTERED_BY_CACHE_AND_REGISTERS	PU0 snoop: Port 0 snoop filtered by both snoop cache and filter registers
2	97	BGP_PU0_SNOOP_PORT1_FILTERED_BY_CACHE_AND_REGISTERS	PU0 snoop: Port 1 snoop filtered by both snoop cache and filter registers
2	98	BGP_PU0_SNOOP_PORT2_FILTERED_BY_CACHE_AND_REGISTERS	PU0 snoop: Port 2 snoop filtered by both snoop cache and filter registers
2	99	BGP_PU0_SNOOP_PORT3_FILTERED_BY_CACHE_AND_REGISTERS	PU0 snoop: Port 3 snoop filtered by both snoop cache and filter registers
2	100	BGP_PU1_SNOOP_PORT0_CACHE_REJECTED_REQUEST	PU1 snoop: Port 0 snoop cache rejected a request
2	101	BGP_PU1_SNOOP_PORT1_CACHE_REJECTED_REQUEST	PU1 snoop: Port 1 snoop cache rejected a request
2	102	BGP_PU1_SNOOP_PORT2_CACHE_REJECTED_REQUEST	PU1 snoop: Port 2 snoop cache rejected a request
2	103	BGP_PU1_SNOOP_PORT3_CACHE_REJECTED_REQUEST	PU1 snoop: Port 3 snoop cache rejected a request
2	104	BGP_PU1_SNOOP_PORT0_HIT_STREAM_REGISTER_IN_ACTIVE_SET	PU1 snoop: Port 0 request hit a stream register in the active set
2	105	BGP_PU1_SNOOP_PORT1_HIT_STREAM_REGISTER_IN_ACTIVE_SET	PU1 snoop: Port 1 request hit a stream register in the active set
2	106	BGP_PU1_SNOOP_PORT2_HIT_STREAM_REGISTER_IN_ACTIVE_SET	PU1 snoop: Port 2 request hit a stream register in the active set
2	107	BGP_PU1_SNOOP_PORT3_HIT_STREAM_REGISTER_IN_ACTIVE_SET	PU1 snoop: Port 3 request hit a stream register in the active set
2	108	BGP_PU1_SNOOP_PORT0_HIT_STREAM_REGISTER_IN_HISTORY_SET	PU1 snoop: Port 0 request hit a stream register in the history set
2	109	BGP_PU1_SNOOP_PORT1_HIT_STREAM_REGISTER_IN_HISTORY_SET	PU1 snoop: Port 1 request hit a stream register in the history set
2	110	BGP_PU1_SNOOP_PORT2_HIT_STREAM_REGISTER_IN_HISTORY_SET	PU1 snoop: Port 2 request hit a stream register in the history set
2	111	BGP_PU1_SNOOP_PORT3_HIT_STREAM_REGISTER_IN_HISTORY_SET	PU1 snoop: Port 3 request hit a stream register in the history set
2	112	BGP_PU1_SNOOP_PORT0_STREAM_REGISTER_REJECTED_REQUEST	PU1 snoop: Port 0 stream register rejected a request

Group	Counter	Event name	Event description
2	113	BGP_PU1_SNOOP_PORT1_STREAM_REGISTER_REJECTED_REQUEST	PU1 snoop: Port 1 stream register rejected a request
2	114	BGP_PU1_SNOOP_PORT2_STREAM_REGISTER_REJECTED_REQUEST	PU1 snoop: Port 2 stream register rejected a request
2	115	BGP_PU1_SNOOP_PORT3_STREAM_REGISTER_REJECTED_REQUEST	PU1 snoop: Port 3 stream register rejected a request
2	116	BGP_PU1_SNOOP_PORT0_RANGE_FILTER_REJECTED_REQUEST	PU1 snoop: Port 0 range filter rejected a request
2	117	BGP_PU1_SNOOP_PORT1_RANGE_FILTER_REJECTED_REQUEST	PU1 snoop: Port 1 range filter rejected a request
2	118	BGP_PU1_SNOOP_PORT2_RANGE_FILTER_REJECTED_REQUEST	PU1 snoop: Port 2 range filter rejected a request
2	119	BGP_PU1_SNOOP_PORT3_RANGE_FILTER_REJECTED_REQUEST	PU1 snoop: Port 3 range filter rejected a request
2	120	BGP_PU1_SNOOP_PORT0_UPDATED_CACHE_LINE	PU1 snoop: Port 0 snoop cache updated cache line
2	121	BGP_PU1_SNOOP_PORT1_UPDATED_CACHE_LINE	PU1 snoop: Port 1 snoop cache updated cache line
2	122	BGP_PU1_SNOOP_PORT2_UPDATED_CACHE_LINE	PU1 snoop: Port 2 snoop cache updated cache line
2	123	BGP_PU1_SNOOP_PORT3_UPDATED_CACHE_LINE	PU1 snoop: Port 3 snoop cache updated cache line
2	124	BGP_PU1_SNOOP_PORT0_FILTERED_BY_CACHE_AND_REGISTERS	PU1 snoop: Port 0 snoop filtered by both snoop cache and filter registers
2	125	BGP_PU1_SNOOP_PORT1_FILTERED_BY_CACHE_AND_REGISTERS	PU1 snoop: Port 1 snoop filtered by both snoop cache and filter registers
2	126	BGP_PU1_SNOOP_PORT2_FILTERED_BY_CACHE_AND_REGISTERS	PU1 snoop: Port 2 snoop filtered by both snoop cache and filter registers
2	127	BGP_PU1_SNOOP_PORT3_FILTERED_BY_CACHE_AND_REGISTERS	PU1 snoop: Port 3 snoop filtered by both snoop cache and filter registers
2	128	BGP_TORUS_XP_TOKEN_ACK_PACKETS	Torus: Number of protocol token/ack packets in xp
2	129	BGP_TORUS_XP_ACKS	Torus: Number of protocol ack packets in xp
2	130	BGP_TORUS_XP_VCD0_32BCHUNKS	Torus: Number of 32B chunks sent on dynamic vc 0
2	131	BGP_TORUS_XP_VCD1_32BCHUNKS	Torus: Number of 32B chunks sent on dynamic vc 1
2	132	BGP_TORUS_XP_VCBN_32BCHUNKS	Torus: Number of 32B chunks sent on bubble vc 2
2	133	BGP_TORUS_XP_VCBP_32BCHUNKS	Torus: Number of 32B chunks sent on priority vc 3



Group	Counter	Event name	Event description
2	134	BGP_TORUS_XP_NO_TOKENS	Torus: xp link avail, no vcd0 vcd1 tokens
2	135	BGP_TORUS_XP_NO_VCD0_TOKENS	Torus: xp link avail; no vcd0 vcd; vcbn tokens
2	136	BGP_TORUS_XP_NO_VCBN_TOKENS	Torus: xp link avail; no vcbn tokens
2	137	BGP_TORUS_XP_NO_VCBP_TOKENS	Torus: xp link avail; no vcbp tokens
2	138	BGP_TORUS_XM_TOKEN_ACK_PACKETS	Torus: Number of protocol token/ack packets in xm
2	139	BGP_TORUS_XM_ACKS	Torus: Number of protocol ack packets in xm
2	140	BGP_TORUS_XM_VCD0_32BCHUNKS	Torus: Number of 32B chunks sent on dynamic vc 0
2	141	BGP_TORUS_XM_VCD1_32BCHUNKS	Torus: Number of 32B chunks sent on dynamic vc 1
2	142	BGP_TORUS_XM_VCBN_32BCHUNKS	Torus: Number of 32B chunks sent on bubble vc 2
2	143	BGP_TORUS_XM_VCBP_32BCHUNKS	Torus: Number of 32B chunks sent on priority vc 3
2	144	BGP_TORUS_XM_NO_TOKENS	Torus: xm link avail; no vcd0 vcd1 tokens
2	145	BGP_TORUS_XM_NO_VCD0_TOKENS	Torus: xm link avail; no vcd0 vcd; vcbn tokens
2	146	BGP_TORUS_XM_NO_VCBN_TOKENS	Torus: xm link avail; no vcbn tokens
2	147	BGP_TORUS_XM_NO_VCBP_TOKENS	Torus: xm link avail; no vcbp tokens
2	148	BGP_TORUS_YP_TOKEN_ACK_PACKETS	Torus: Number of protocol token/ack packets in yp
2	149	BGP_TORUS_YP_ACKS	Torus: Number of protocol ack packets in yp
2	150	BGP_TORUS_YP_VCD0_32BCHUNKS	Torus: Number of 32B chunks sent on dynamic vc 0
2	151	BGP_TORUS_YP_VCD1_32BCHUNKS	Torus: Number of 32B chunks sent on dynamic vc 1
2	152	BGP_TORUS_YP_VCBN_32BCHUNKS	Torus: Number of 32B chunks sent on bubble vc 2
2	153	BGP_TORUS_YP_VCBP_32BCHUNKS	Torus: Number of 32B chunks sent on priority vc 3
2	154	BGP_TORUS_YP_NO_TOKENS	Torus: yp link avail; no vcd0 vcd1tokens
2	155	BGP_TORUS_YP_NO_VCD0_TOKENS	Torus: yp link avail; no vcd0 vcd; vcbn tokens
2	156	BGP_TORUS_YP_NO_VCBN_TOKENS	Torus: yp link avail; no vcbn tokens
2	157	BGP_TORUS_YP_NO_VCBP_TOKENS	Torus: yp link avail; no vcbp tokens

Group	Counter	Event name	Event description
2	158	BGP_TORUS_YM_TOKEN_ACK_PACKETS	Torus: Number of protocol token/ack packets in ym
2	159	BGP_TORUS_YM_ACKS	Torus: Number of protocol ack packets in ym
2	160	BGP_TORUS_YM_VCD0_32BCHUNKS	Torus: Number of 32B chunks sent on dynamic vc 0
2	161	BGP_TORUS_YM_VCD1_32BCHUNKS	Torus: Number of 32B chunks sent on dynamic vc 1
2	162	BGP_TORUS_YM_VCBN_32BCHUNKS	Torus: Number of 32B chunks sent on bubble vc 2
2	163	BGP_TORUS_YM_VCBP_32BCHUNKS	Torus: Number of 32B chunks sent on priority vc 3
2	164	BGP_TORUS_YM_NO_TOKENS	Torus: ym link avail; no vcd0 vcd1 tokens
2	165	BGP_TORUS_YM_NO_VCD0_TOKENS	Torus: ym link avail; no vcd0 vcd; vcbn tokens
2	166	BGP_TORUS_YM_NO_VCBN_TOKENS	Torus: ym link avail; no vcbn tokens
2	167	BGP_TORUS_YM_NO_VCBP_TOKENS	Torus: ym link avail; no vcbp tokens
2	168	BGP_TORUS_ZP_TOKEN_ACK_PACKETS	Torus: Number of protocol token/ack packets in zp
2	169	BGP_TORUS_ZP_ACKS	Torus: Number of protocol ack packets in zp
2	170	BGP_TORUS_ZP_VCD0_32BCHUNKS	Torus: Number of 32B chunks sent on dynamic vc 0
2	171	BGP_TORUS_ZP_VCD1_32BCHUNKS	Torus: Number of 32B chunks sent on dynamic vc 1
2	172	BGP_TORUS_ZP_VCBN_32BCHUNKS	Torus: Number of 32B chunks sent on bubble vc 2
2	173	BGP_TORUS_ZP_VCBP_32BCHUNKS	Torus: Number of 32B chunks sent on priority vc 3
2	174	BGP_TORUS_ZP_NO_TOKENS	Torus: zp link avail; no vcd0 vcd1 tokens
2	175	BGP_TORUS_ZP_NO_VCD0_TOKENS	Torus: zp link avail; no vcd0 vcd; vcbn tokens
2	176	BGP_TORUS_ZP_NO_VCBN_TOKENS	Torus: zp link avail; no vcbn tokens
2	177	BGP_TORUS_ZP_NO_VCBP_TOKENS	Torus: zp link avail; no vcbp tokens
2	178	BGP_TORUS_ZM_TOKEN_ACK_PACKETS	Torus: Number of protocol token/ack packets in zm
2	179	BGP_TORUS_ZM_ACKS	Torus: Number of protocol ack packets in zm
2	180	BGP_TORUS_ZM_VCD0_32BCHUNKS	Torus: Number of 32B chunks sent on dynamic vc 0

Group	Counter	Event name	Event description
2	181	BGP_TORUS_ZM_VCD1_32BCHUNKS	Torus: Number of 32B chunks sent on dynamic vc 1
2	182	BGP_TORUS_ZM_VCBN_32BCHUNKS	Torus: Number of 32B chunks sent on bubble vc 2
2	183	BGP_TORUS_ZM_VCBP_32BCHUNKS	Torus: Number of 32B chunks sent on priority vc 3
2	184	BGP_TORUS_ZM_NO_TOKENS	Torus: zm link avail; no vcd0 vcd1 tokens
2	185	BGP_TORUS_ZM_NO_VCD0_TOKENS	Torus: zm link avail; no vcd0 vcd; vcbn tokens
2	186	BGP_TORUS_RESERVED_1	Torus: zm link avail; no vcbn tokens
2	187	BGP_TORUS_RESERVED_2	Torus: zm link avail; no vcbp tokens
2	188	BGP_DMA_RESERVED_7	DMA: Reserved
2	189	BGP_DMA_RESERVED_8	DMA: Reserved
2	190	BGP_DMA_RESERVED_9	DMA: Reserved
2	191	BGP_DMA_RESERVED_10	DMA: Reserved
2	192	BGP_DMA_RESERVED_11	DMA: Reserved
2	193	BGP_DMA_RESERVED_12	DMA: Reserved
2	194	BGP_DMA_RESERVED_13	DMA: Reserved
2	195	BGP_DMA_RESERVED_14	DMA: Reserved
2	196	BGP_DMA_RESERVED_15	DMA: Reserved
2	197	BGP_DMA_RESERVED_16	DMA: Reserved
2	198	BGP_DMA_RESERVED_17	DMA: Reserved
2	199	BGP_DMA_RESERVED_18	DMA: Reserved
2	200	BGP_DMA_RESERVED_19	DMA: Reserved
2	201	BGP_DMA_RESERVED_20	DMA: Reserved
2	202	BGP_DMA_RESERVED_21	DMA: Reserved
2	203	BGP_DMA_RESERVED_22	DMA: Reserved
2	204	BGP_COL_AR2_ABORT_UM2	Collective: Arbiter receiver 2 abort
2	205	BGP_COL_AR1_ABORT_UM2	Collective: Arbiter receiver 1 abort
2	206	BGP_COL_AR0_ABORT_UM2	Collective: Arbiter receiver 0 abort
2	207	BGP_COL_A_LOCAL_CLIENT_ABORT	Collective: Arbiter local client abort
2	208	BGP_COL_AR0_VC0_FULL	Collective: Receiver 0 vc0 full
2	209	BGP_COL_AR0_VC1_FULL	Collective: Receiver 0 vc1 full
2	210	BGP_COL_AR1_VC0_FULL	Collective: Receiver 1 vc0 full
2	211	BGP_COL_AR1_VC1_FULL	Collective: Receiver 1 vc1 full

Group	Counter	Event name	Event description
2	212	BGP_COL_AR2_VC0_FULL	Collective: Receiver 2 vc0 full
2	213	BGP_COL_AR2_VC1_FULL	Collective: Receiver 2 vc1 full
2	214	BGP_COL_AS0_VC0_EMPTY	Collective: Sender 0 vc0 empty
2	215	BGP_COL_AS0_VC1_EMPTY	Collective: Sender 0 vc1 empty
2	216	BGP_COL_AS0_RESENDS	Collective: Sender 0 resend attempts
2	217	BGP_COL_AS1_VC0_EMPTY	Collective: Sender 1 vc0 empty
2	218	BGP_COL_AS1_VC1_EMPTY	Collective: Sender 1 vc1 empty
2	219	BGP_COL_AS1_RESENDS	Collective: Sender 1 resend attempts
2	220	BGP_COL_AS2_VC0_EMPTY	Collective: Sender 2 vc0 empty
2	221	BGP_COL_AS2_VC1_EMPTY	Collective: Sender 2 vc1 empty
2	222	BGP_COL_AS2_RESENDS	Collective: Sender 2 resend attempts
2	223	BGP_MISC_RESERVED_4	Misc: Reserved
2	224	BGP_MISC_RESERVED_5	Misc: Reserved
2	225	BGP_MISC_RESERVED_6	Misc: Reserved
2	226	BGP_MISC_RESERVED_7	Misc: Reserved
2	227	BGP_MISC_RESERVED_8	Misc: Reserved
2	228	BGP_MISC_RESERVED_9	Misc: Reserved
2	229	BGP_MISC_RESERVED_10	Misc: Reserved
2	230	BGP_MISC_RESERVED_11	Misc: Reserved
2	231	BGP_MISC_RESERVED_12	Misc: Reserved
2	232	BGP_MISC_RESERVED_13	Misc: Reserved
2	233	BGP_MISC_RESERVED_14	Misc: Reserved
2	234	BGP_MISC_RESERVED_15	Misc: Reserved
2	235	BGP_MISC_RESERVED_16	Misc: Reserved
2	236	BGP_MISC_RESERVED_17	Misc: Reserved
2	237	BGP_MISC_RESERVED_18	Misc: Reserved
2	238	BGP_MISC_RESERVED_19	Misc: Reserved
2	239	BGP_MISC_RESERVED_20	Misc: Reserved
2	240	BGP_MISC_RESERVED_21	Misc: Reserved
2	241	BGP_MISC_RESERVED_22	Misc: Reserved
2	242	BGP_MISC_RESERVED_23	Misc: Reserved
2	243	BGP_MISC_RESERVED_24	Misc: Reserved
2	244	BGP_MISC_RESERVED_25	Misc: Reserved
2	245	BGP_MISC_RESERVED_26	Misc: Reserved

Group	Counter	Event name	Event description
2	246	BGP_MISC_RESERVED_27	Misc: Reserved
2	247	BGP_MISC_RESERVED_28	Misc: Reserved
2	248	BGP_MISC_RESERVED_29	Misc: Reserved
2	249	BGP_MISC_RESERVED_30	Misc: Reserved
2	250	BGP_MISC_RESERVED_31	Misc: Reserved
2	251	BGP_MISC_RESERVED_32	Misc: Reserved
2	252	BGP_MISC_RESERVED_33	Misc: Reserved
2	253	BGP_MISC_RESERVED_34	Misc: Reserved
2	254	BGP_MISC_RESERVED_35	Misc: Reserved
2	255	BGP_MISC_ELAPSED_TIME_UM2	Misc: Elapsed time
3	0	BGP_PU2_JPIPE_INSTRUCTIONS_UM3	PU2: Number of J-pipe instructions
3	1	BGP_PU2_JPIPE_ADD_SUB_UM3	PU2: PowerPC Add/Sub in J-pipe
3	2	BGP_PU2_JPIPE_LOGICAL_OPS_UM3	PU2: PowerPC logical operations in J-pipe
3	3	BGP_PU2_JPIPE_SHROTMK_UM3	PU2: Shift, rotate, mask instructions
3	4	BGP_PU2_IPIPE_INSTRUCTIONS_UM3	PU2: Number of I-pipe instructions
3	5	BGP_PU2_IPIPE_MULT_DIV_UM3	PU2: PowerPC Mul/Div in I-pipe
3	6	BGP_PU2_IPIPE_ADD_SUB_UM3	PU2: PowerPC Add/Sub in I-pipe
3	7	BGP_PU2_IPIPE_LOGICAL_OPS_UM3	PU2: PowerPC logical operations in I-pipe
3	8	BGP_PU2_IPIPE_SHROTMK_UM3	PU2: Shift, rotate, mask instructions
3	9	BGP_PU2_IPIPE_BRANCHES_UM3	PU2: PowerPC branches
3	10	BGP_PU2_IPIPE_TLB_OPS_UM3	PU2: PowerPC TLB operations
3	11	BGP_PU2_IPIPE_PROCESS_CONTROL_UM3	PU2: PowerPC process control
3	12	BGP_PU2_IPIPE_OTHER_UM3	PU2: PowerPC other I-pipe operations
3	13	BGP_PU2_DCACHE_LINEFILLINPROG_UM3	PU2: Number of cycles D-cache LineFillInProgress
3	14	BGP_PU2_ICACHE_LINEFILLINPROG_UM3	PU2: Number of cycles I-cache LineFillInProgress
3	15	BGP_PU2_DCACHE_MISS_UM3	PU2: Accesses to D cache that miss in D Cache
3	16	BGP_PU2_DCACHE_HIT_UM3	PU2: Accesses to D cache that hit in D Cache
3	17	BGP_PU2_DATA_LOADS_UM3	PU2: PowerPC data loads
3	18	BGP_PU2_DATA_STORES_UM3	PU2: PowerPC data stores
3	19	BGP_PU2_DCACHE_OPS_UM3	PU2: D cache operations



Group	Counter	Event name	Event description
3	36	BGP_PU3_JPIPE_ADD_SUB_UM3	PU3: PowerPC Add/Sub in J-pipe
3	37	BGP_PU3_JPIPE_LOGICAL_OPS_UM3	PU3: PowerPC logical operations in J-pipe
3	38	BGP_PU3_JPIPE_SHROTMK_UM3	PU3: Shift, rotate, mask instructions
3	39	BGP_PU3_IPIPE_INSTRUCTIONS_UM3	PU3: Number of I-pipe instructions
3	40	BGP_PU3_IPIPE_MULT_DIV_UM3	PU3: PowerPC Mul/Div in I-pipe
3	41	BGP_PU3_IPIPE_ADD_SUB_UM3	PU3: PowerPC Add/Sub in I-pipe
3	42	BGP_PU3_IPIPE_LOGICAL_OPS_UM3	PU3: PowerPC logical operations in I-pipe
3	43	BGP_PU3_IPIPE_SHROTMK_UM3	PU3: Shift, rotate, mask instructions
3	44	BGP_PU3_IPIPE_BRANCHES_UM3	PU3: PowerPC branches
3	45	BGP_PU3_IPIPE_TLB_OPS_UM3	PU3: PowerPC TLB operations
3	46	BGP_PU3_IPIPE_PROCESS_CONTROL_UM3	PU3: PowerPC process control
3	47	BGP_PU3_IPIPE_OTHER_UM3	PU3: PowerPC other I-pipe operations
3	48	BGP_PU3_DCACHE_LINEFILLINPROG_UM3	PU3: Number of cycles D-cache LineFillInProgress
3	49	BGP_PU3_ICACHE_LINEFILLINPROG_UM3	PU3: Number of cycles I-cache LineFillInProgress
3	50	BGP_PU3_DCACHE_MISS_UM3	PU3: Accesses to D cache that miss in D Cache
3	51	BGP_PU3_DCACHE_HIT_UM3	PU3: Accesses to D cache that hit in D Cache
3	52	BGP_PU3_DATA_LOADS_UM3	PU3: PowerPC data loads
3	53	BGP_PU3_DATA_STORES_UM3	PU3: PowerPC data stores
3	54	BGP_PU3_DCACHE_OPS_UM3	PU3: D cache operations
3	55	BGP_PU3_ICACHE_MISS_UM3	PU3: Accesses to I cache that miss in I Cache
3	56	BGP_PU3_ICACHE_HIT_UM3	PU3: Accesses to I cache that hit in I Cache
3	57	BGP_PU3_FPU_ADD_SUB_1_UM3	PU3: PowerPC FP Add/Sub (fadd, fadds, fsub, fsubs)
3	58	BGP_PU3_FPU_MULT_1_UM3	PU3: PowerPC FP Mult (fmul, fmuls)
3	59	BGP_PU3_FPU_FMA_2_UM3	PU3: PowerPC FP FMA (fmadd, fmadds, fmsub, fmsubs, fnmadd, fnmadds, fnmsub, fnmsubs; one result generated per instruction, two flops)
3	60	BGP_PU3_FPU_DIV_1_UM3	PU3: PowerPC FP Div (fddiv, fddivs; Single Pipe Divide)

Group	Counter	Event name	Event description
3	61	BGP_PU3_FPU_OTHER_NON_STORAGE_OPS_UM3	PU3: PowerPC FP remaining non-storage instructions (fabs, fnabs, frsp, fctiw, fctiwz, fres, frsqte, fsel, fmr, fneg, fcmphu, fcmpho, mffs, mcrfs, mtfshi, mtfshf, mtfshb0, mtfshb1)
3	62	BGP_PU3_FPU_ADD_SUB_2_UM3	PU3: Dual pipe Add/Sub (fpadd, fpsub)
3	63	BGP_PU3_FPU_MULT_2_UM3	PU3: Dual pipe Mult (fpmul, fxmul, fxpmul, fxsmul)
3	64	BGP_PU3_FPU_FMA_4_UM3	PU3: Dual pipe FMAs (fpmadd, fpmnadd, fpmsub, fpnmsub, fxmadd, fxnmadd, fxmsub, fxnmsub, fxcpmadd, fxcsmadd, fxcnmmadd, fxcnsmadd, fxcpmsub, fxcmsub, fxcnpsub, fxcnmsub, fxcnpsma, fxcnpsma, fxcnpsma, fxcnpsma, fxcnpsma, fxcnpsma, fxcnpsma, fxcnpsma; two results generated per instruction, four flops)
3	65	BGP_PU3_FPU_DUAL_PIPE_OTHER_NON_STORAGE_OPS_UM3	PU3: Dual pipe remaining non-storage instructions (fpmr, fpneg, fsmr, fsneg, fxmr, fsmfp, fsmtp, fpabs, fpnabs, fsabs, fsnabs, frsp, fpctiw, fpctiwz, fpre, frsqte, fpsel, fscmp)
3	66	BGP_PU3_FPU_QUADWORD_LOADS_UM3	PU3: Quad Word Loads (ffpdx, lfpdux, lfxdx, lfxdux)
3	67	BGP_PU3_FPU_OTHER_LOADS_UM3	PU3: All other Loads (lfs, lfsx, lfsu, lfsux, lfpsx, fpsux, lfdsx, lfdsux, lfssx, lfssux, lfd, lfdx, lfdu, lfdux, lfxsx, lfxsux)
3	68	BGP_PU3_FPU_QUADWORD_STORES_UM3	PU3: Quad Word Stores (stfpdx, stfpdux, stfxdx, stfxdux)
3	69	BGP_PU3_FPU_OTHER_STORES_UM3	PU3: All other FPU Stores (stfs, stfsx, stfsu, stfsux, stfd, stfdx, stfdu, stfdx, stfiwx, stfpx, stfpx, stfpiwx, stfsdx, stfsdux, stfssx, stfssux, stfxsx, stfxsux)
3	70	BGP_PU2_L1_INVALIDATION_UM3	PU2 L1: Invalidation requested
3	71	BGP_PU3_L1_INVALIDATION_UM3	PU3 L1: Invalidation requested
3	72	BGP_COL_A_CH2_VC0_HAVE	Collective: Arbiter ch2_vc0_have
3	73	BGP_COL_A_CH1_VC0_HAVE	Collective: Arbiter ch1_vc0_have
3	74	BGP_COL_A_CH0_VC0_HAVE	Collective: Arbiter ch0_vc0_have
3	75	BGP_COL_A_INJECT_VC0_HAVE	Collective: Arbiter inj_vc0_have
3	76	BGP_COL_A_CH2_VC1_HAVE	Collective: Arbiter ch2_vc1_have
3	77	BGP_COL_A_CH1_VC1_HAVE	Collective: Arbiter ch1_vc1_have
3	78	BGP_COL_A_CH0_VC1_HAVE	Collective: Arbiter ch0_vc1_have
3	79	BGP_COL_A_INJECT_VC1_HAVE	Collective: Arbiter inj_vc1_have
3	80	BGP_COL_AC_GREEDY_MODE	Collective: arbiter_core greedy_mode



Group	Counter	Event name	Event description
3	81	BGP_COL_AC_PENDING_REQUESTS_UM3	Collective: arbiter_core requests pending
3	82	BGP_COL_AC_WAITING_REQUESTS_UM3	Collective: arbiter_core requests waiting (ready to go)
3	83	BGP_COL_ACLS0_WINS	Collective: Arbiter class 0 wins
3	84	BGP_COL_ACLS1_WINS	Collective: Arbiter class 1 wins
3	85	BGP_COL_ACLS2_WINS	Collective: Arbiter class 2 wins
3	86	BGP_COL_ACLS3_WINS	Collective: Arbiter class 3 wins
3	87	BGP_COL_ACLS4_WINS	Collective: Arbiter class 4 wins
3	88	BGP_COL_ACLS5_WINS	Collective: Arbiter class 5 wins
3	89	BGP_COL_ACLS6_WINS	Collective: Arbiter class 6 wins
3	90	BGP_COL_ACLS7_WINS	Collective: Arbiter class 7 wins
3	91	BGP_COL_ACLS8_WINS	Collective: Arbiter class 8 wins
3	92	BGP_COL_ACLS9_WINS	Collective: Arbiter class 9 wins
3	93	BGP_COL_ACLS10_WINS	Collective: Arbiter class 10 wins
3	94	BGP_COL_ACLS11_WINS	Collective: Arbiter class 11 wins
3	95	BGP_COL_ACLS12_WINS	Collective: Arbiter class 12 wins
3	96	BGP_COL_ACLS13_WINS	Collective: Arbiter class 13 wins
3	97	BGP_COL_ACLS14_WINS	Collective: Arbiter class 14 wins
3	98	BGP_COL_ACLS15_WINS	Collective: Arbiter class 15 wins
3	99	BGP_COL_AS2_BUSY	Collective: Arbiter sender 2 busy
3	100	BGP_COL_AS1_BUSY	Collective: Arbiter sender 1 busy
3	101	BGP_COL_AS1_BUSY_RECEPTION	Collective: Arbiter sender 0 busy
3	102	BGP_COL_ALC_BUSY	Collective: Arbiter local client busy (reception)
3	103	BGP_COL_AR2_BUSY	Collective: Arbiter receiver 2 busy
3	104	BGP_COL_AR1_BUSY	Collective: Arbiter receiver 1 busy
3	105	BGP_COL_AR0_BUSY	Collective: Arbiter receiver 0 busy
3	106	BGP_COL_ALC_BUSY_INJECT	Collective: Arbiter local client busy (injection)
3	107	BGP_COL_ALU_BUSY	Collective: Arbiter ALU busy
3	108	BGP_COL_AR2_ABORT_UM3	Collective: Arbiter receiver 2 abort
3	109	BGP_COL_AR1_ABORT_UM3	Collective: Arbiter receiver 1 abort
3	110	BGP_COL_AR0_ABORT_UM3	Collective: Arbiter receiver 0 abort
3	111	BGP_COL_ALC_ABORT	Collective: Arbiter local client abort

Group	Counter	Event name	Event description
3	112	BGP_COL_AR2_PACKET_TAKEN_UM3	Collective: Arbiter receiver 2 packet taken
3	113	BGP_COL_AR1_PACKET_TAKEN_UM3	Collective: Arbiter receiver 1 packet taken
3	114	BGP_COL_AR0_PACKET_TAKEN_UM3	Collective: Arbiter receiver 0 packet taken
3	115	BGP_COL_ALC_PACKET_TAKEN_UM3	Collective: Arbiter local client packet taken
3	116	BGP_COL_AR0_VC0_DATA_PACKET_RECEIVED	Collective: Receiver 0 vc0 data packet received
3	117	BGP_COL_AR0_VC1_DATA_PACKET_RECEIVED	Collective: Receiver 0 vc1 data packet received
3	118	BGP_COL_AR0_VC1_FULL_UM3	Collective: Receiver 0 vc1 full
3	119	BGP_COL_AR0_HEADER_PARITY_ERROR_UM3	Collective: Receiver 0 header parity error
3	120	BGP_COL_AR1_VC0_DATA_PACKET_RECEIVED	Collective: Receiver 1 vc0 data packet received
3	121	BGP_COL_AR1_VC1_DATA_PACKET_RECEIVED	Collective: Receiver 1 vc1 data packet received
3	122	BGP_COL_AR1_VC0_FULL_UM3	Collective: Receiver 1 vc0 full
3	123	BGP_COL_AR1_VC1_FULL_UM3	Collective: Receiver 1 vc1 full
3	124	BGP_COL_AR2_VC0_DATA_PACKET_RECEIVED	Collective: Receiver 2 vc0 data packet received
3	125	BGP_COL_AR2_VC1_DATA_PACKET_RECEIVED	Collective: Receiver 2 vc1 data packet received
3	126	BGP_COL_AR2_VC0_FULL_UM3	Collective: Receiver 2 vc0 full
3	127	BGP_COL_AR2_VC1_FULL_UM3	Collective: Receiver 2 vc1 full
3	128	BGP_COL_AS0_VC0_EMPTY_UM3	Collective: Sender 0 vc0 empty
3	129	BGP_COL_AS0_VC1_EMPTY_UM3	Collective: Sender 0 vc1 empty
3	130	BGP_COL_AS0_VC0_DATA_PACKETS_SENT_UM3	Collective: Sender 0 vc0 DATA packets sent
3	131	BGP_COL_AS0_VC1_DATA_PACKETS_SENT_UM3	Collective: Sender 0 vc1 DATA packets sent
3	132	BGP_COL_AS0_RESENDS_UM3	Collective: Sender 0 resend attempts
3	133	BGP_COL_AS1_VC0_EMPTY_UM3	Collective: Sender 1 vc0 empty
3	134	BGP_COL_AS1_VC1_EMPTY_UM3	Collective: Sender 1 vc1 empty
3	135	BGP_COL_AS1_VC0_DATA_PACKETS_SENT_UM3	Collective: Sender 1 vc0 DATA packets sent
3	136	BGP_COL_AS1_VC1_DATA_PACKETS_SENT_UM3	Collective: Sender 1 vc1 DATA packets sent

Group	Counter	Event name	Event description
3	137	BGP_COL_AS1_RESENDS_UM3	Collective: Sender 1 resend attempts
3	138	BGP_COL_AS2_VC0_EMPTY_UM3	Collective: Sender 2 vc0 empty
3	139	BGP_COL_AS2_VC1_EMPTY_UM3	Collective: Sender 2 vc1 empty
3	140	BGP_COL_AS2_VC0_DATA_PACKETS_SENT_UM3	Collective: Sender 2 vc0 DATA packets sent
3	141	BGP_COL_AS2_VC1_DATA_PACKETS_SENT_UM3	Collective: Sender 2 vc1 DATA packets sent
3	142	BGP_COL_AS2_RESENDS_UM3	Collective: Sender 2 resend attempts
3	143	BGP_COL_INJECT_VC0_HEADER_ADDED	Collective: Injection vc0 header added
3	144	BGP_COL_INJECT_VC1_HEADER_ADDED	Collective: Injection vc1 header added
3	145	BGP_COL_RECEPTION_VC0_PACKED_ADDED	Collective: Reception vc0 packet added
3	146	BGP_COL_RECEPTION_VC1_PACKED_ADDED	Collective: Reception vc1 packet added
3	147	BGP_PU2_SNOOP_PORT0_CACHE_REJECTED_REQUEST	PU2 snoop: Port 0 snoop cache rejected a request
3	148	BGP_PU2_SNOOP_PORT1_CACHE_REJECTED_REQUEST	PU2 snoop: Port 1 snoop cache rejected a request
3	149	BGP_PU2_SNOOP_PORT2_CACHE_REJECTED_REQUEST	PU2 snoop: Port 2 snoop cache rejected a request
3	150	BGP_PU2_SNOOP_PORT3_CACHE_REJECTED_REQUEST	PU2 snoop: Port 3 snoop cache rejected a request
3	151	BGP_PU2_SNOOP_PORT0_HIT_STREAM_REGISTER_IN_ACTIVE_SET	PU2 snoop: Port 0 request hit a stream register in the active set
3	152	BGP_PU2_SNOOP_PORT1_HIT_STREAM_REGISTER_IN_ACTIVE_SET	PU2 snoop: Port 1 request hit a stream register in the active set
3	153	BGP_PU2_SNOOP_PORT2_HIT_STREAM_REGISTER_IN_ACTIVE_SET	PU2 snoop: Port 2 request hit a stream register in the active set
3	154	BGP_PU2_SNOOP_PORT3_HIT_STREAM_REGISTER_IN_ACTIVE_SET	PU2 snoop: Port 3 request hit a stream register in the active set
3	155	BGP_PU2_SNOOP_PORT0_HIT_STREAM_REGISTER_IN_HISTORY_SET	PU2 snoop: Port 0 request hit a stream register in the history set
3	156	BGP_PU2_SNOOP_PORT1_HIT_STREAM_REGISTER_IN_HISTORY_SET	PU2 snoop: Port 1 request hit a stream register in the history set
3	157	BGP_PU2_SNOOP_PORT2_HIT_STREAM_REGISTER_IN_HISTORY_SET	PU2 snoop: Port 2 request hit a stream register in the history set
3	158	BGP_PU2_SNOOP_PORT3_HIT_STREAM_REGISTER_IN_HISTORY_SET	PU2 snoop: Port 3 request hit a stream register in the history set
3	159	BGP_PU2_SNOOP_PORT0_STREAM_REGISTER_REJECTED_REQUEST	PU2 snoop: Port 0 stream register rejected a request
3	160	BGP_PU2_SNOOP_PORT1_STREAM_REGISTER_REJECTED_REQUEST	PU2 snoop: Port 1 stream register rejected a request

Group	Counter	Event name	Event description
3	161	BGP_PU2_SNOOP_PORT2_STREAM_REGISTER_REJECTED_REQUEST	PU2 snoop: Port 2 stream register rejected a request
3	162	BGP_PU2_SNOOP_PORT3_STREAM_REGISTER_REJECTED_REQUEST	PU2 snoop: Port 3 stream register rejected a request
3	163	BGP_PU2_SNOOP_PORT0_RANGE_FILTER_REJECTED_REQUEST	PU2 snoop: Port 0 range filter rejected a request
3	164	BGP_PU2_SNOOP_PORT1_RANGE_FILTER_REJECTED_REQUEST	PU2 snoop: Port 1 range filter rejected a request
3	165	BGP_PU2_SNOOP_PORT2_RANGE_FILTER_REJECTED_REQUEST	PU2 snoop: Port 2 range filter rejected a request
3	166	BGP_PU2_SNOOP_PORT3_RANGE_FILTER_REJECTED_REQUEST	PU2 snoop: Port 3 range filter rejected a request
3	167	BGP_PU2_SNOOP_PORT0_UPDATED_CACHE_LINE	PU2 snoop: Port 0 snoop cache updated cache line
3	168	BGP_PU2_SNOOP_PORT1_UPDATED_CACHE_LINE	PU2 snoop: Port 1 snoop cache updated cache line
3	169	BGP_PU2_SNOOP_PORT2_UPDATED_CACHE_LINE	PU2 snoop: Port 2 snoop cache updated cache line
3	170	BGP_PU2_SNOOP_PORT3_UPDATED_CACHE_LINE	PU2 snoop: Port 3 snoop cache updated cache line
3	171	BGP_PU2_SNOOP_PORT0_FILTERED_BY_CACHE_AND_REGISTERS	PU2 snoop: Port 0 snoop filtered by both snoop cache and filter registers
3	172	BGP_PU2_SNOOP_PORT1_FILTERED_BY_CACHE_AND_REGISTERS	PU2 snoop: Port 1 snoop filtered by both snoop cache and filter registers
3	173	BGP_PU2_SNOOP_PORT2_FILTERED_BY_CACHE_AND_REGISTERS	PU2 snoop: Port 2 snoop filtered by both snoop cache and filter registers
3	174	BGP_PU2_SNOOP_PORT3_FILTERED_BY_CACHE_AND_REGISTERS	PU2 snoop: Port 3 snoop filtered by both snoop cache and filter registers
3	175	BGP_PU3_SNOOP_PORT0_CACHE_REJECTED_REQUEST	PU3 snoop: Port 0 snoop cache rejected a request
3	176	BGP_PU3_SNOOP_PORT1_CACHE_REJECTED_REQUEST	PU3 snoop: Port 1 snoop cache rejected a request
3	177	BGP_PU3_SNOOP_PORT2_CACHE_REJECTED_REQUEST	PU3 snoop: Port 2 snoop cache rejected a request
3	178	BGP_PU3_SNOOP_PORT3_CACHE_REJECTED_REQUEST	PU3 snoop: Port 3 snoop cache rejected a request
3	179	BGP_PU3_SNOOP_PORT0_HIT_STREAM_REGISTER_IN_ACTIVE_SET	PU3 snoop: Port 0 request hit a stream register in the active set
3	180	BGP_PU3_SNOOP_PORT1_HIT_STREAM_REGISTER_IN_ACTIVE_SET	PU3 snoop: Port 1 request hit a stream register in the active set
3	181	BGP_PU3_SNOOP_PORT2_HIT_STREAM_REGISTER_IN_ACTIVE_SET	PU3 snoop: Port 2 request hit a stream register in the active set

Group	Counter	Event name	Event description
3	182	BGP_PU3_SNOOP_PORT3_HIT_STREAM_REGISTER_IN_ACTIVE_SET	PU3 snoop: Port 3 request hit a stream register in the active set
3	183	BGP_PU3_SNOOP_PORT0_HIT_STREAM_REGISTER_IN_HISTORY_SET	PU3 snoop: Port 0 request hit a stream register in the history set
3	184	BGP_PU3_SNOOP_PORT1_HIT_STREAM_REGISTER_IN_HISTORY_SET	PU3 snoop: Port 1 request hit a stream register in the history set
3	185	BGP_PU3_SNOOP_PORT2_HIT_STREAM_REGISTER_IN_HISTORY_SET	PU3 snoop: Port 2 request hit a stream register in the history set
3	186	BGP_PU3_SNOOP_PORT3_HIT_STREAM_REGISTER_IN_HISTORY_SET	PU3 snoop: Port 3 request hit a stream register in the history set
3	187	BGP_PU3_SNOOP_PORT0_STREAM_REGISTER_REJECTED_REQUEST	PU3 snoop: Port 0 stream register rejected a request
3	188	BGP_PU3_SNOOP_PORT1_STREAM_REGISTER_REJECTED_REQUEST	PU3 snoop: Port 1 stream register rejected a request
3	189	BGP_PU3_SNOOP_PORT2_STREAM_REGISTER_REJECTED_REQUEST	PU3 snoop: Port 2 stream register rejected a request
3	190	BGP_PU3_SNOOP_PORT3_STREAM_REGISTER_REJECTED_REQUEST	PU3 snoop: Port 3 stream register rejected a request
3	191	BGP_PU3_SNOOP_PORT0_RANGE_FILTER_REJECTED_REQUEST	PU3 snoop: Port 0 range filter rejected a request
3	192	BGP_PU3_SNOOP_PORT1_RANGE_FILTER_REJECTED_REQUEST	PU3 snoop: Port 1 range filter rejected a request
3	193	BGP_PU3_SNOOP_PORT2_RANGE_FILTER_REJECTED_REQUEST	PU3 snoop: Port 2 range filter rejected a request
3	194	BGP_PU3_SNOOP_PORT3_RANGE_FILTER_REJECTED_REQUEST	PU3 snoop: Port 3 range filter rejected a request
3	195	BGP_PU3_SNOOP_PORT0_UPDATED_CACHE_LINE	PU3 snoop: Port 0 snoop cache updated cache line
3	196	BGP_PU3_SNOOP_PORT1_UPDATED_CACHE_LINE	PU3 snoop: Port 1 snoop cache updated cache line
3	197	BGP_PU3_SNOOP_PORT2_UPDATED_CACHE_LINE	PU3 snoop: Port 2 snoop cache updated cache line
3	198	BGP_PU3_SNOOP_PORT3_UPDATED_CACHE_LINE	PU3 snoop: Port 3 snoop cache updated cache line
3	199	BGP_PU3_SNOOP_PORT0_FILTERED_BY_CACHE_AND_REGISTERS	PU3 snoop: Port 0 snoop filtered by both snoop cache and filter registers
3	200	BGP_PU3_SNOOP_PORT1_FILTERED_BY_CACHE_AND_REGISTERS	PU3 snoop: Port 1 snoop filtered by both snoop cache and filter registers
3	201	BGP_PU3_SNOOP_PORT2_FILTERED_BY_CACHE_AND_REGISTERS	PU3 snoop: Port 2 snoop filtered by both snoop cache and filter registers
3	202	BGP_PU3_SNOOP_PORT3_FILTERED_BY_CACHE_AND_REGISTERS	PU3 snoop: Port 3 snoop filtered by both snoop cache and filter registers

Group	Counter	Event name	Event description
3	203	BGP_MISC_RESERVED_36	Misc: Reserved
3	204	BGP_MISC_RESERVED_37	Misc: Reserved
3	205	BGP_MISC_RESERVED_38	Misc: Reserved
3	206	BGP_MISC_RESERVED_39	Misc: Reserved
3	207	BGP_MISC_RESERVED_40	Misc: Reserved
3	208	BGP_MISC_RESERVED_41	Misc: Reserved
3	209	BGP_MISC_RESERVED_42	Misc: Reserved
3	210	BGP_MISC_RESERVED_43	Misc: Reserved
3	211	BGP_MISC_RESERVED_44	Misc: Reserved
3	212	BGP_MISC_RESERVED_45	Misc: Reserved
3	213	BGP_MISC_RESERVED_46	Misc: Reserved
3	214	BGP_MISC_RESERVED_47	Misc: Reserved
3	215	BGP_MISC_RESERVED_48	Misc: Reserved
3	216	BGP_MISC_RESERVED_49	Misc: Reserved
3	217	BGP_MISC_RESERVED_50	Misc: Reserved
3	218	BGP_MISC_RESERVED_51	Misc: Reserved
3	219	BGP_MISC_RESERVED_52	Misc: Reserved
3	220	BGP_MISC_RESERVED_53	Misc: Reserved
3	221	BGP_MISC_RESERVED_54	Misc: Reserved
3	222	BGP_MISC_RESERVED_55	Misc: Reserved
3	223	BGP_MISC_RESERVED_56	Misc: Reserved
3	224	BGP_MISC_RESERVED_57	Misc: Reserved
3	225	BGP_MISC_RESERVED_58	Misc: Reserved
3	226	BGP_MISC_RESERVED_59	Misc: Reserved
3	227	BGP_MISC_RESERVED_60	Misc: Reserved
3	228	BGP_MISC_RESERVED_61	Misc: Reserved
3	229	BGP_MISC_RESERVED_62	Misc: Reserved
3	230	BGP_MISC_RESERVED_63	Misc: Reserved
3	231	BGP_MISC_RESERVED_64	Misc: Reserved
3	232	BGP_MISC_RESERVED_65	Misc: Reserved
3	233	BGP_MISC_RESERVED_66	Misc: Reserved
3	234	BGP_MISC_RESERVED_67	Misc: Reserved
3	235	BGP_MISC_RESERVED_68	Misc: Reserved
3	236	BGP_MISC_RESERVED_69	Misc: Reserved

Group	Counter	Event name	Event description
3	237	BGP_MISC_RESERVED_70	Misc: Reserved
3	238	BGP_MISC_RESERVED_71	Misc: Reserved
3	239	BGP_MISC_RESERVED_72	Misc: Reserved
3	240	BGP_MISC_RESERVED_73	Misc: Reserved
3	241	BGP_MISC_RESERVED_74	Misc: Reserved
3	242	BGP_MISC_RESERVED_75	Misc: Reserved
3	243	BGP_MISC_RESERVED_76	Misc: Reserved
3	244	BGP_MISC_RESERVED_77	Misc: Reserved
3	245	BGP_MISC_RESERVED_78	Misc: Reserved
3	246	BGP_MISC_RESERVED_79	Misc: Reserved
3	247	BGP_MISC_RESERVED_80	Misc: Reserved
3	248	BGP_MISC_RESERVED_81	Misc: Reserved
3	249	BGP_MISC_RESERVED_82	Misc: Reserved
3	250	BGP_MISC_RESERVED_83	Misc: Reserved
3	251	BGP_MISC_RESERVED_84	Misc: Reserved
3	252	BGP_MISC_RESERVED_85	Misc: Reserved
3	253	BGP_MISC_RESERVED_86	Misc: Reserved
3	254	BGP_MISC_RESERVED_87	Misc: Reserved
3	255	BGP_MISC_ELAPSED_TIME_UM3	Misc: Elapsed time

### 3.3 Derived metrics

Some events are difficult to interpret. Sometimes a combination of events provide better information. In the sequel, such a recombination of basic events is called a *derived metric*.

Since each derived metric has its own set of ingredients, not all derived metrics are printed for each group. HPM automatically finds those derived metrics that are computable and prints them. As a convenience to the user, both the value of the derived metric and its definition are printed, if the environment variable HPM\_PRINT\_FORMULA is set.

### 3.4 Inheritance

*Counter virtualization* and the group (that is, the set of events) that is monitored are inherited from the process to any of the group's children, in particular threads that are spawned via OpenMP. However, there are differences among the various operating systems:

- ▶ On AIX, all counter values of a process group can be collected.
- ▶ On Linux and Blue Gene systems, counter values are available only to the parent, when the child has finished.

To use that concept, libhpm provides two types of start and stop functions:

- ▶ **hpmStart** and **hpmStop** start and stop counting on all processes and threads of a process group.
- ▶ **hpmTstart** and **hpmTstop** start and stop counting only for the thread from which they are called.

On Linux and Blue Gene systems, the **hpmStart** and **hpmStop** start and stop routines cannot be properly implemented, because the parent has no access to the counting environment of the child before this child has ended. Therefore the functionality of **hpmStart** and **hpmStop** is disabled on Linux and Blue Gene systems. The calls to **hpmStart** and **hpmStop** are folded into calls to **hpmTstart** and **hpmTstop**. As a result, they are identical and can be freely mixed on Linux and Blue Gene systems. However, we do not recommend mixing the routines because instrumentation like this would not port to AIX.

## 3.5 Inclusive and exclusive values

For a motivating example of the term *exclusive values*, refer to Example 3-1 on page 40. This program snippet provides an example of two properly nested instrumentation sections. For section 1, we can consider the exclusive time and exclusive counter values. By that, we mean the difference of the values for section 1 and section 2. The original values for section 1 are called *inclusive values* for matter of distinction. The terms *inclusive* and *exclusive* for the embracing instrumentation section are chosen to indicate whether counter values and times for the contained sections are included or excluded.

Of course the extra computation of exclusive values generates overhead, which is not always wanted. Therefore the computation of exclusive values is carried out only if the environment variable `HPM_EXCLUSIVE_VALUES` is set to 'Y[...]', 'y[...]', or '1'.

The exact definition of exclusive is based on parent-child relations among the instrumented sections. Roughly spoken, the exclusive value for the parent is derived from the inclusive value of the parent reduced by the inclusive value of all children.

Instrumented sections are not required to be properly nested, but can overlap in arbitrary fashion. Unfortunately, this overlapping destroys (or at least obscures) the natural parent-child relations among instrumented sections and complicates the definition of exclusive values.

### 3.5.1 Parent-child relations

The simplest way to establish parent child relations is to request the user to state them explicitly. New calls in the HPM API have been introduced to enable the user to establish the relations of choice. These functions are **hpmStartx** and **hpmTstartx** and their Fortran equivalents. The additional “x” in the function name can be interpreted as “extended” or “explicit”. The first two parameters of this function are the instrumented section ID and the ID of the parent instrumented section. The latter must exist. Otherwise HPM exits with an error message such as in the following example:

```
hpmcount ERROR - Illegal instance id specified
```

Not every user wants to undergo the hassle of explicitly building an ancestry tree among instrumented sections. Therefore HPM provides an automatic search for parents, which is supposed to closely mimic the behavior of properly nested instrumented sections. This automatic search is triggered by either specifying the value `HPM_AUTO_PARENT` to the second parameter of **hpmStartx** and **hpmTstartx**, or by using the classical start routines



**hpmStart** and **hpmTstart**. These two alternatives are equivalent. Indeed the second is implemented through the first alternative.

### 3.5.2 Handling overlap issues

Because the user can establish arbitrary parent child relations, the definition of the explicit duration or explicit counter values is far from obvious. Each instrumented section occupies a subset of the time line during program execution. This subset is a finite union of intervals with the left or lower boundaries marked by calls to **hpmStart**[x] or **hpmTstart**[x]. The right or upper boundaries are marked by calls to **hpmStop** or **hpmTstop**. The duration is the accumulated length of this union of intervals. The counter values are the number of those events that occur within this subset of time.

The main step in defining the meaning of exclusive values is to define the subset of the time line to which they are associated:

1. Represent the parent and every child by the corresponding subset of the time line (called the *parent set* and *child sets*).
2. Take the union of the child sets.
3. Reduce the parent set by the portion that is overlapping with this union.
4. Using set theoretic terms, take the difference of the parent set with the union of the child sets.

The exclusive duration is the accumulated length of the resulting union of intervals. The exclusive counter values are the number of those events that occur within this subset of time.

### 3.5.3 Computation of exclusive values for derived metrics

The task of computing exclusive values for derived metrics might sound complicated at first. It is simple, given the work done already in the previous subsections. The basic observation is that we are given a subset of the time line that is associated to the notion of *exclusive values*. How this set was constructed is not important. We assume that the interval boundaries are marked by calls to **hpmStart** and **hpmStop** for a new *virtual* instrumented section. In this case, it is obvious how to compute the derived metrics, which is to apply the usual definitions.

## 3.6 Function reference

The following instrumentation functions are provided:

- **hpmInit**( taskID, progName )  
  **f\_hpmInit**( taskID, progName )
  - taskID is an integer value that indicates the node ID. It is now depreciated. In an earlier version, this value indicated the node ID. It is no longer used and can be set to any value.
  - progName is a string with the program name. If the environment variable **HPM\_OUTPUT\_NAME** is not set, this string is used as a default value for the output name.

- ▶ `hpmStart( instID, label )`  
`f_hpmstart( instID, label )`
  - `instID` is the instrumented section ID. It should be  $> 0$  and  $\leq 1000$ .
  - To run a program with more than 1000 instrumented sections, the user should set the environment variable `HPM_NUM_INST_PTS`. In this case, `instID` should be less than the value set for `HPM_NUM_INST_PTS`.
  - `Label` is a string containing a label, which is displayed by PeekPerf.
- ▶ `hpmStartx( instID, par_ID, label )`  
`f_hpmstartx( instID, par_ID, label )`
  - `instID` is the instrumented section ID. It should be  $> 0$  and  $\leq 1000$ .
  - To run a program with more than 1000 instrumented sections, the user should set the environment variable `HPM_NUM_INST_PTS`. In this case, `instID` should be less than the value set for `HPM_NUM_INST_PTS`.
  - `par_ID` is the instrumentation ID of the parent section. See 3.5, “Inclusive and exclusive values” on page 84.
  - `Label` is a string that contains a label, which is displayed by PeekPerf.
- ▶ `hpmStop( instID )`  
`f_hpmstop( instID )`
  - For each call to **hpmStart**, there should be a corresponding call to **hpmStop** with a matching `instID`.
  - If not provided explicitly, an implicit call to **hpmStop** is made at **hpmTerminate**.
- ▶ `hpmTstart( instID, label )`  
`f_hpmtstart( instID, label )`  
`hpmTstartx( instID, par_ID, label )`  
`f_hpmtstartx( instID, par_ID, label )`  
`hpmTstop( instID )`  
`f_hpmtstop( instID )`

In order to instrument threaded applications, use the pair **hpmTstart** and **hpmTstop** to start and stop the counters independently on each thread. Notice that two distinct threads using the same `instID` generate an error. See 3.10, “Multithreaded program instrumentation issues” on page 89, for examples.
- ▶ `hpmGetTimeAndCounters( numCounters, time, values )`  
`f_GetTimeAndCounters ( numCounters, time, values )`  
`hpmGetCounters( values )`  
`f_hpmGetCounters ( values )`

These functions have been temporarily disabled in this release. They will be reintroduced in the next release.
- ▶ `hpmTerminate( taskID )`  
`f_hpmterminate( taskID )`
  - All active instrumented code sections receive an **hpmStop**.
  - This function generates the output.
  - If the program exits without calling `hpmTerminate`, no performance information is generated.

## 3.7 Measurement overhead

As in previous versions of HPM, the instrumentation overhead is caught by calls to the wall clock timer at entry and exit of calls to `hpmStart[x]`, `hpmStop`, `hpmTstart[x]`, and `hpmTstop`. The previous version tried to eliminate (or hide) the overhead from the measured results. The current version prints the timing of the accumulated overhead (separate for every instrumented section) in the ASCII output (\*.hpm file), so that the user can decide what to do with this information:

- ▶ If the overhead is several orders of magnitude smaller than the total duration of the instrumented section, you can safely ignore the overhead timing.
- ▶ If the overhead is in the same order as the total duration of the instrumented section, you should be suspicious of the results.
- ▶ If the overhead is within 20% of the measured wall clock time, a warning is printed to the ASCII output file.

## 3.8 Output

If no environment variable is specified, libhpm writes two files. These files contain (roughly) the same information, but use different formats:

- ▶ The file name can be specified via environment variable `HPM_OUTPUT_NAME=<name>`.
- ▶ If `HPM_OUTPUT_NAME` is not set, the string “progName” as specified in the second parameter to `hpmInit` is taken as the default. See 3.6, “Function reference” on page 85.
- ▶ The name <name> is expanded into three different file names:
  - `<name>.hpm` is the file name for ASCII output, which is a one-to-one copy of the screen output.
  - `<name>.viz` is the file name for XML output.
  - `<name>.csv` is the file name for output as a comma separated value (CSV) file. This is not implemented in the current release.
- ▶ Which of these output files is generated is governed by three additional environment variables. If none of the variables are set, the ASCII and the XML output is generated. If at least one variable is set, the following rules apply:
  - `HPM_ASC_OUTPUT` if set to 'Y[...]', 'y[...]' or '1' triggers the ASCII output.
  - `HPM_VIZ_OUTPUT` if set to 'Y[...]', 'y[...]' or '1' triggers the XML output.
  - `HPM_CSV_OUTPUT` if set to 'Y[...]', 'y[...]' or '1' triggers the CSV output. This is not implemented in the current release.
- ▶ The file name can be made unique by setting the environment variable `HPM_UNIQUE_FILE_NAME=1`. This triggers the following changes:
  - The following string is inserted before the last dot (.) in the file name:  
`_<hostname>_<process_id>_<date>_<time>`
  - If the file name has no dot, the string is appended to the file name.
  - If the only occurrence of dot is the first character of the file name, the string is prepended, but the leading dash (.) is skipped.
  - If the host name contains a dot (*long form*), only the portion preceding the first dot is taken. If a batch queuing system is used, the host name is taken from the execution host, not the submitting host.

- Similarly for MPI parallel programs, the host name is taken from the node where the MPI task is running. The addition of the process ID enforces different file names for MPI tasks running on the same node.
- If used for an MPI parallel program, **hpmcount** tries to extract the MPI task ID (or MPI rank with respect to MPI\_COMM\_WORLD) from the MPI environment. If successful, the process ID is replaced with the MPI task ID.
- The date is given as dd.mm.yyyy, and the time is given by hh.mm.ss in a 24-hour format using the local time zone.

## 3.9 Examples of libhpm for C and C++

Example 3-2 shows the syntax for C and C++, which are the same. The libhpm routines are declared as having external C linkage in C++.

*Example 3-2 C and C++ example*

---

```

declaration:
    #include "libhpm.h"
use:
    hpmInit( tasked, "my program" );
    hpmStart( 1, "outer call" );
    do_work();
    hpmStart( 2, "computing meaning of life" );
    do_more_work();
    hpmStop( 2 );
    hpmStop( 1 );
    hpmTerminate( taskID );

```

---

Fortran programs (shown in Example 3-3) should call the functions with the prefix *f\_*. Also, in Example 3-3, notice that the declaration is required on all source files that have instrumentation calls.

*Example 3-3 Fortran example*

---

```

declaration:
    #include "f_hpm.h"
use:
    call f_hpminit( taskID, "my program" )
    call f_hpmstart( 1, "Do Loop" )
    do ...
        call do_work()
        call f_hpmstart( 5, "computing meaning of life" );
        call do_more_work();
        call f_hpmstop( 5 );
    end do
    call f_hpmstop( 1 )
    call f_hpmterminate( taskID )

```

---

## 3.10 Multithreaded program instrumentation issues

When placing instrumentation inside of parallel regions, use different ID numbers for each thread, as shown in Example 3-4 for Fortran.

*Example 3-4 Multithreaded program*

---

```
!$OMP PARALLEL
!$OMP&PRIVATE (instID)
    instID = 30+omp_get_thread_num()
    call f_hpmTstart( instID, "computing meaning of life" )
!$OMP DO
    do ...
        do_work()
    end do
    call f_hpmTstop( instID )
!$OMP END PARALLEL
```

---

If two threads use the same ID numbers for call to **hpmTstart** or **hpmTstop**, libhpm exits with the following error message:

hpmcount ERROR - Instance ID on wrong thread

## 3.11 Considerations for MPI parallel programs

Libhpm is inherently sequential, looking only at the hardware performance counters of a single process (and its children, as explained in 3.4, “Inheritance” on page 83). When started with **poe** or **mpi run**, each MPI task does its own hardware performance counting and these instances are completely ignorant of each other, unless additional action is taken as described in the following sections. Consequently, each instance writes its own output. If the environment variable **HPM\_OUTPUT\_NAME** is used, each instance uses the same file name, which results in writing into the same file, if a parallel file system is used. Of course, this can be (and should be) prevented by making the file names unique through the **HPM\_UNIQUE\_FILE\_NAME** environment variable. Still it might be an unwanted side effect to have that many output files.

For this reason, the environment variable **HPM\_AGGREGATE** does aggregation before (possibly) restricting the output to a subset of MPI tasks. This formulation is deliberately vague, because there can be many ways to aggregate hardware performance counter information across MPI tasks. One way is to take averages, but maximum or minimum values can also be considered. The situation is further complicated by running different groups on different MPI tasks. Take averages and maximum and minimum values only on groups that are alike.

Therefore, the environment variable **HPM\_AGGREGATE** takes a value, which is the name of a plug-in that defines the aggregation strategy. Each plug-in is a shared object file that contain two functions called *distributor* and *aggregator*.

On the Blue Gene/L system, there are no shared objects. Therefore the plug-ins are simple object files. The **HPM\_AGGREGATE** environment variable is not used on the Blue Gene/L system, but the plug-ins are statically linked with the library. On the Blue Gene/P system, you can choose to do it either way.

### 3.11.1 Distributors

The motivating example for the distributor function allows a different hardware counter group on each MPI task. Therefore, the *distributor* is a subroutine that determines the MPI task ID (or MPI rank with respect to MPI\_COMM\_WORLD) from the MPI environment for the current process, and sets or resets environment variables depending on this information. The environment variable can be any environment variable, not just HPM\_EVENT\_SET, which motivated this function.

Consequently, the distributor is called before any environment variable is evaluated by HPM. Even if an environment variable is evaluated prior to the call of the distributor, it is re-evaluated afterwards.

The aggregator must adapt to the HPM group settings done by the distributor. This is why distributors and aggregators always come in pairs. Each plug-in contains one such pair.

### 3.11.2 Aggregators

The motivating example for the aggregator function is the aggregation of the hardware counter data across the MPI tasks. In the simplest case, this can be an average of the corresponding values. Hence this function is called at the following times:

- ▶ After the hardware counter data is gathered
- ▶ Before the data is printed
- ▶ Before the derived metrics are computed

In a generalized view, the aggregator takes the raw results and rearranges them for output. Also, depending on the information of the MPI task ID (or MPI rank with respect to MPI\_COMM\_WORLD) the aggregator sets, or does not set, a flag to mark the current MPI task for HPM printing.

### 3.11.3 Plug-ins shipped with HPCT

The following plug-ins are shipped with the toolkit. You can find them in \$(IHPCT\_BASE)/lib or \$(IHPCT\_BASE)/lib64.

- ▶ *mirror.so* is the plug-in that is called when no plug-in is requested. The aggregator mirrors the raw hardware counter data in a one-to-one fashion into the output function, hence the name. It also flags each MPI task as a printing task. The corresponding distributor is a void function.
- ▶ *loc merge.so* does a local merge on each MPI task separately. It is identical to the mirror.so plug-in except for those MPI tasks that change the hardware counter groups in the course of the measurement.

The different counter data, which is collected for only part of the measuring interval, is proportionally extended to the whole interval and joined into one big group that enters derived metrics computation. This way, more derived metrics can be determined at the risk of computing garbage. The user is responsible for using this plug-in only when it makes sense to use it. It also flags each MPI task as a printing task. The corresponding distributor is a void function.

- ▶ *single.so* does the same as mirror.so, but only on MPI task 0. The output on all other tasks is discarded.
- ▶ *average.so* is a plug-in for taking averages across MPI tasks. The distributor is reading the environment variable HPM\_EVENT\_DISTR, which is supposed to be a comma separated list of group numbers, and distributes these group numbers in a round-robin fashion to the MPI

tasks. The aggregator first builds an MPI communicator of all tasks with an equal hardware performance counting scenario. The communicator groups might be different from the original round-robin distribution, because the scenarios are considered incomparable:

- If the counting group has been changed during execution.
- If the corresponding timing differs by more than 2 seconds from the average.

Next the aggregator takes the average across the subgroups formed by this communicator. Finally it flags the MPI rank 0 in each group as a printing host.

### 3.11.4 User-defined plug-ins

This set of plug-ins is only a starter kit and many more might be desirable. Rather than taking the average, you can think of taking a maximum or minimum. There is also the possibility of taking a *history\_merge.so* by blending in results from previous measurements. Chances are that however big the list of shipped plug-ins might be, the one that is needed is missing from the set (“Murphy’s law of HPM plug-ins”). The only viable solution comes with disclosing the interface between a plug-in and tool and allowing for user defined plug-ins.

The easiest way to enable users to write their own plug-ins is by providing examples. Hence the plug-ins described previously are provided in source code together with the makefile that was used to generate the shared objects files. These files can be found in the `$(IHPCT_BASE)/examples/plugins` directory.

### 3.11.5 Detailed interface description

Each distributor and aggregator is a function that returns an integer that is 0 on success and  $\neq 0$  on error. In most cases, the errors occur when calling a system call such as `malloc()`, which sets the `errno` variable. If the distributor or aggregator returns the value of `errno` as a return code, the calling HPM tool sees an expansion of this `errno` code into a readable error message. If returning the `errno` is not viable, the function returns a negative value.

The function prototypes are defined in the `$(IHPCT_BASE)/include/hpm_agg.h` file. This is a short file with the following contents:

```
#include "hpm_data.h"
int distributor(void);
int aggregator(int num_in, hpm_event_vector in,
int *num_out, hpm_event_vector *out,
int *is_print_task);
```

The distributor has no parameters and is required to set or reset environment variables, via `setenv()`.

The aggregator takes the current hpm values on each task as an input vector `in` and returns the aggregated values on the output vector `out` on selected or all MPI tasks. For utmost flexibility, the aggregator is responsible for allocating the memory that is needed to hold the output vector `out`. The definition of the data types used for `in` and `out` are provided in the `$(IHPCT_BASE)/include/hpm_data.h` file.

Finally the aggregator is supposed to set (or unset) a flag to mark the current MPI task for HPM printing.

From the previous definitions, it is apparent that the interface is defined in the C language. While in principle it possible to use another language for programming plug-ins, the user is

responsible for using the same memory layout for the input and output variables. No explicit Fortran interface is provided.

The `hpm_event_vector` in is a vector or list of `num_in` entries of type `hpm_data_item`. The latter is a struct that contains members that describe the definition and the results of a single hardware performance counting task.

Example 3-5 describes the types of parameters that are used in a call to a function aggregator.

*Example 3-5 Definition of `hpm_event_vector`*

---

```
#define HPM_NTIM 7
#define HPM_TIME_WALLCLOCK      0
#define HPM_TIME_CYCLE          1
#define HPM_TIME_USER           2
#define HPM_TIME_SYSTEM         3
#define HPM_TIME_START          4
#define HPM_TIME_STOP           5
#define HPM_TIME_OVERHEAD       6

typedef struct {
    int          num_data;
    hpm_event_info *data;
    double       times[HPM_NTIM];
    int          is_mplex_cont;
    int          is_rusage;
    int          mpi_task_id;
    int          instr_id;
    int          is_exclusive;
    char         *description;
    char         *xml_descr;
} hpm_data_item;

typedef hpm_data_item *hpm_event_vector;
```

---

Counting the events from a certain HPM group on one MPI task is represented by a single element of type hpm data item.

If several instrumented sections are used, each instrumented code section uses separate elements of type hpm data item to record the results. Each of element has the member `instr_id` set with the first argument of **hpmStart**, and the logical member `is_exclusive` set to `TRUE_` or `FALSE_` depending on whether the element holds inclusive or exclusive counter results. See 3.5, “Inclusive and exclusive values” on page 84, for details. Then all of these different elements are concatenated into a single vector.

Finally, the data from a call to `getrusage()` is prepended to this vector. The `rusage` data forms the vector element with index 0. This vector element is the only element with struct member `is_rusage` set to `TRUE_` to distinguish it from ordinary hardware performance counter data.

The output vector is of the same format. Each vector element enters the derived metrics computation separately (unless `is_rusage == TRUE_`). Then all vector elements (and the corresponding derived metrics) are printed in the order given by the vector out. The output of each vector element is preceded by the string given in a member description, which can include line feeds as appropriate. The XML output is marked with the text given in `xml_descr`. This way, the input vector `in` provides a complete picture of what was measured on each MPI



task. The output vector `out` allows complete control of what is printed on which MPI task in what order.

### 3.11.6 Getting the plug-ins to work

The plug-ins have been compiled with the following makefile:

```
$(IHPCT_BASE)/examples/plugins/Makefile
```

This compilation occurs by using the following command:

```
<g>make ARCH=<appropriate_archtitecture>
```

The include files for the various architectures are provided in the `make` subdirectory. Note the following subtleties:

- ▶ The makefile distinguishes “sequential” (specified in `PLUGIN_SRC`) and “parallel” plug-ins (specified in `PLUGIN_PAR_SRC`). The latter plug-ins are compiled and linked with the MPI wrapper script for the compiler or linker. Unlike a static library, generation of a shared object requires linking, not just compilation.
- ▶ On the Blue Gene/L system, there are no shared objects. Therefore, ordinary object files are generated. On the Blue Gene/L and Blue Gene/P systems, everything is parallel.
- ▶ Restrictions are observed when writing plug-in code. The MPI standard document disallows calling `MPI_Init()` twice on the same process. It appears that this is not supported on the majority of MPI software stacks, not even if an `MPI_Finalize()` is called between the two invocations of `MPI_Init()`.
- ▶ The distributor is called by `hpmInit()`. If it contains MPI calls, this enforces the distributor to have `MPI_Init()` prior to `hpmInit()`. To lift this restriction, the distributor must not call any MPI function. The MPI task ID should be extracted by inspecting environment variables that have been set by the MPI software stack.
- ▶ The aggregator usually cannot avoid calling MPI functions. Before calling `MPI_Init()`, it must check whether the instrumented application has already done so. If the instrumented application is an MPI application, it cannot be called after `MPI_Finalize()`. The aggregator is called by `hpmTerminate()`. Therefore, `hpmTerminate()` must be called between the calls to `MPI_Init()` and `MPI_Finalize()`.
- ▶ `libhpm` uses a call to `dlopen()` to access the plug-in and uses its functions. There is no `dlopen()` on the Blue Gene/L system. Plug-ins are statically linked to the application. On the Blue Gene/P system, both ways to access the plug-ins can be used.





# High Performance Computing Toolkit GUI

The High Performance Computing Toolkit (HPCT) graphical user interface (GUI) is the visual control center of HPCT. With this GUI, you can control instrumentation, execute the application, and visualize and analyze the collected performance data within the same user interface. The following dimensions of performance data are provided in our current framework:

- ▶ CPU (Hardware Performance Monitoring (HPM))
- ▶ Message Passing Interface (MPI)
- ▶ Threads (OpenMP)
- ▶ Memory
- ▶ I/O

The collected performance data is mapped to the source code, so that you can more easily find bottlenecks and points for optimizations. The HPCT GUI provides filtering and sorting capabilities to help you analyze the data.

## 4.1 Starting the HPCT GUI

You start the HPCT GUI from a command line by using either of the following commands:

```
peekperf  
peekperf <-num max_src_files> <vizfiles>
```

You can specify more than one .viz file. The HPCT GUI opens all the .viz files and combines the data from all of them. The HPCT GUI also tries to open the source files if the source files are available. In some applications, there might be hundreds of source files. By default, the HPCT GUI opens up to fifteen source files. If there are more than fifteen source files, the HPCT GUI prompts you for input to select a list of the files to be opened. You can reset the default value by using the -num option.

The following syntax is for binary instrumentation:

```
peekperf <-num max_src_files> <binary> <vizfiles>
```

The HPCT GUI invokes the binary instrumentation engine and obtains information about the binary. Then from the GUI, you can control the instrumentation.

## 4.2 HPCT GUI Main window (Visualization)

As mentioned previously, the HPCT GUI tries to find your source files. If it fails to locate the files, a window prompts you to select the top-level directory for your source code. If the HPCT GUI finds more than one file with the same name, you are prompted to select the correct file. You can also open the files manually by selecting **File** → **Open Sources**. The .viz files can be opened by selecting **File** → **Open Performance Data**.

Figure 4-1 shows the performance data visualization interface of the HPCT GUI. In this mode, two windows are open. The Data Visualization Window, on the left, contains the collected performance data. The Source Code Window, on the right, displays the source file.

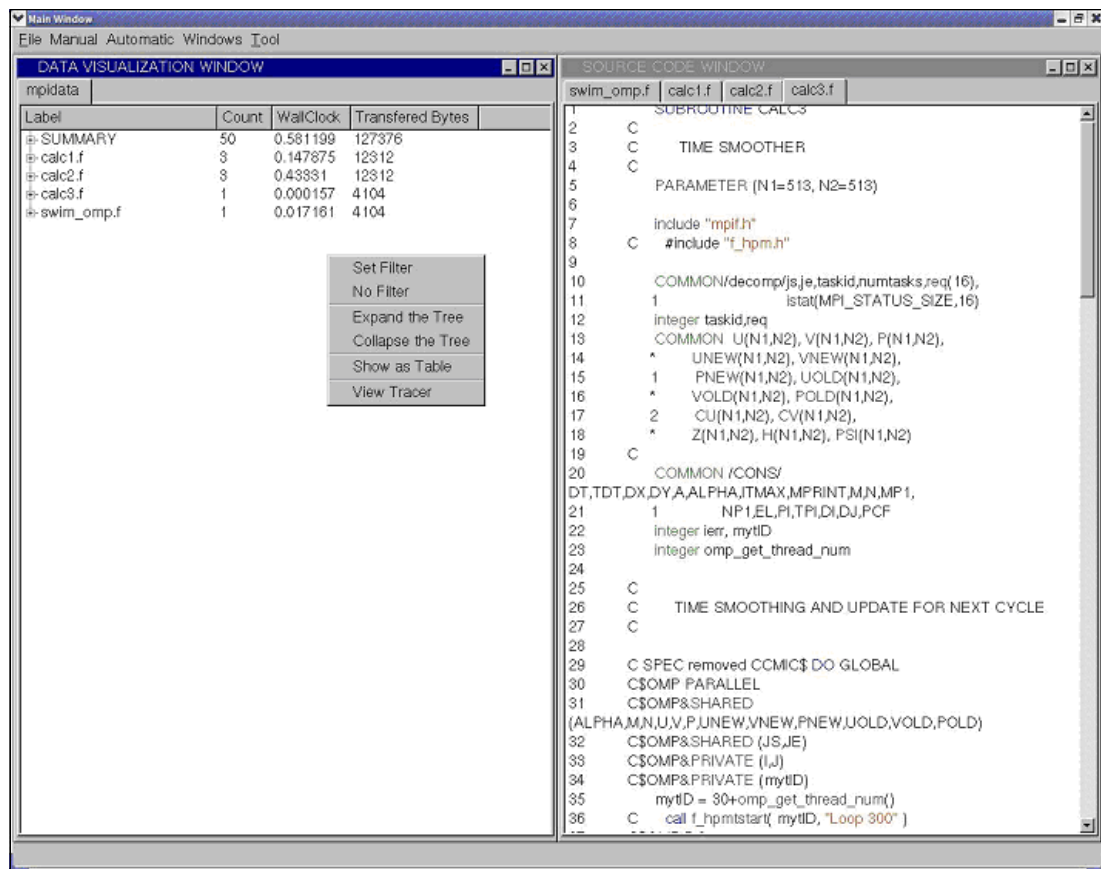


Figure 4-1 HPCT GUI

In the Data Visualization Window, the data is presented in a hierarchical tree format. Clicking the plus sign (+) expands that specific section of the tree. Clicking the minus sign (-) collapses the section of the tree.

After you expand any of the sections, you can click one of the leaf nodes and the corresponding line of source code is highlighted in the Source Code Window (right pane). If you right-click a leaf node, a window opens (Figure 4-2) that contains all of the performance data collected in more detail.

**Leaf node:** The term *leaf node* refers to the lowest level of the tree. For MPI, the leaf node is an MPI function. An HPM leaf node refers to a function or a user-defined region.

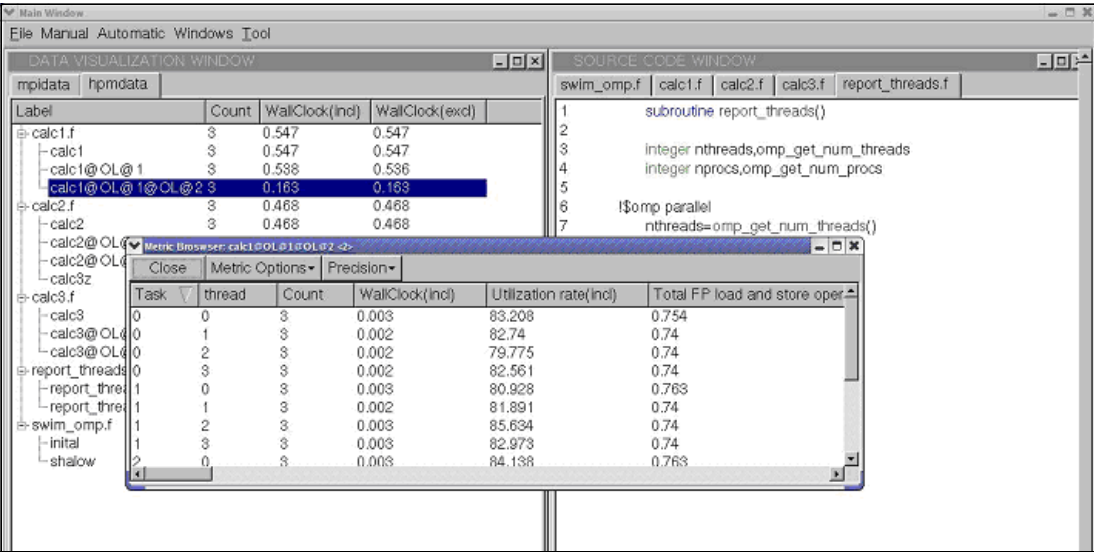


Figure 4-2 All performance data for specified node

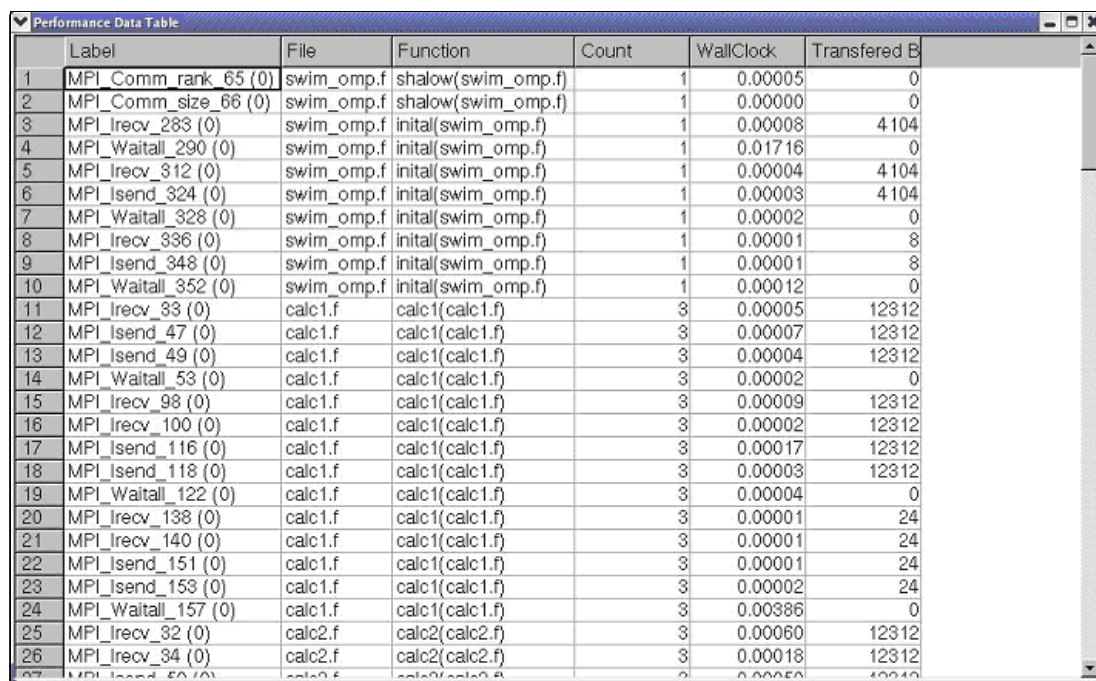
If you right-click a non-leaf node or other empty space, a context menu opens as shown in Figure 4-3 (also shown in Figure 4-1 on page 97). You can collapse or expand the tree by selecting *Collapse the Tree* or *Expand the Tree*. You can filter the performance data by selecting *Set Filter*. If you want to go back to the original state, you can select *No Filter*. In addition to the tree view of performance data, you can also display the performance data as a tabular form by selecting the *Show as Table*. If the collected performance data is MPI, you see *View Tracer* in the context menu. By selecting *View Tracer*, the Trace Viewer opens. However, if the collected performance data is I/O, the *View Tracer* option opens a different viewer for the I/O trace data.



Figure 4-3 Context menu

## Show as Table

Figure 4-4 shows the tabular form of performance data that is returned when you select *Show as Table* in the context menu. You can sort each column by clicking the header of each column. Right-click to open a context menu that has options with which you can hide the column or save the data into a plain text file. If you want to move a column, press the Ctrl key, click the column header, and drag the column to the new location. You can also map the data to the source code by clicking the row number on the left.



	Label	File	Function	Count	WallClock	Transferred B
1	MPI_Comm_rank_65 (0)	swim_omp.f	shalow(swim_omp.f)	1	0.00005	0
2	MPI_Comm_size_66 (0)	swim_omp.f	shalow(swim_omp.f)	1	0.00000	0
3	MPI_Irecv_283 (0)	swim_omp.f	initail(swim_omp.f)	1	0.00008	4104
4	MPI_Waitall_290 (0)	swim_omp.f	initail(swim_omp.f)	1	0.01716	0
5	MPI_Irecv_312 (0)	swim_omp.f	initail(swim_omp.f)	1	0.00004	4104
6	MPI_Isend_324 (0)	swim_omp.f	initail(swim_omp.f)	1	0.00003	4104
7	MPI_Waitall_328 (0)	swim_omp.f	initail(swim_omp.f)	1	0.00002	0
8	MPI_Irecv_336 (0)	swim_omp.f	initail(swim_omp.f)	1	0.00001	8
9	MPI_Isend_348 (0)	swim_omp.f	initail(swim_omp.f)	1	0.00001	8
10	MPI_Waitall_352 (0)	swim_omp.f	initail(swim_omp.f)	1	0.00012	0
11	MPI_Irecv_33 (0)	calc1.f	calc1(calc1.f)	3	0.00005	12312
12	MPI_Isend_47 (0)	calc1.f	calc1(calc1.f)	3	0.00007	12312
13	MPI_Isend_49 (0)	calc1.f	calc1(calc1.f)	3	0.00004	12312
14	MPI_Waitall_53 (0)	calc1.f	calc1(calc1.f)	3	0.00002	0
15	MPI_Irecv_98 (0)	calc1.f	calc1(calc1.f)	3	0.00009	12312
16	MPI_Irecv_100 (0)	calc1.f	calc1(calc1.f)	3	0.00002	12312
17	MPI_Isend_116 (0)	calc1.f	calc1(calc1.f)	3	0.00017	12312
18	MPI_Isend_118 (0)	calc1.f	calc1(calc1.f)	3	0.00003	12312
19	MPI_Waitall_122 (0)	calc1.f	calc1(calc1.f)	3	0.00004	0
20	MPI_Irecv_138 (0)	calc1.f	calc1(calc1.f)	3	0.00001	24
21	MPI_Irecv_140 (0)	calc1.f	calc1(calc1.f)	3	0.00001	24
22	MPI_Isend_151 (0)	calc1.f	calc1(calc1.f)	3	0.00001	24
23	MPI_Isend_153 (0)	calc1.f	calc1(calc1.f)	3	0.00002	24
24	MPI_Waitall_157 (0)	calc1.f	calc1(calc1.f)	3	0.00386	0
25	MPI_Irecv_32 (0)	calc2.f	calc2(calc2.f)	3	0.00060	12312
26	MPI_Irecv_34 (0)	calc2.f	calc2(calc2.f)	3	0.00018	12312
27	MPI_Isend_50 (0)	calc2.f	calc2(calc2.f)	3	0.00050	12312

Figure 4-4 Show as Table window

## View Tracer

The MPI Trace Viewer window (Figure 4-5) shows the tasks at the y axis and the time at the x axis. For every task, you can see the timeline. Every MPI call is highlighted with another color. The MPI traces can be viewed with a black or a bright background. When using the bright background, every event is surrounded by a black rectangle that makes it easier to identify short events.

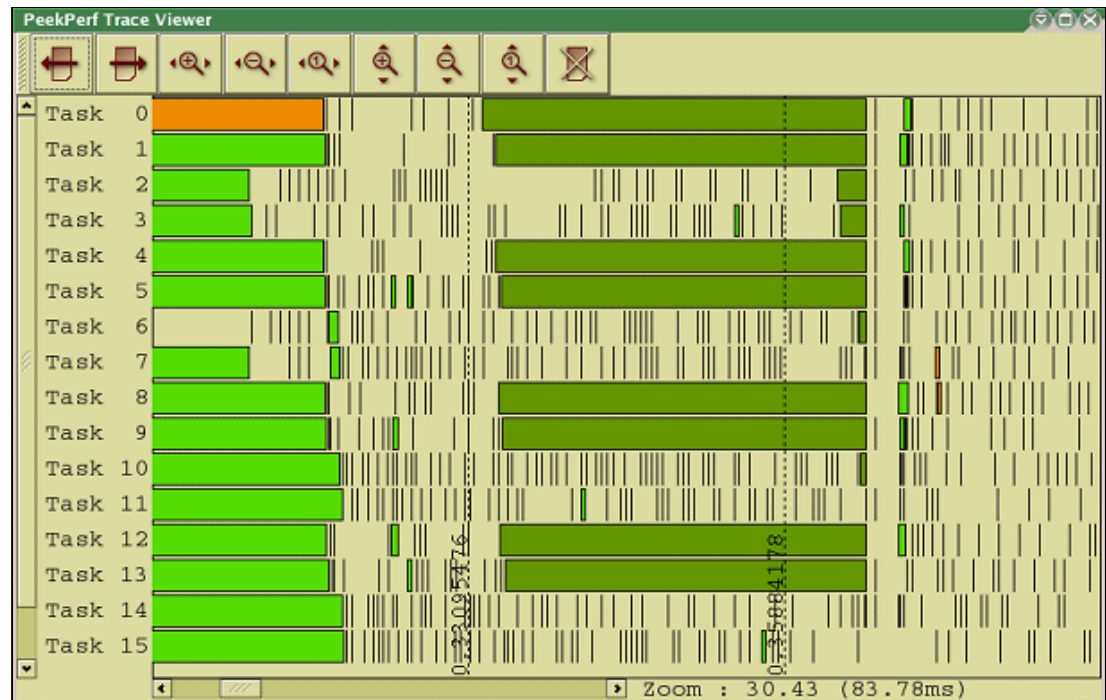


Figure 4-5 MPI Trace Viewer



Figure 4-6 shows the same trace with a different zoom level.

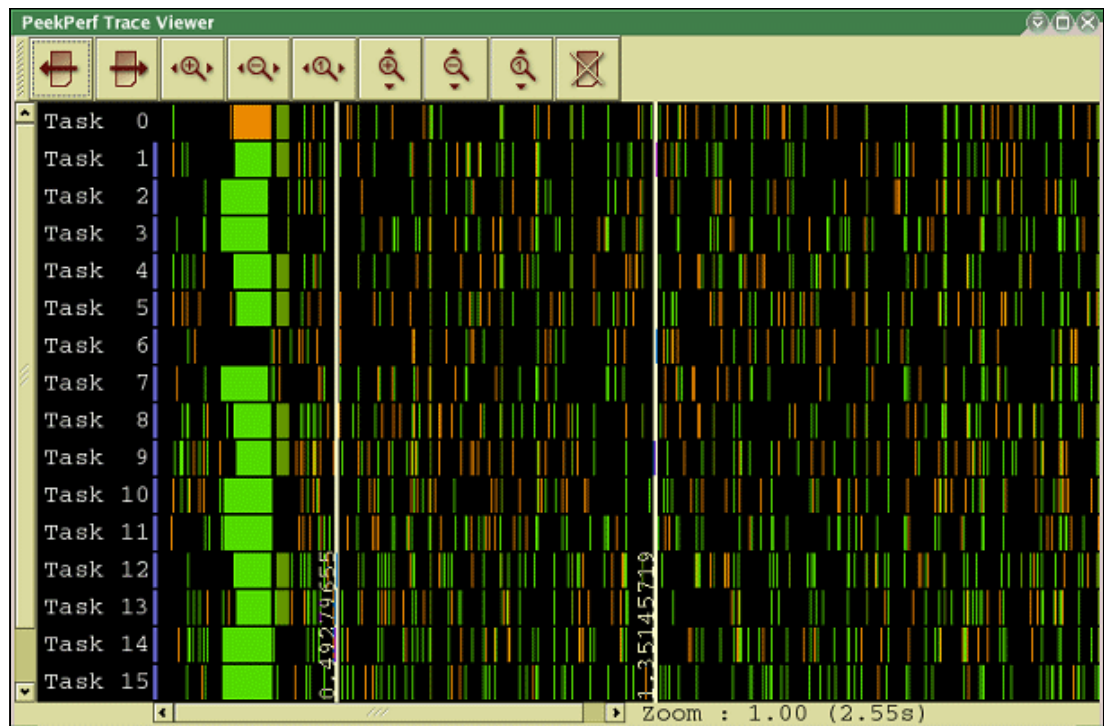


Figure 4-6 Trace Viewer with a new zoom level

Figure 4-7 shows the Identifier window, which is shown beside the Trace Viewer. You use the legend to easily map from the timeline to an event within the timeline. Additionally you can suppress some types of events from being displayed by clicking the event type in the Identifier window and deselecting the event.

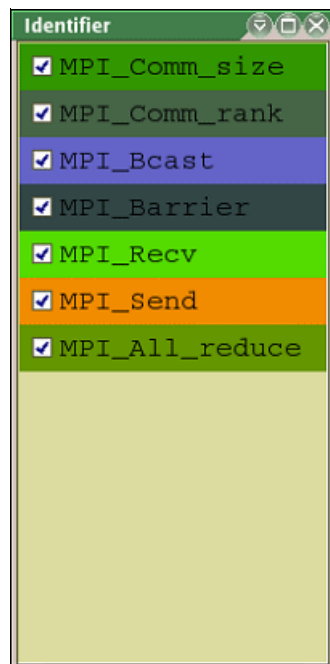


Figure 4-7 Trace Viewer Identifier window

You can click an event in the Trace Viewer window to highlight the corresponding line of source in the Main Window of the HPCT GUI. If you hold down the left mouse button and move the mouse pointer to another point, you see two lines. The first line is shown at the origin (where you clicked the mouse button). The second line is shown at the current position of the mouse. If you release the mouse button, the timeline zooms automatically into this selected timeframe.

When you right-click an event in the Trace Viewer window, a box opens as shown in Figure 4-8. This window represents a summary of the collected data for the selected event.

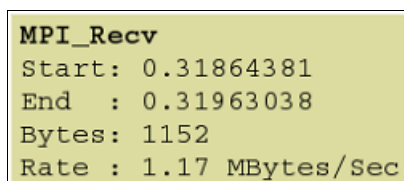


Figure 4-8 Summary of data collected

**Transfer rate:** Whenever an event transmits bytes, we try to calculate a transfer rate by dividing the number of transferred bytes by the elapsed time. In case of non-blocking calls, for example `MPI_Irecv()`, this data does not show the associated physical transfer rate.

You can navigate around the Trace Viewer by using the toolbar at the top of the viewer (Figure 4-9). This toolbar can be undocked from the window. You can zoom in and out vertically and horizontally.



Figure 4-9 Navigation toolbar

You can also use your keyboard to navigate through the trace. Table 4-1 provides a list of keys and their corresponding action.

Table 4-1 Navigation keys

Key	Action
Left arrow	Move trace to the left
Right arrow	Move trace to the right
Up arrow	Scroll up through tasks
Down arrow	Scroll down through tasks
Page up	Scroll trace faster to the left
Page down	Scroll trace faster to the right
z or y	Zoom time in
x	Zoom time out
a	Zoom tasks in
s	Zoom tasks out

If you are displaying an I/O trace, a window similar to the one shown in Figure 4-10 opens. The I/O Trace Viewer is the window in the right side of the figure.

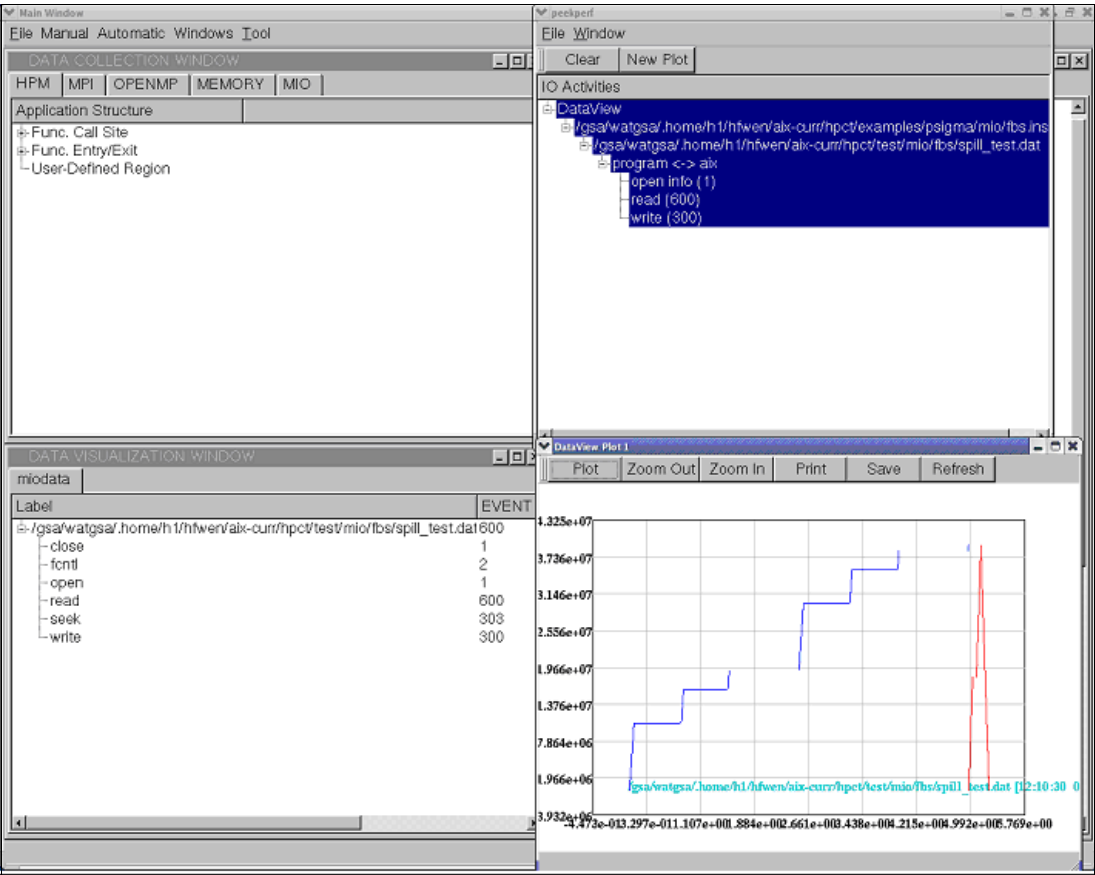


Figure 4-10 I/O Trace Viewer

## 4.3 HPCT GUI Main Window with instrumentation

When the binary is given to the HPCT GUI, the Data Collection Window (Figure 4-11) opens. In this window, you are able to control the instrumentation. The binary can be given via the command line or by selecting **File → Open Binary**.

The tree in each panel presents the program structure and can be created based on the type of performance data. For example, click the **HPM** tab. The tree in the HPM panel contains two subtrees. The *Func. Entry/Exit* subtree shows all the functions. The call site of the functions is in another *Func. Call Site* subtree.

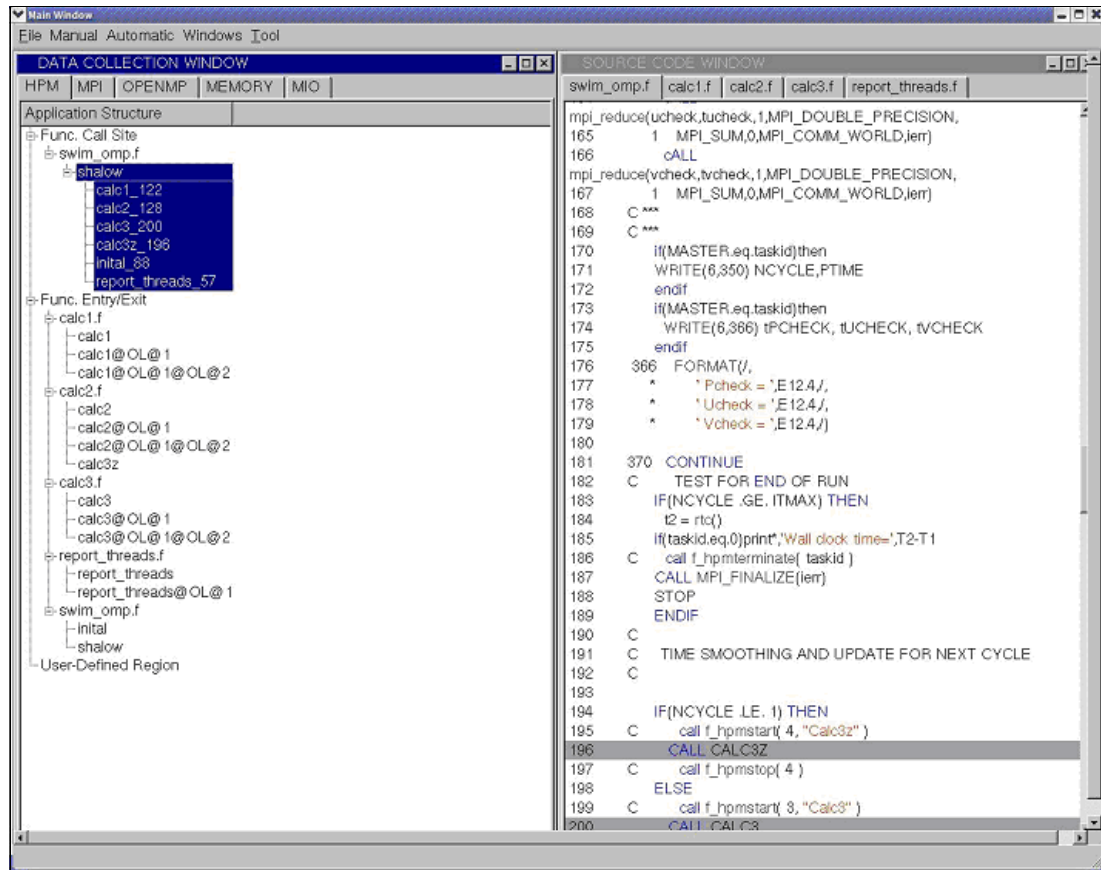


Figure 4-11 Main window With instrumentation

The context menu of Data Collection Window provides searching capabilities in the tree. Right-click a blank portion of the panel and select the **Search** option. Then enter a keyword. The HPCT GUI searches the entire tree and gives information related to the keyword. For example, Figure 4-12 shows the results of the keyword *calc1*.

Search Results for calc1					
	Path	File	Function	Call Site	
1	/gsa/watgsa/.home	calc1.f	calc1		
2	/gsa/watgsa/.home	calc1.f	calc1@OL@1		
3	/gsa/watgsa/.home	calc1.f	calc1@OL@1@OL@2		
4	/gsa/watgsa/.home	swim_omp.f	shalow	calc1_122	

Figure 4-12 Search results

The *instrumentation* is selected by clicking the nodes in the tree. For example, when you click the *shallow* node, all the children in the shallow node are selected and highlighted (see Figure 4-11). If you want to deselect it, click the selected node again. All the children including this node are then deselected. If you want to clear all your currently selected instrumentation, select **Tool** → **Clear Instrumentation**.

At this point, the instrumented application is not generated yet. You are only selecting the places to put the instrumentation. After you browse each panel and decide which instrumentation to use, you select **Automatic** → **Generate an Instrumented Application**. When the application is done, a window opens that indicates if the instrumentation is completed. It also indicated the name of the instrumented application.

After the instrumented application is generated, you can run the instrumented application if the application can be run in the same machine. You can do this by selecting **Automatic** → **Run an Instrumented Application**. Figure 4-13 shows an example after the instrumented application has run. The HPM and MPI data are collected in a single run.

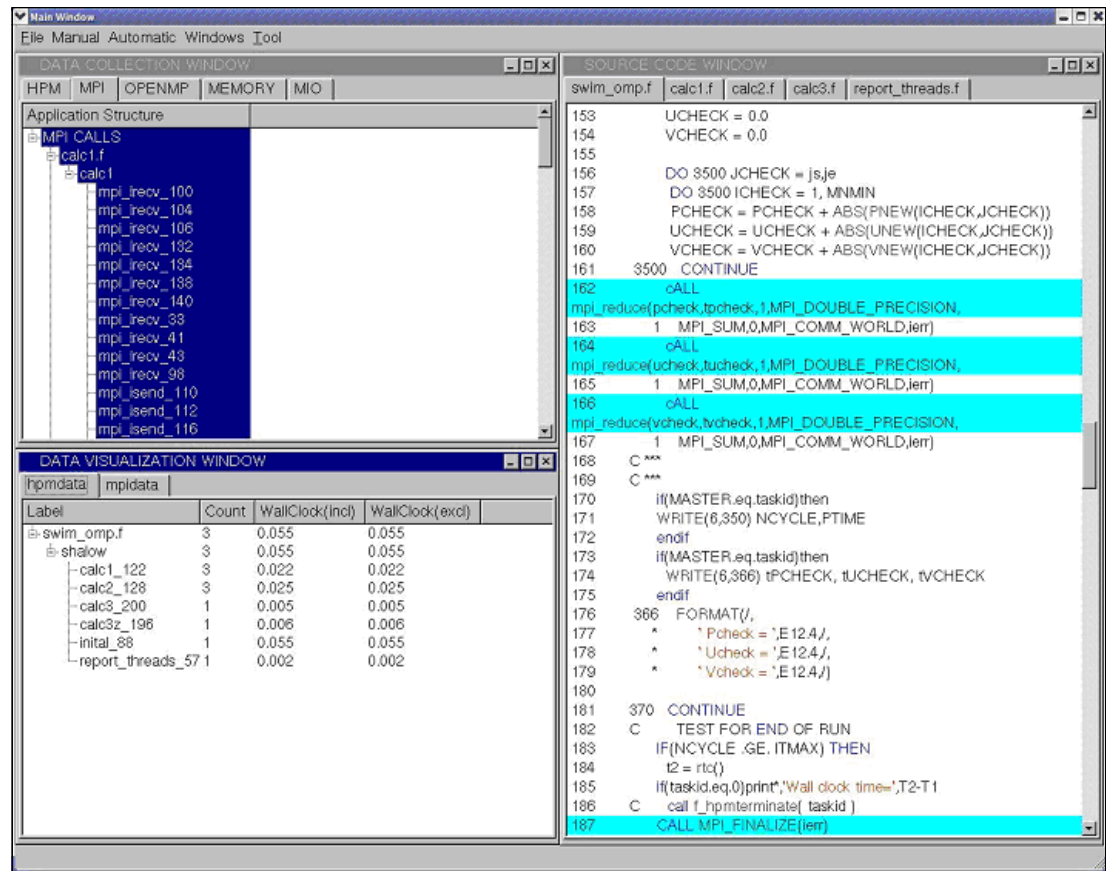


Figure 4-13 Instrumented application

For MPI, the tree is displayed in two different ways. One way is to group by MPI function classes (see Figure 4-14).

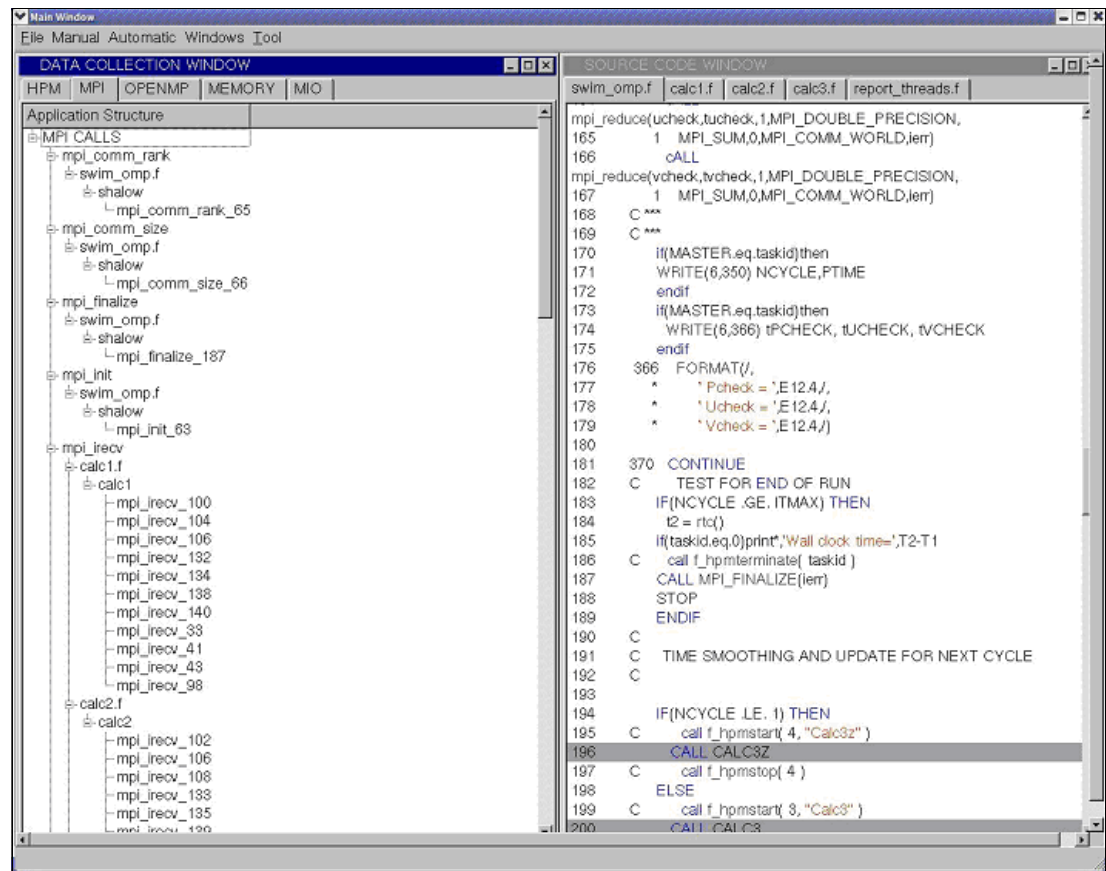


Figure 4-14 Grouped by MPI functions

The other way is to group by File and Function (see Figure 4-15). The display for MPI can be switched by the option in the context menu of the MPI panel.

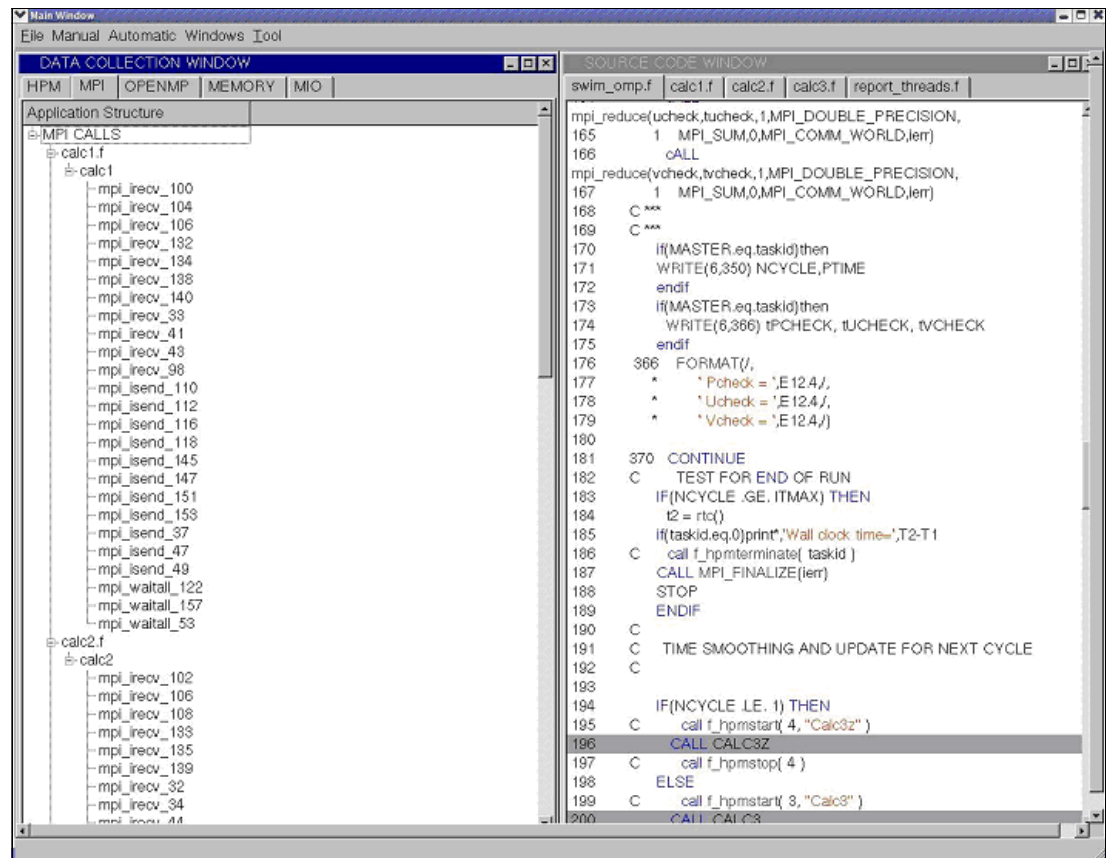


Figure 4-15 Grouped by file or function



You can set up environment variables by selecting **Automatic** → **Set the Environment Variable**. For HPM, if you want to change the event group, you can open the context menu in the HPM panel. Right-click in an open area of the HPM Counter Groups and select the **Set the Counter Group** option.

In the window that opens (Figure 4-16), you can change the event group. You can view the counter group information by clicking the Help button, which shows the counter information in the current platform.

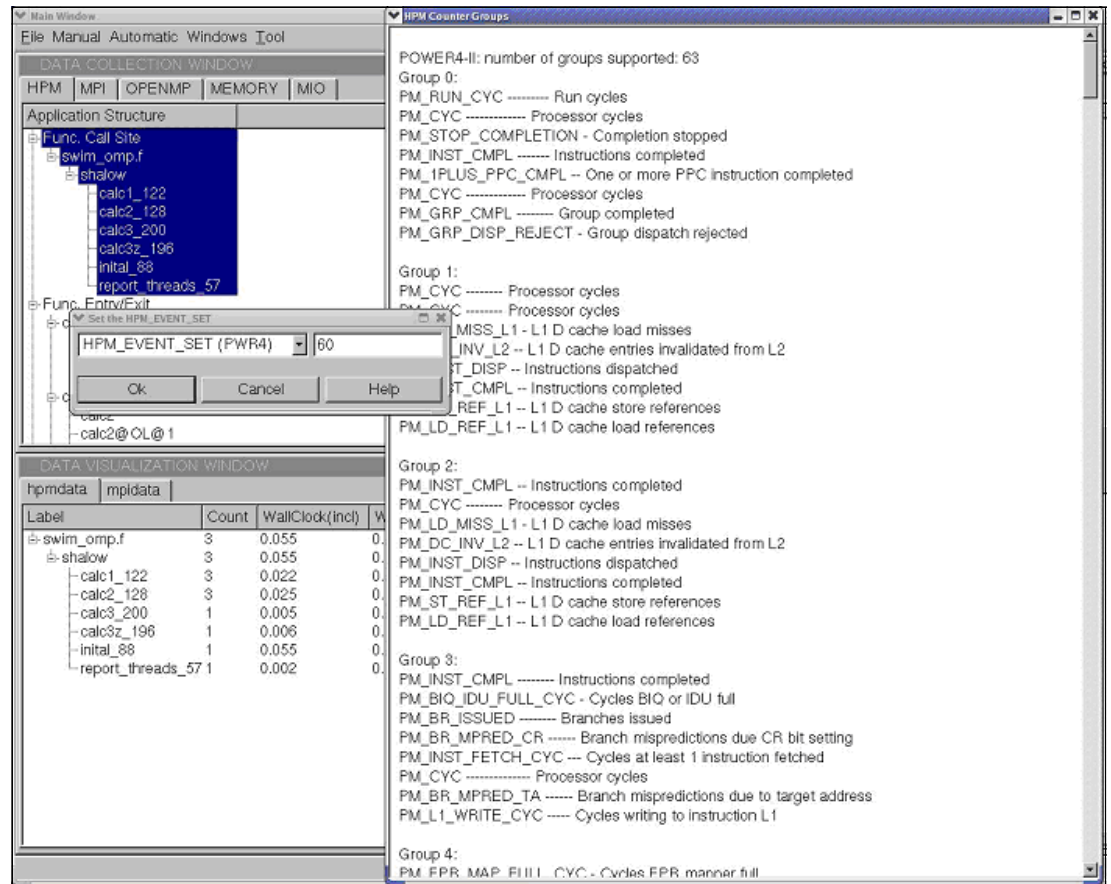


Figure 4-16 Setting the HPM\_EVENT\_SET



## 4.4 HPCT GUI simple IDE

In the Source Code Window, you can right-click and either open and edit the file using the vi editor. After the file is changed and saved, you can reload it back to the Source Code Window. The HPCT GUI detects whether the change occurred and prompts you if you want to reload it. You can also edit any file by selecting **Manual** → **Edit**. If you want to compile your application without switching to another terminal, you can do so by selecting **Manual** → **Compile**. A window opens in which you can supply a command. You can run any executable by selecting **Manual** → **Run**.

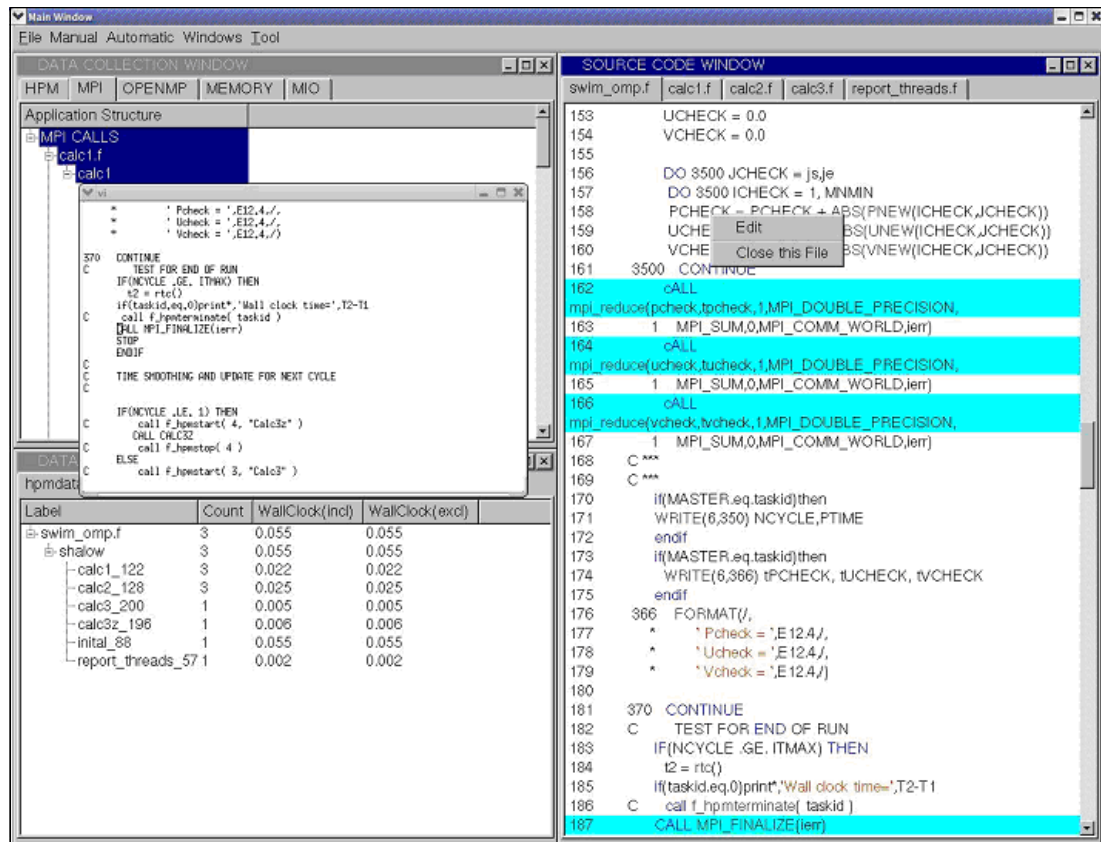


Figure 4-17 Editing the source code





## I/O performance

The Modular I/O (MIO) library was developed by the Advanced Computing Technology Center (ACTC) of the IBM Thomas J. Watson Research Center to address the need for an applications-level method for optimizing I/O. Applications frequently have little logic built into them to provide users the opportunity to optimize the I/O performance of the application. The absence of application-level I/O tuning leaves the user at the mercy of the operating system to provide the tuning mechanisms for I/O performance. Typically, multiple applications are run on a given system that have conflicting needs for high performance I/O, resulting at best a set of tuning parameters that provide moderate performance for the application mix.

The MIO library allows users to analyze the I/O of their application and then tune the I/O at the application level for a more optimal performance for the configuration of the current operating system.

A common I/O pattern exhibited is the sequential reading of large files (tens of gigabytes). Applications that exhibit this I/O pattern tend to benefit minimally from operating system buffer caches. Large operating system buffer pools are ineffective since there is little, if any, data reuse, and system buffer pools typically do not provide prefetching of user data.

However, the MIO library can be used to address this issue by invoking the *pf* module that detects the sequential access pattern and asynchronously preloads a much smaller cache space with data that is needed. The *pf* cache needs only to be large enough to contain enough pages to maintain sufficient read ahead (prefetching). The *pf* module can optionally use direct I/O, which avoids an extra memory copy to the system buffer pool and frees the system buffers from the one-time access of the I/O traffic, allowing the system buffers to be used more productively.

## 5.1 Design summary

The MIO library consists of four I/O modules that may be invoked at runtime on a per-file basis. The following modules are currently available:

<b>mio</b>	The interface to the user program
<b>pf</b>	A data prefetching module
<b>trace</b>	A statistics gathering module
<b>aix</b>	The MIO interface to the operating system

For each file that is opened with MIO, a minimum of two modules is invoked:

- ▶ The mio module that converts the user MIO calls (MIO\_open64, MIO\_read, MIO\_write, ...) into the internal calling sequence of MIO
- ▶ The aix module that converts the internal calling sequence of MIO into the appropriate system calls (open64, read, write,...)

Between the mio and aix module invocations, the user can specify the invocation of the other modules, pf and trace. Aside from the requirement that the mio module be at the top of the module stack and that the aix module be at the bottom of the stack, it does not matter to the MIO modules what other modules are placed in the stack for a given file.

## 5.2 Runtime control of MIO

MIO is controlled via four environment variables:

- ▶ MIO\_STATS
- ▶ MIO\_FILES
- ▶ MIO\_DEFAULTS
- ▶ MIO\_DEBUG

### 5.2.1 MIO\_STATS

MIO\_STATS is used to indicate a file that will be used as a repository for diagnostic messages and for output requested from the MIO modules. It is interpreted as a file name with two special cases. If the file is either stderr or stdout, the output is directed toward the appropriate file stream. If the first character of MIO\_STATS is a plus sign (+), the file name to be used is the string that follows the + sign and the file is opened for appending. Without the preceding + sign, the file is overwritten.

### 5.2.2 MIO\_FILES

MIO\_FILES is the key to determining which modules are to be invoked for a given file when MIO\_open64 is called. MIO\_FILES has the following format:

```
first_name_list [ module list ] second_name_list [ module list] ...
```

When MIO\_open64 is called, MIO checks for the existence of MIO\_FILES and parses it as follows:

- ▶ MIO\_FILES is parsed left to right. All characters up to the next occurrence of the left bracket ([) are taken as a file\_name\_list. A file\_name\_list is a colon (:) separated list of file\_name\_templates.

File\_name\_templates are used to match the name of the file that is being opened by MIO and might use the following wildcard characters:

- An asterisk (\*) matches zero or more characters of a directory or file name.
  - A question mark (?) matches one character of a directory or file name.
  - Double asterisks (\*\*) match all remaining characters of a full path name.
- If the file\_name\_template does not contain a forward slash (/), then all directory information in the file name passed to the MIO\_open64 subroutine is ignored and matching is applied only to the leaf name of the file that is being opened.

Here are a two examples of wildcards in action:

- If the name of the file that is being opened is matched by one of the file\_name\_templates in the file\_name\_list, then the module list to be invoked is taken as the string between the following brackets ([ ]). If the name of the file does not match any of the file\_name\_templates in the file\_name\_list, the parser moves on to the next file\_name\_list and attempts to match there. If the name of the file that is being opened does not match any of the file\_name\_templates in any of the file\_name\_lists, then the file is opened with a default invocation of the aix module.
- If a match has occurred, the modules to be invoked are taken from the associated module list in MIO\_FILES. The modules are invoked in left to right order, with the leftmost module that is closest to the user program and the rightmost module that is closest to the operating system. If the module list does not start with the mio module, a default invocation of the mio module is prepended. If the aix module is not specified, a default invocation of the aix module is appended.

The following example shows the handling of MIO\_FILES. Let us assume that the MIO\_FILES is set as follows:

```
MIO_FILES= *.dat:*.scr [ trace ] *.f01:*.f02:*.f03 [ trace | pf | trace ]
```

If the test.dat file were opened by MIO\_open64, the file name test.dat would match \*.dat, resulting in the following modules being invoked:

```
mio | trace | aix
```

If the test.f02 file were opened by MIO\_open64, the file name test.f02 would match the second file\_name\_template in the second file\_name\_list resulting in the following modules being invoked:

```
mio | trace | pf | trace | aix
```

Each module has its own hardcoded defaults options for a default invocation. The user might override the defaults options by specifying them in MIO\_FILES. The following example turns on stats for the trace module and requests that the output be directed to the my.stats file:

```
MIO_FILES= *.dat : *.scr [ trace/stats=my.stats ]
```

The options for a module are delimited with a forward slash (/). Some options require an associated integer value and others may require a string value. For those requiring a string value, if the string includes a forward slash (/), enclose the string in braces ({}).

For those options that require an integer value, the user might append the integer value with a k, m, g, or t to represent kilo (1024), mega (1024\*\*2), giga (1024\*\*3), or tera (1024\*\*4). Integer values can also be input in base 10, 8, or 16. If the integer value is prepended with a 0x, the integer is interpreted as base 16. If the integer value is prepended with a 0, the integer is interpreted as base 8. Failing the previous two tests, the integer is interpreted as base 10.

### 5.2.3 MIO\_DEFAULTS

MIO\_DEFAULTS is intended as a tool to keep MIO\_FILES more readable. If the user specifies several modules for multiple file\_name\_list/module\_list pairs, then MIO\_FILES might become quite long. If the user repeatedly overrides the hardcoded defaults, it might be easier to specify new defaults for a module by specifying them in MIO\_DEFAULTS.

MIO\_DEFAULTS is a comma separated list of modules with their new defaults. For example, let us assume that MIO\_DEFAULTS is set as follows:

```
MIO_DEFAULTS = trace/events=prob.events , aix/debug
```

Now any default invocation of the trace module has binary event tracing enabled and directed toward the prob.events file, and any default invocation of the aix module has debug enabled.

### 5.2.4 MIO\_DEBUG

MIO\_DEBUG is intended as an aid in debugging the usage of MIO. MIO searches MIO\_DEFAULTS for keywords and provides debugging output for the option. The following keywords are available:

<b>ALL</b>	Turns on all the MIO_DEBUG keywords
<b>ENV</b>	Outputs environment variable matching requests
<b>OPEN</b>	Outputs open request made to MIO_open64
<b>MODULES</b>	Outputs modules invoked for each call to MIO_open64
<b>TIMESTAMP</b>	Places a timestamp preceding each entry into a stats file
<b>DEF</b>	Outputs the definition table of each module

## 5.3 Module descriptions and options

In this section, we describe the different modules and the options that control their runtime behavior. As described previously, these options can be specified in MIO\_FILES.

#### ► mio

The mio module is the interface to the user program:

```
code_defaults=/set_oflags=0/clear_oflags=0/nomode
```

This module has the following options:

<b>mode=(0,0,0)</b>	Override the default mode
<b>nomode</b>	Do not override the mode
<b>direct</b>	Set the O_DIRECT bit
<b>nodirect</b>	Clear the O_DIRECT bit

#### ► pf

Pf is the data prefetch module:

```
code_defaults=/nodirect/stats=mioout/bytes/cache_size=64k/page_size=4k/prefetch=1/asynchronous/global/release/nopffw/memcpy/stride=1/nolistio/tag={ }/notag
```

This module has the following options:

<b>norelease</b>	Do not free the global cache pages when the global cache file usage count goes to zero
<b>release</b>	Free the global cache pages when the global cache file usage count goes to zero
<b>private</b>	Use a private cache

<b>global=(0,255,0)</b>	Use a global cache (0 to 255)
<b>asynchronous</b>	Use asynchronous calls to the child module
<b>synchronous</b>	Use synchronous calls to the child module
<b>noasynchronous</b>	Alias for synchronous
<b>direct</b>	Use DIRECT I/O
<b>nodirect</b>	Do not use DIRECT I/O
<b>bytes</b>	Stats output in units of bytes
<b>kbytes</b>	Stats output in units of KB
<b>mbytes</b>	Stats output in units of MB
<b>gbytes</b>	Stats output in units of GB
<b>tbytes</b>	Stats output in units of TB
<b>cache_size=(16384,1073741824,64k)</b>	
	The total size of the cache (in bytes)
<b>page_size=(4096,1073741824,4k)</b>	
	The size of each cache page (in bytes)
<b>prefetch=(1,100,1)</b>	The number of pages to prefetch
<b>stride=(1,1073741824,1)</b>	
	Stride factor, in pages, default is 1
<b>stats=mioout</b>	Output prefetch usage statistics
<b>nostats</b>	Do not output prefetch usage statistics
<b>inter</b>	Output intermediate prefetch usage statistics on kill -30
<b>nointer</b>	Do not output intermediate prefetch usage statistics
<b>retain</b>	Retain file data after close for subsequent reopen
<b>noretain</b>	Do not retain file data after close for subsequent reopen
<b>listio</b>	Use the listio mechanism
<b>nolistio</b>	Do not use the listio mechanism
<b>tag=</b>	String to prefix stats flow
<b>notag</b>	Do not use prefix stats flow
► trace	
Trace is a statistics gathering module:	
code_defaults=/stats=mioout/events=trace.events/noevents/sample=1/ inter=30/nointer	
This module has the following options:	
<b>stats=mioout</b>	Output statistics on close
<b>nostats</b>	Do not output statistics on close
<b>events=trace.events</b>	Generate a binary events file
<b>noevents</b>	Do not generate a binary events file
<b>bytes</b>	Output statistics in units of bytes
<b>kbytes</b>	Output statistics in units of kilobytes

<b>mbytes</b>	Output statistics in units of megabytes
<b>gbytes</b>	Output statistics in units of gigabytes
<b>tbytes</b>	Output statistics in units of terabytes
<b>inter=(30,30,30)</b>	Output intermediate trace statistics on kill -30
<b>nointer</b>	Do not output intermediate statistics

► **aix**

The aix module is the MIO interface to the operating system:

`code_defaults=/nodebug`

This module has the following options:

<b>debug</b>	Print debug statements for open and close
<b>nodebug</b>	Do not print debug statements for open and close

## 5.4 Library implementation

The interface to the MIO library was designed to be simple to implement. For applications that use the POSIX standard `open64`, `read`, `write`, `lseek64`, `fsync`, `ftruncate64`, `fstat64`, `ffinfo`, `fcntl`, and `close` I/O calls, the application programmer must only introduce `#defines` to redirect the I/O calls to use the MIO library. The `#defines` for a simple code are as follows:

**Defines:** The defines are for `open64`, `lseek64`, `ftruncate64` and `fstat64`. It is assumed that the application has defined `_LARGE_FILES` to allow for file offsets greater than 2 GB. With the define of `_LARGE_FILES`, the use of `open`, `lseek`, `ftruncate`, and `fstat` in the application is redefined to `open64`, `lseek64`, `ftruncate64`, and `fstat64` by the MACROS in `/usr/include/fcntl.h`.

- `#define open64(a,b,c) MIO_open64(a,b,c,0)`
- `#define close MIO_close`
- `#define lseek64 MIO_lseek64`
- `#define read MIO_read`
- `#define write MIO_write`
- `#define ftruncate64 MIO_ftruncate64`
- `#define fstat64 MIO_fstat64`
- `#define fcntl MIO_fcntl`
- `#define ffinfo MIO_ffinfo`
- `#define fsync MIO_fsync`

The only MIO call with arguments that differ from the corresponding standard POSIX system call is `MIO_open64`, with `MIO_open64` requiring a fourth argument that is a pointer to an `MIO_extra` structure. The simplest implementation is to pass a zero pointer as the fourth argument to `MIO_open64`.



## 5.5 Sample implementation

The following simple example demonstrates that the implementation of MIO consists of five files:

- ▶ A simple C program named `example.c`
- ▶ A makefile to compile `example.c`
- ▶ A script to run the program `example.c`
- ▶ The MIO header file `MIO_user.h`
- ▶ The resulting MIO\_STATS file `example.stats`

The `example.csh` file compiles and runs the example program. The argument to `example` is the file that is to be created, written to, and read forward and backward.

`Example.c` (Example 5-1) issues 100 writes of 16 KB, seeks to the beginning of the file, issues 100 reads of 16 KB, and then seeks backward through the file reading 16 KB records. At the end, the file is truncated to 0 bytes in length.

*Example 5-1 The example.c file*

---

```
Example.c
#define _LARGE_FILES
#include <fcntl.h>
#include <stdio.h>
#include <errno.h>

#include "MIO_user.h"

/* Note that we define open64, lseek64, ftruncate64, and not the
 * open, lseek, and ftruncate that are used in the code. This is
 * because MIO_user.h defines _LARGE_FILES which forces <fcntl.h> to
 * redefine open, lseek, and ftruncate as open64, lseek64, and
 * ftruncate64
 */

#define open64(a,b,c) MIO_open64(a,b,c,0)
#define close      MIO_close
#define lseek64    MIO_lseek64
#define write      MIO_write
#define read       MIO_read
#define ftruncate64 MIO_ftruncate64

#define RECSIZE 16384
#define NREC    100

main(int argc, char **argv)
{
    int i, fd, status ;
    char *name ;
    char *buffer ;
    int64 ret64 ;

    if( argc < 2 ){
        fprintf(stderr,"Usage : example file_name\n");
        exit(-1);
    }
}
```

```

name = argv[1] ;

buffer = (char *)malloc(RECSIZE);
memset( buffer, 0, RECSIZE ) ;

fd = open(name, O_RDWR|O_TRUNC|O_CREAT, 0640 ) ;
if( fd < 0 ){
    fprintf(stderr,"Unable to open file %s errno=%d\n",name,errno);
    exit(-1);
}

/* write the file */
for(i=0;i<NREC;i++){
    status = write( fd, buffer, RECSIZE ) ;
}

/* read the file forwards */
ret64 = lseek(fd, 0, SEEK_SET ) ;
for(i=0;i<NREC;i++){
    status = read( fd, buffer, RECSIZE ) ;
}

/* read the file backwards */
for(i=0;i<NREC;i++){
    ret64 = lseek(fd, (NREC-i-1)*RECSIZE, SEEK_SET ) ;
    status = read( fd, buffer, RECSIZE ) ;
}

/* truncate the file back to 0 bytes*/
status = ftruncate( fd, 0 ) ;

/* close the file */
status = close(fd);
}

```

---

Example 5-2 shows the makefile.bgp file.

---

*Example 5-2 The makefile.bgp file*

---

```

CC=/bgusr/walkup/bin/mpxlc
INC=-I$(IHPCT_BASE)/src/mio

MIO_LIB = -L$(IHPCT_BASE)/lib -lmio
### for 64-bit applications ###
#MB=64
#MIO_LIB=$(IHPCT_BASE)/lib64

### for 32-bit applications ###
$MB=32
MIO_LIB=$(IHPCT_BASE)/lib

.c.o:
    $(CC) -c -DBGP -DBGL -DMIO_LARGE_FILES $(INC) \
        -DpLinux \
        -DOS_TYPE="pLinux\" \
        -D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE \

```

```

        -D_FILE_OFFSET_BITS=64 -D_GNU_SOURCE \
    $<

all: example

example: example.o
        $(CC) $(CFLAGS) $(INC) -o example example.o -L$(MIO_LIB) -lmio

clean:
        rm -f *.o example *.evt *.stats *.pbm

realclean: clean

```

---

For Example 5-3, we ran the program using **mpirun** on partition N11\_32\_1.

---

*Example 5-3 Run script*

---

```

mpirun -verbose 1 -partition N11_32_1 \
-cwd /bgusr/sseelam/simple-example -np 1 \
-env "MIO_STATS=xml.stats" \
-env "MIO_DEFAULTS=trace" -env "MIO_FILES=[trace/events=example.evt|pf]" \
-env "MIO_DEBUG=all" \
-exe /bgusr/sseelam/simple-example/example \
-args "hello"

```

---

Example 5-4 contains the sample MIO header file.

---

*Example 5-4 MIO header file*

---

```

#if !defined(_MIO_USER_H )
#define _MIO_USER_H

#if !defined(_LARGE_FILES)
#define _LARGE_FILES
#endif

#include <fcntl.h>
#include <stdio.h>
#include <errno.h>
#include <aio.h>

#define MIO_EXTRA_SKIP_MIO_FILES_FLAG      0x2
#define MIO_EXTRA_SKIP_MIO_DEFAULTS_FLAG  0x4
#define MIO_EXTRA_OPEN_FOR_UNLINK         0x8

#define MIO_EXTRA_COOKIE_OLD 0x7a78746b
#define MIO_EXTRA_COOKIE     0x7a78746c
struct mio_extra_old {
    int    cookie ;
    int    taskid ;
    int64  bufsiz ;
    char *modules ;
    char *logical_name ;
} ;
struct mio_extra {
    int    cookie ;

```

```

    int    taskid ;
    int64  bufsiz ;
    char *modules ;
    char *logical_name ;
    int    flags ;
    int    extra_errno ;
    int    reserved[8] ;
} ;

#define MIO_EXTRA_FLAG_SCRATCH 0x80000000

#if !defined(MIO_STRUCT_AIOCB)
#define MIO_STRUCT_AIOCB struct aiocb64
#define MIO_STRUCT_LIOCB struct liocb64
#endif

extern int MIO_open64(char *, int, int, struct mio_extra *);
extern int MIO_fstat64( int, struct stat64 *);
extern int64 MIO_lseek64( int, int64, int);
extern int MIO_ftruncate64( int, int64);
extern int MIO_read( int, void *, int);
extern int MIO_write( int, void *, int);
extern int MIO_close( int);
extern int MIO_fcntl( int, int, int *);
extern int MIO_ffinfo( int, int, struct diocapbuf *, int);
extern int MIO_fsync( int);
extern int64 MIO_str_to_long( char *);
extern int MIO_str_to_long_vector(char *, int64*, int);
extern int MIO_aio_read64( int , MIO_STRUCT_AIOCB * ) ;
extern int MIO_aio_write64( int , MIO_STRUCT_AIOCB * ) ;
extern int MIO_aio_suspend64( int , MIO_STRUCT_AIOCB ** ) ;
extern int MIO_aio_nwait64( int , int , MIO_STRUCT_AIOCB ** ) ;
extern int MIO_aio_cancel64( int , MIO_STRUCT_AIOCB * ) ;
extern int MIO_lio_listio64( int , MIO_STRUCT_LIOCB **, int , void * ) ;

#if defined(USE_MIO_DEFINES)
#define open64(a,b,c)      MIO_open64(a,b,c,0)
#define close(a)           MIO_close(a)
#define read(a,b,c)        MIO_read(a,b,c)
#define write(a,b,c)       MIO_write(a,b,c)
#define lseek64(a,b,c)     MIO_lseek64(a,b,c)
#define ftruncate64(a,b)   MIO_ftruncate64(a,b)
#define fsync(a)           MIO_fsync(a)
#define fstat64(a,b)       MIO_fstat64(a,b)
#define ffinfo(a,b,c,d)    MIO_ffinfo(a,b,c,d)
#define fcntl(a,b,c)       MIO_fcntl(a,b,c)
#define aio_read64(a,b)    MIO_aio_read64(a,b)
#define aio_write64(a,b)   MIO_aio_write64(a,b)
#define aio_suspend64(a,b) MIO_aio_suspend64(a,b)
#define aio_nwait(a,b,c)   MIO_aio_nwait64(a,b,c)

```

```
#define aio_cancel64(a,b)    MIO_aio_cancel64(a,b)
#define lio_listio64(a,b,c,d) MIO_lio_listio64(a,b,c,d)
#endif

#endif /* _MIO_USER_H */
```

---

The results can be displayed as simple statistics or in a figure. Example 5-5 shows how to return the summary results as statistics.

*Example 5-5 Program to display statistics*

---

```
MIO statistics file : Thu Jan 1 00:04:37 1970
hostname=172.24.101.122 : without aio available
Program=(null) pid=100 (not threaded)
MIO library libmio.a 3.0.3.046 BGL 32 bit addressing built Nov 12 2007 16:11:58
MIO_INCLUDE_PATH=(null)
MIO_STATS      =xml.stats
MIO_DEBUG      =all
MIO_FILES      =*[trace/events=example.evt|pf]
MIO_DEFAULTS   =trace
```

---

Example 5-6 provides an example of returning the results in a figure.

*Example 5-6 Program to create a figure*

---

```
Trace close : program <-> pf : hello : (4915200/0.49)=9991380.00 bytes/s
      demand rate=8722271.00 bytes/s=4915200/(0.59-0.03))
      open_size=0, current_size=0 max_size=1638400
mode =0640 FileSystemType=NFS sector size=4096
oflags =0x242=RDWR CREAT TRUNC
open          1      0.00
write         100    0.20   1638400 ==   1638400   16384   16384
read          200    0.30   3276800 ==   3276800   16384   16384
seek          101    0.00
              101 backward seeks average=48503
fcntl         1      0.00
trunc         1      0.04
close         1      0.00
```

---

Figure 5-1 shows the I/O access pattern of the application. The blue line on the left indicates writing activity and the red lines on the right indicates reading. Forward and backward seeks are respectively indicated by the positive and negative slopes of the lines.

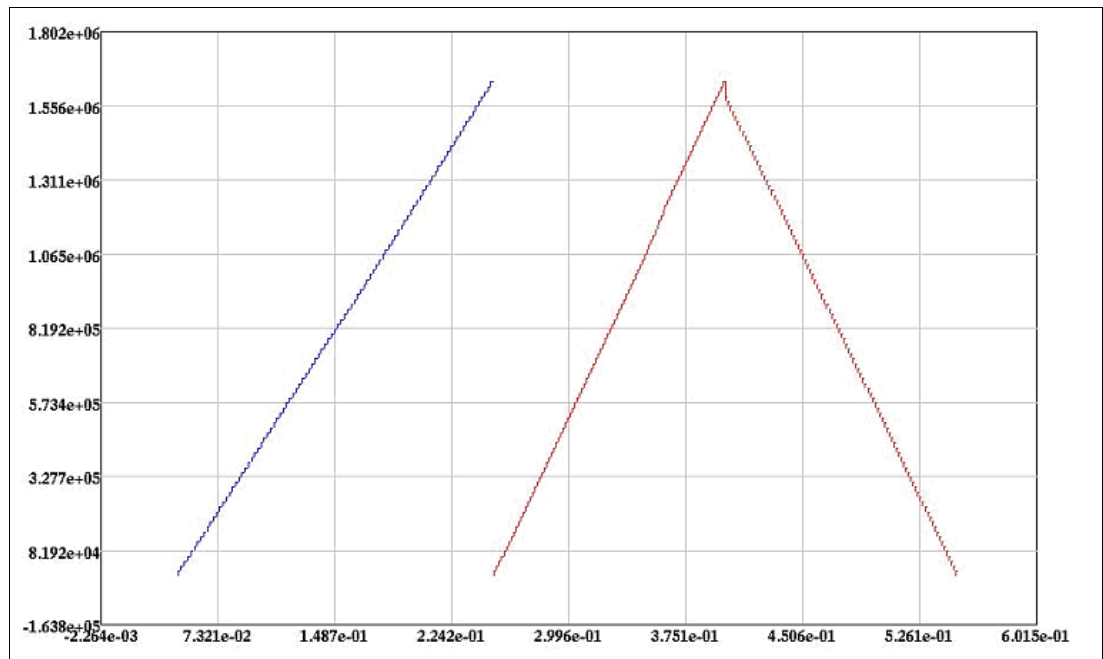


Figure 5-1 I/O access pattern

In the following figures, we show examples of MIO tracing of the I/O access pattern that is present in the IOR and BTIO benchmarks. Figure 5-2 shows the POSIX I/O profiling for the IOR application.

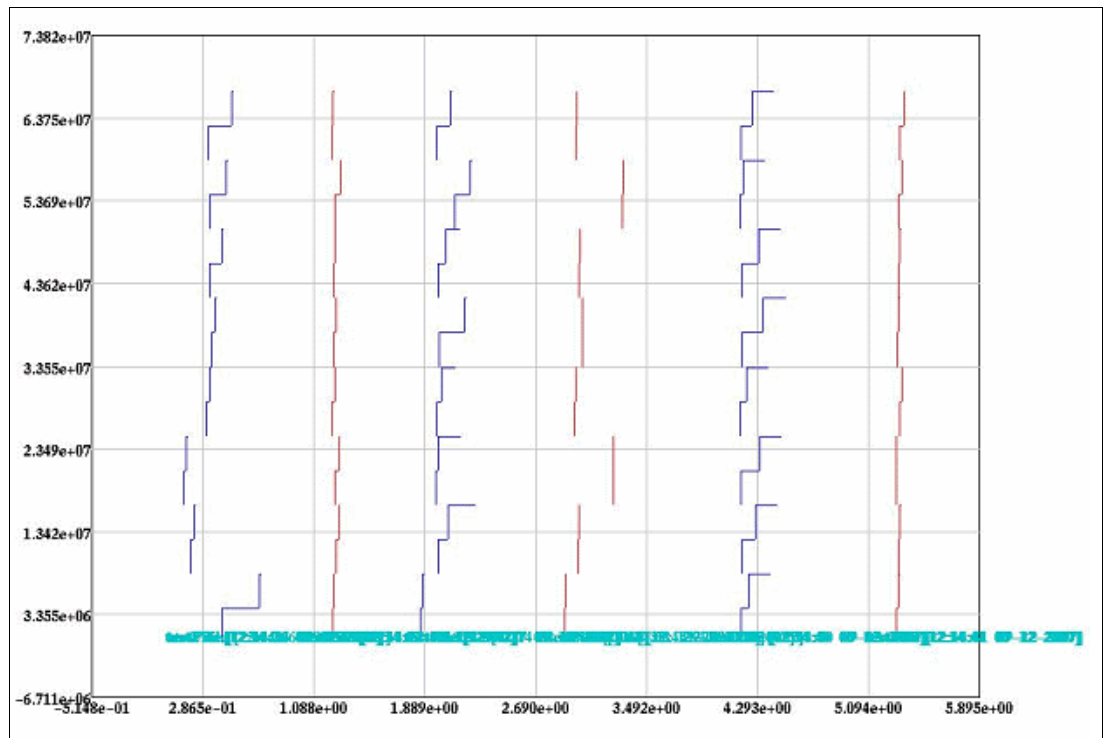


Figure 5-2 POSIX

Figure 5-3 is the Fortran I/O for the BTIO application.

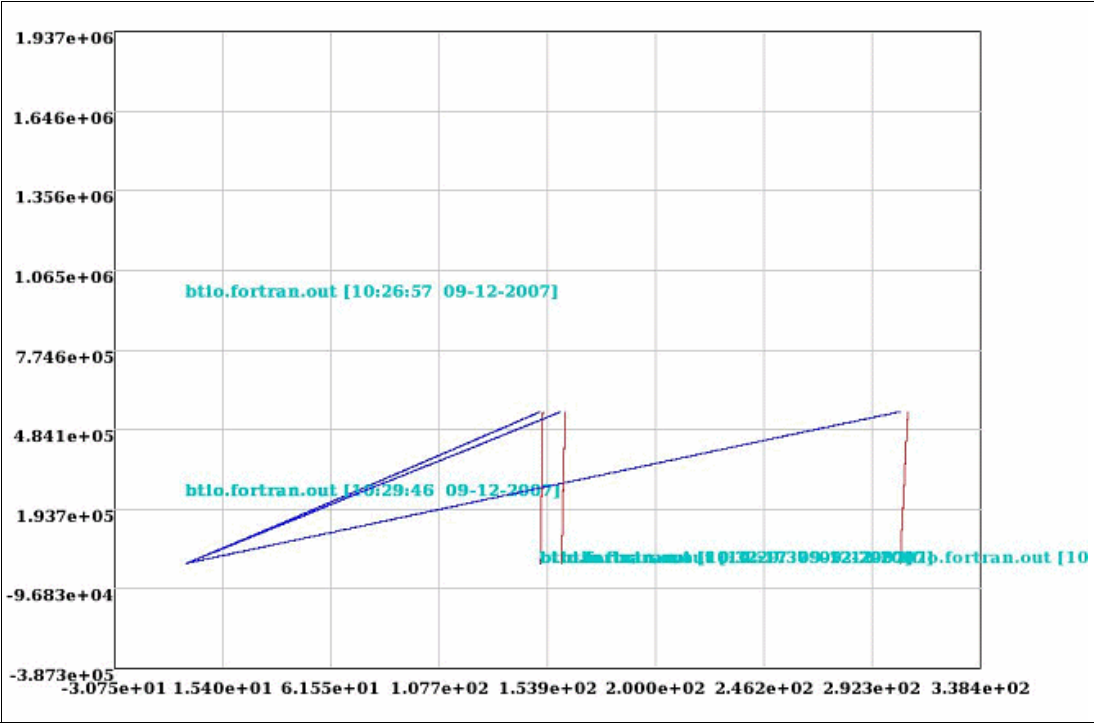


Figure 5-3 Fortran

Figure 5-4 shows EPIO mode for the BTIO application.

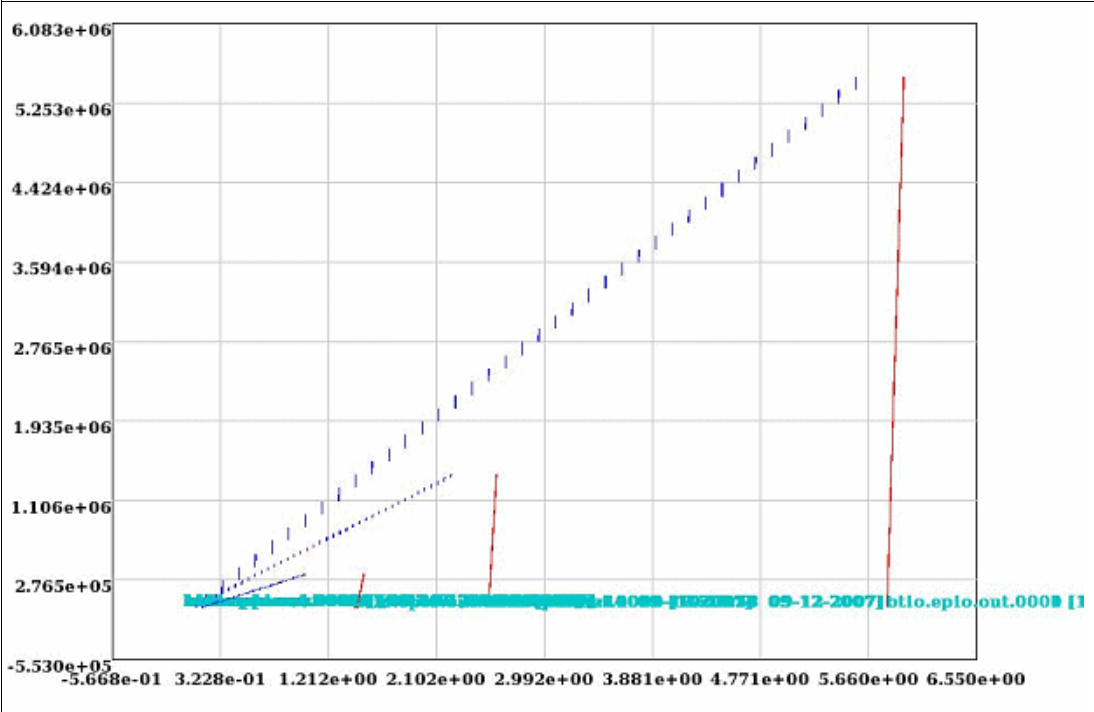


Figure 5-4 EPIO





# Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this paper.

## IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks” on page 125. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *Blue Gene Safety Considerations*, REDP-4257
- ▶ *Evolution of the IBM System Blue Gene Solution*, REDP-4247
- ▶ *IBM System Blue Gene Solution: Blue Gene/P Application Development*, SG24-7287
- ▶ *IBM System Blue Gene Solution: Blue Gene/P System Administration*, SG24-7417

## How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

[ibm.com/redbooks](http://ibm.com/redbooks)

## Help from IBM

IBM Support and downloads

[ibm.com/support](http://ibm.com/support)

IBM Global Services

[ibm.com/services](http://ibm.com/services)







**Redpaper**

# IBM System Blue Gene Solution: High Performance Computing Toolkit for Blue Gene/P

**Tools to visualize and  
analyze your  
performance data**

**Instructions for the  
Xprofiler and High  
Performance  
Computing Toolkit  
GUIs**

**Tips to optimize your  
application's  
performance**

This IBM Redpaper publication is one in a series of IBM documents written specifically for the IBM System Blue Gene/P Solution. The Blue Gene/P system is the second generation of a massively parallel supercomputer from IBM in the IBM System Blue Gene Solution series. This paper provides an overview of the IBM High Performance Computing Toolkit for the Blue Gene/P system.

We begin by describing the Message Passing Interface (MPI) Profiler and Tracer tool, which collects profiling and tracing data for MPI programs. We explain the system requirements as well as configuration, compiling, linking, environment variables, and output.

Next we discuss how to use Xprofiler for CPU profiling. We then move on to discuss Hardware Performance Monitoring (HPM), including the use and behavior of the libhpm library. Afterward, we describe the GUI of the High Performance Computing Toolkit (HPCT). This single interface provides a means to execute the application and visualize and analyze the collected performance data.

Finally we address I/O performance. Specifically, we discuss the features of the Modular I/O (MIO) library that was developed to assist in optimizing an application's I/O.

**INTERNATIONAL  
TECHNICAL  
SUPPORT  
ORGANIZATION**

**BUILDING TECHNICAL  
INFORMATION BASED ON  
PRACTICAL EXPERIENCE**

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

**For more information:  
[ibm.com/redbooks](http://ibm.com/redbooks)**