

IBM System Blue Gene Solution: Performance Analysis Tools

PAPI installation and usage

Tools to visualize and analyze your
performance data

I/O node and GPFS
performance tuning



Gary Lakner
I-Hsin Chung
Dr. Guojing Cong
Scott Fadden
Nicholas Goracke
David Klepacki
Jeffrey Lien
Dr. Christoph Pospiech
Seetharami R. Seelam
Hui-Fang Wen



International Technical Support Organization

**IBM System Blue Gene Solution: Performance Analysis
Tools**

November 2008

Archived

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

Archived

Second Edition (November 2008)

This edition applies to Version 1, Release 2, Modification 0 of IBM System Blue Gene/P Solution (product number 5733-BGP).

© Copyright International Business Machines Corporation 2007, 2008. 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

Notices	vii
Trademarks	viii
Preface	ix
The team that wrote this paper	ix
Become a published author	x
Comments welcome	xi
Summary of changes	xiii
November 2008, Second Edition	xiii
Part 1. Performance Application Programming Interface	1
Chapter 1. Implementing the Performance Application Programming Interface on Blue Gene/P	3
1.1 Installing PAPI	4
1.2 Modifications to PAPI for Blue Gene/P	5
1.2.1 PAPI_read	5
1.2.2 papi_bgp_read	6
1.2.3 PAPI presets	6
1.2.4 Limitations	14
Chapter 2. PAPI usage	15
2.1 Usage example	16
2.1.1 Headers	17
2.1.2 Flow of events	17
2.1.3 Running the code	18
2.2 Suggestions	19
2.2.1 Using PAPI preset values	19
2.2.2 Running in Dual or Virtual Node mode	20
2.2.3 Additional methods	22
Chapter 3. Automatically Available Performance Counters	29
3.1 Automatically Available Performance Counters for MPI applications	30
3.1.1 Enabling Automatically Available Performance Counters	30
3.1.2 Compiling with Automatically Available Performance Counters	30
3.1.3 How performance counters work	30
3.1.4 Changing runtime parameters	32
3.2 Performance counters in HTC Mode	32
3.2.1 Modifications to application code	33
3.2.2 Compiling the application	34
3.2.3 Running the application	34
3.2.4 Caveats when running in HTC mode	35
Part 2. High Performance Computing Toolkit	37
Chapter 4. MPI Profiler and Tracer	39
4.1 System and software requirements	40
4.2 Compiling and linking	40
4.3 Environment variables	41

4.3.1	TRACE_ALL_EVENTS	41
4.3.2	TRACE_ALL_TASKS	42
4.3.3	TRACE_MAX_RANK	42
4.3.4	TRACEBACK_LEVEL	42
4.3.5	SWAP_BYTES	42
4.3.6	TRACE_SEND_PATTERN (Blue Gene/L and Blue Gene/P only)	43
4.4	Output	43
4.4.1	Plain text file	44
4.4.2	The VIZ file	47
4.4.3	Trace file	48
4.5	Configuration	48
4.5.1	Configuration functions	48
4.5.2	Data structure	49
4.5.3	Utility functions	51
4.6	Related issues	54
4.6.1	Overhead	54
4.6.2	Multithreading	54
Chapter 5. CPU profiling using Xprofiler		55
5.1	Starting Xprofiler	56
5.2	Understanding the Xprofiler display	58
5.2.1	Xprofiler main menus	59
5.2.2	Elements of the function call tree	60
5.2.3	Manipulating the Xprofiler display	62
5.3	Getting performance data for your application	69
Chapter 6. Hardware Performance Monitoring		77
6.1	HPM	78
6.2	Events and groups	79
6.3	Derived metrics	121
6.4	Inheritance	121
6.5	Inclusive and exclusive values	122
6.5.1	Parent-child relations	122
6.5.2	Handling overlap issues	123
6.5.3	Computation of exclusive values for derived metrics	123
6.6	Function reference	123
6.7	Measurement overhead	125
6.8	Output	125
6.9	Examples of libhpm for C and C++	126
6.10	Multithreaded program instrumentation issues	127
6.11	Considerations for MPI parallel programs	127
6.11.1	Distributors	128
6.11.2	Aggregators	128
6.11.3	Plug-ins shipped with HPCT	128
6.11.4	User-defined plug-ins	129
6.11.5	Detailed interface description	129
6.11.6	Getting the plug-ins to work	131
Chapter 7. High Performance Computing Toolkit GUI		133
7.1	Starting the HPCT GUI	134
7.2	HPCT GUI Main window (Visualization)	134
7.3	HPCT GUI Main Window with instrumentation	142
7.4	HPCT GUI simple IDE	147

Chapter 8. I/O performance	149
8.1 Design summary	150
8.2 Runtime control of MIO	150
8.2.1 MIO_STATS	150
8.2.2 MIO_FILES	150
8.2.3 MIO_DEFAULTS	152
8.2.4 MIO_DEBUG	152
8.3 Module descriptions and options	152
8.4 Library implementation	154
8.5 Sample implementation	155
Part 3. GPFS and I/O node	163
Chapter 9. IBM General Parallel File System and I/O node performance tuning	165
9.1 Cluster architecture	166
9.2 System configuration and settings	166
9.2.1 TCP settings	166
9.2.2 GPFS NSD server configuration and setup	167
9.3 Verifying startup	169
9.4 Performance tuning the InfiniBand interface	171
9.5 GPFS Blue Gene/P I/O node settings	172
9.6 GPFS Service and login node settings	174
9.7 Storage array settings	175
9.8 Performance verification	175
9.8.1 Storage performance	176
9.8.2 Network performance tools	177
9.9 Total system performance	181
9.10 Troubleshooting I/O size	183
9.11 Example scripts	184
9.11.1 dd scripts	184
9.11.2 NSDPerf scripts and commands	187
Related publications	191
IBM Redbooks	191
How to get IBM Redbooks	191
Help from IBM	191

Archived

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

IBM, the IBM logo, and [ibm.com](http://www.ibm.com) are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at <http://www.ibm.com/legal/copytrade.shtml>

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®
General Parallel File System™
GPFS™

IBM®
POWER™
Power Systems™
POWER4™
POWER5™
POWER5+™
POWER6™

PowerPC®
Redbooks®
Redbooks (logo) ®
System i®
System p®
Tracer™

The following terms are trademarks of other companies:

InfiniBand, and the InfiniBand design marks are trademarks and/or service marks of the InfiniBand Trade Association.

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. The first section of this paper provides an overview of the Performance Application Programming Interface (PAPI). In this section, we describe the Blue Gene® implementation of PAPI and usage. The second part of this paper gives a detailed description of the IBM High Performance Computing Toolkit for the Blue Gene/P™ system. Included in this section is a discussion of the MPI Profiler and Tracer™, CPU profiling, the Hardware Performance Monitor, and the Modular I/O library. The final section of this paper gives tips for tuning I/O performance and the IBM General Parallel File System™ (GPFS™).

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), Poughkeepsie Center.

Gary Lakner is a Staff Software Engineer for IBM Rochester on assignment in the ITSO. He is a member of the Blue Gene 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.

Scott Fadden is a performance engineer on the General Parallel File System (GPFS) development team at IBM. He has more than 10 years experience designing and implementing solutions for enterprise analytics, ETL and Business intelligence from large data warehouses to grid computing. As a GPFS performance engineer he specializes in complex solution design and performance tuning.

Nicholas Goracke is a Software Engineer for IBM Rochester working on both the Blue Gene Functional Test Team and Performance Team. He specializes in Blue Gene debugger interface and Web GUI test, as well as development and design of the performance counter APIs. Nicholas joined IBM in 2005 after graduating from the University of Minnesota: Twin Cities with a Bachelor's degree in Computer Science.

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.

Jeffrey Lien is an Advisory Software Engineer for IBM Rochester and is a member of the Blue Gene performance team. He specializes in measuring and tuning I/O performance for Blue Gene customers as well as running I/O benchmarks needed for customer bids. Prior to joining the Blue Gene performance team, Jeff developed and tested I/O device drivers used on Ethernet and modem adapter cards. Jeff has been with IBM since 1990.

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 in 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 the 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 Master's in Computer Science degree from University of Maryland, College Park. She has been with IBM since 2005.

Special thanks to:

Tom Gray
Todd Kelsey
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

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 book form found at:

ibm.com/redbooks

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

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

Archived

Summary of changes

This section describes the technical changes made in this edition of the paper and in previous editions. This edition might also include minor corrections and editorial changes that are not identified.

Summary of Changes
for *IBM System Blue Gene Solution: Performance Analysis Tools*
as created or updated on June 4, 2009

November 2008, Second Edition

This revision reflects the addition, deletion, or modification of new and changed information described in the following sections.

New information

- ▶ Part 1, “Performance Application Programming Interface” on page 1 was added to this version of this paper. It includes three chapters, one on implementing PAPI, another on PAPI Usage, and the third describes Automatically Available Performance Counters.
- ▶ Part 3, “GPFS and I/O node” on page 163 has been added to help with performance tuning GPFS and the Blue Gene I/O nodes.

Archived



Part 1

Performance Application Programming Interface

This part provides details about the Performance Application Programming Interface (PAPI) support that is provided on Blue Gene/P.

Archived

Implementing the Performance Application Programming Interface on Blue Gene/P

The Performance Application Programming Interface (PAPI) implementation on Blue Gene/P is based on the 3.9.0 release (PAPI-C). When you finish installing the PAPI library using the steps outlined in this chapter, you will have a PAPI library that consists of two parts: the common library API and a substrate interface. The common library API contains all of the PAPI functions and is identical to all other platform implementations, while the substrate contains all of the Blue Gene/P platform-specific code to enable these functions.

The information in this chapter can also be located on your service node in the file `/bgsys/drivers/ppcfloor/tools/papi/papi_install.txt`.

Note: General information about PAPI can be located on the PAPI Web site at:

<http://icl.cs.utk.edu/papi/index.html>

An overview of PAPI C Version 3.9.0 and the technical documentation can be found by clicking on the **Software** link.

1.1 Installing PAPI

There are two files required to build the PAPI library for Blue Gene/P. First, you will need to download the PAPI-C Version 3.9.0 Technical Pre-Release tar file available at:

<http://icl.cs.utk.edu/papi/software/index.html>

This tar file contains all of the base PAPI functionality.

Second, you will need the PAPI patch file, which will provide all of the platform specific code necessary to run on Blue Gene/P. This patch is included with the Blue Gene/P software installation, and can be found in the `/bgsys/drivers/ppcfloor/tools/papi` directory.

The following steps will walk you through installing PAPI-C:

1. Place the `papi-c-3.9.0.tar.gz` file that was downloaded from the PAPI Web site into your home directory. Extract the code by running the following command:

```
tar -xzvf /home/bgpUser/papi-c-3.9.0.tar.gz -C /home/bgpUser
```

2. Apply the patch using the following command:

```
patch -d /home/bgpUser/papi-c-3.9.0 -i  
/bgsys/drivers/ppcfloor/tools/papi/papi_390.patch -p 5
```

You should see output similar to Example 1-1.

Example 1-1 Output from running the patch command

```
patching file INSTALL.txt  
patching file src/config.h  
patching file src/config.log  
patching file src/config.status  
patching file src/configure  
patching file src/configure.in  
patching file src/ctests/bgp/Makefile  
patching file src/ctests/bgp/papi_1.c  
patching file src/ctests/native.c  
patching file src/ctests/papi_test.h  
patching file src/extras.c  
patching file src/f77papi.h  
patching file src/f90papi.h  
patching file src/fpapi.h  
patching file src/linux-bgp.c  
patching file src/linux-bgp.h  
patching file src/linux-bgp-memory.c  
patching file src/linux-bgp-native-events.h  
patching file src/linux-bgp-preset-events.c  
patching file src/Makefile  
patching file src/Makefile.inc  
patching file src/Makefile.linux-bgp  
patching file src/papi.c  
patching file src/papi_data.c  
patching file src/papi_fwrappers__.c  
patching file src/papi_fwrappers_.c  
patching file src/PAPI_FWRAPPERS.c  
patching file src/papi.h  
patching file src/papi_h1.c  
patching file src/papi_internal.c
```

```
patching file src/papi_internal.h
patching file src/papi_preset.h
patching file src/papi_protos.h
patching file src/papiStdEventDefs.h
patching file src/papi_vector.c
```

3. To build this version of PAPI, edit `/home/bgpUser/papi-c-3.9.0/src/Makefile.linux-bgp` and set the following variables:
 - `BGP_SYSDIR = /bgsys/drivers/ppcfloor`.
 - `BGP_PAPI_PATH =` to the path where the built PAPI code is to be installed, for example, `/home/bgpUser/PAPI_Library`.

Tip: Before moving to the next step, make sure the compilers (by default, `/bgsys/drivers/ppcfloor/gnu-linux/bin`) are in your path, for example:

```
export PATH=$PATH:/bgsys/drivers/ppcfloor/gnu-linux/bin
```

4. Change to the PAPI source directory, then run the `configure` command:

```
./configure --with-OS=bgp
```

Errors relating to this command will be sent to the `config.log` file.c.- 5. Make and install PAPI-C. While still in the `src` directory, run the following commands:

```
make
make install
```

This completes the install process. By linking to this library, you will be able to make use of PAPI functions in your application.

1.2 Modifications to PAPI for Blue Gene/P

The substrate, which enables the base PAPI functions for use on Blue Gene/P, is built on top of the Blue Gene/P performance counter interface. Documentation for this interface can be found in the `/bgsys/drivers/ppcfloor/arch/include/UPC.h` header file. To illustrate how the substrate works, the `PAPI_read` function is provided as an example. This function is located in the `papi.c` file of the `src` directory of the PAPI V3.9.0 download.

1.2.1 PAPI_read

The code included in the `PAPI_read` function is identical among all platform implementations. The substrate provides platform-specific customized methods for the `_papi_hwi_*` calls. These platform-specific methods are located in the `linux-bgp.c` file, which will also be located in the `src` directory of the PAPI V3.9.0 download after the Blue Gene/P patch has been applied.

1.2.2 papi_bgp_read

The bolded section, `BGP_UPC_Read_Counters((long_long*)&this_state->counters[0], BGP_UPC_MAXIMUM_LENGTH_READ_COUNTERS_ONLY, BGP_UPC_READ_SHARED)`, in Example 1-2 makes a call to the Blue Gene/P UPC interface to read the hardware counters, then passes that data back to the base PAPI code.

Example 1-2 _papi_bgp_read

```
//_papi_bgp_read code
int _papi_bgp_read(hwd_context_t *ctx, hwd_control_state_t *this_state, long_long
**dp, int flags)
{
// printf("_papi_bgp_read: this_state* = %p\n", this_state);
// printf("_papi_bgp_read: (long_long*)&this_state->counters[0] = %p\n",
(long_long*)&this_state->counters[0]);
// printf("_papi_bgp_read: (long_long*)&this_state->counters[1] = %p\n",
(long_long*)&this_state->counters[1]);
if (BGP_UPC_Read_Counters((long_long*)&this_state->counters[0],
BGP_UPC_MAXIMUM_LENGTH_READ_COUNTERS_ONLY, BGP_UPC_READ_SHARED) < 0)
return PAPI_ESBSTR;
*dp = (long_long*)&this_state->counters[0];

// printf("_papi_bgp_read: dp = %p\n", dp);
// printf("_papi_bgp_read: *dp = %p\n", *dp);
// printf("_papi_bgp_read: (*dp)[0]* = %p\n", &((*dp)[0]));
// printf("_papi_bgp_read: (*dp)[1]* = %p\n", &((*dp)[1]));
// printf("_papi_bgp_read: (*dp)[2]* = %p\n", &((*dp)[2]));
// int i;
// for (i=0; i<256; i++)
// if ((*dp)[i])
// printf("_papi_bgp_read: i=%d, (*dp)[i]=%lld\n", i, (*dp)[i]);

return PAPI_OK;
}
```

1.2.3 PAPI presets

There are two Blue Gene/P specific PAPI presets included to monitor torus activity. They are:

- ▶ `PAPI_BGP_TS_DPKT`: Counts torus packets sent in x, y, and z directions.
- ▶ `PAPI_BGP_TS_32B`: Counts torus 32 byte chunks sent in x, y, and z directions.

The Blue Gene/P substrate for PAPI includes a default mapping of standard PAPI events to available counters in the Blue Gene/P hardware counter infrastructure. Due to the nature of the application-specific integrated circuit (ASIC) design of Blue Gene/P, many events available on commodity machines are not available on this platform. In addition, because the design of the UPC unit for Blue Gene/P only allows for monitoring of a subset of cores on any given node, many of the standard PAPI presets are hard coded to include counters corresponding to cores 0 and 1.

The following PAPI presets return accurate results for applications running in SMP mode using zero or one threads (see Example 1-3 on page 7). They can be used with applications running in SMP using 2+ threads, dual mode, or virtual node mode, but only events from cores 0 and 1 will be counted. Suggestions for obtaining values for these PAPI presets on cores 2 and 3 are included in Example 1-4 on page 11.

Example 1-3 Presets for SMP processing

```
/* Level 1 Data Cache Misses */
{PAPI_L1_DCM, {DERIVED_ADD,
              {PNE_BGP_PU0_DCACHE_MISS,
               PNE_BGP_PU1_DCACHE_MISS,
               PAPI_NULL}},
              {0,}}},

/* Level 1 Instruction Cache Misses */
{PAPI_L1_ICM, {DERIVED_ADD,
              {PNE_BGP_PU0_ICACHE_MISS,
               PNE_BGP_PU1_ICACHE_MISS,
               PAPI_NULL}},
              {0,}}},

/* Level 1 Total Cache Misses */
{PAPI_L1_TCM, {DERIVED_ADD,
              {PNE_BGP_PU0_DCACHE_MISS,
               PNE_BGP_PU1_DCACHE_MISS,
               PNE_BGP_PU0_ICACHE_MISS,
               PNE_BGP_PU1_ICACHE_MISS,
               PAPI_NULL}},
              {0,}}},

/* Snoops */
{PAPI_CA_SNP, {DERIVED_ADD,
              {PNE_BGP_PU0_L1_INVALIDATION_REQUESTS,
               PNE_BGP_PU1_L1_INVALIDATION_REQUESTS,
               PAPI_NULL}},
              {0,}}},

/* Prefetch Data Instruction Caused a Miss */
{PAPI_PRF_DM, {DERIVED_ADD,
              {PNE_BGP_PU0_ICACHE_MISS,
               PNE_BGP_PU1_ICACHE_MISS,
               PAPI_NULL}},
              {0,}}},

/* FMA instructions completed */
{PAPI_FMA_INS, {DERIVED_ADD,
               {PNE_BGP_PU0_FPU_FMA_2,
                PNE_BGP_PU1_FPU_FMA_2,
                PNE_BGP_PU0_FPU_FMA_4,
                PNE_BGP_PU1_FPU_FMA_4,
                PAPI_NULL}},
               {0,}}},

/* Floating point instructions */
{PAPI_FP_INS, {DERIVED_ADD,
              {PNE_BGP_PU0_FPU_ADD_SUB_1,
               PNE_BGP_PU1_FPU_ADD_SUB_1,
               PNE_BGP_PU0_FPU_MULT_1,
               PNE_BGP_PU1_FPU_MULT_1,
               PNE_BGP_PU0_FPU_FMA_2,
               PNE_BGP_PU1_FPU_FMA_2,
```

```

        PNE_BGP_PU0_FPU_DIV_1,
        PNE_BGP_PU1_FPU_DIV_1,
        PNE_BGP_PU0_FPU_OTHER_NON_STORAGE_OPS,
        PNE_BGP_PU1_FPU_OTHER_NON_STORAGE_OPS,
        PNE_BGP_PU0_FPU_ADD_SUB_2,
        PNE_BGP_PU1_FPU_ADD_SUB_2,
        PNE_BGP_PU0_FPU_MULT_2,
        PNE_BGP_PU1_FPU_MULT_2,
        PNE_BGP_PU0_FPU_FMA_4,
        PNE_BGP_PU1_FPU_FMA_4,
        PNE_BGP_PU0_FPU_DUAL_PIPE_OTHER_NON_STORAGE_OPS,
        PNE_BGP_PU1_FPU_DUAL_PIPE_OTHER_NON_STORAGE_OPS,
        PAPI_NULL},
    {0,}},

/* Load Instructions Executed */
{PAPI_LD_INS, {DERIVED_ADD,
               {PNE_BGP_PU0_DATA_LOADS,
                PNE_BGP_PU1_DATA_LOADS,
                PAPI_NULL},
               {0,}}},

/* Store Instructions Executed */
{PAPI_SR_INS, {DERIVED_ADD,
               {PNE_BGP_PU0_DATA_STORES,
                PNE_BGP_PU1_DATA_STORES,
                PAPI_NULL},
               {0,}}},

/* Total Load/Store Instructions Executed */
{PAPI_LST_INS, {DERIVED_ADD,
                {PNE_BGP_PU0_DATA_LOADS,
                 PNE_BGP_PU1_DATA_LOADS,
                 PNE_BGP_PU0_DATA_STORES,
                 PNE_BGP_PU1_DATA_STORES,
                 PAPI_NULL},
                {0,}}},

/* Level 1 Data Cache Hit */
{PAPI_L1_DCH, {DERIVED_ADD,
               {PNE_BGP_PU0_DCACHE_HIT,
                PNE_BGP_PU1_DCACHE_HIT,
                PAPI_NULL},
               {0,}}},

/* Level 1 Data Cache Accesses */
{PAPI_L1_DCA, {DERIVED_ADD,
               {PNE_BGP_PU0_DCACHE_HIT,
                PNE_BGP_PU1_DCACHE_HIT,
                PNE_BGP_PU0_DCACHE_MISS,
                PNE_BGP_PU1_DCACHE_MISS,
                PAPI_NULL},
               {0,}}},

/* Level 1 Data Cache Reads */

```



```

{PAPI_L1_DCR, {DERIVED_ADD,
               {PNE_BGP_PU0_DATA_LOADS,
                PNE_BGP_PU1_DATA_LOADS,
                PAPI_NULL}},
              {0,}}},

/* Level 1 Instruction Cache Hits */
{PAPI_L1_ICH, {DERIVED_ADD,
               {PNE_BGP_PU0_ICACHE_HIT,
                PNE_BGP_PU1_ICACHE_HIT,
                PAPI_NULL}},
              {0,}}},

/* Level 1 Instruction Cache Accesses */
{PAPI_L1_ICA, {DERIVED_ADD,
               {PNE_BGP_PU0_ICACHE_HIT,
                PNE_BGP_PU1_ICACHE_HIT,
                PNE_BGP_PU0_ICACHE_MISS,
                PNE_BGP_PU1_ICACHE_MISS,
                PAPI_NULL}},
              {0,}}},

/* Level 1 Instruction Cache Reads */
{PAPI_L1_ICR, {DERIVED_ADD,
               {PNE_BGP_PU0_ICACHE_HIT,
                PNE_BGP_PU1_ICACHE_HIT,
                PNE_BGP_PU0_ICACHE_MISS,
                PNE_BGP_PU1_ICACHE_MISS,
                PAPI_NULL}},
              {0,}}},

/* Level 1 Instruction Cache Writes */
{PAPI_L1_ICW, {DERIVED_ADD,
               {PNE_BGP_PU0_ICACHE_LINEFILLINPROG,
                PNE_BGP_PU1_ICACHE_LINEFILLINPROG,
                PAPI_NULL}},
              {0,}}},

/* Level 1 Total Cache Hits */
{PAPI_L1_TCH, {DERIVED_ADD,
               {PNE_BGP_PU0_DCACHE_HIT,
                PNE_BGP_PU1_DCACHE_HIT,
                PNE_BGP_PU0_ICACHE_HIT,
                PNE_BGP_PU1_ICACHE_HIT,
                PAPI_NULL}},
              {0,}}},

/* Level 1 Total Cache Accesses */
{PAPI_L1_TCA, {DERIVED_ADD,
               {PNE_BGP_PU0_DCACHE_HIT,
                PNE_BGP_PU1_DCACHE_HIT,
                PNE_BGP_PU0_ICACHE_HIT,
                PNE_BGP_PU1_ICACHE_HIT,
                PNE_BGP_PU0_DCACHE_MISS,
                PNE_BGP_PU1_DCACHE_MISS,
                PAPI_NULL}},
              {0,}}},

```

```

        PNE_BGP_PU0_ICACHE_MISS,
        PNE_BGP_PU1_ICACHE_MISS,
        PNE_BGP_PU0_DCACHE_LINEFILLINPROG,
        PNE_BGP_PU1_DCACHE_LINEFILLINPROG,
        PAPI_NULL},
    {0,}},

/* Level 1 Total Cache Reads */
{PAPI_L1_TCR, {DERIVED_ADD,
    {PNE_BGP_PU0_DCACHE_HIT,
    PNE_BGP_PU1_DCACHE_HIT,
    PNE_BGP_PU0_ICACHE_HIT,
    PNE_BGP_PU1_ICACHE_HIT,
    PNE_BGP_PU0_DCACHE_MISS,
    PNE_BGP_PU1_DCACHE_MISS,
    PNE_BGP_PU0_ICACHE_MISS,
    PNE_BGP_PU1_ICACHE_MISS,
    PAPI_NULL},
    {0,}}},

/* Level 1 Total Cache Writes */
{PAPI_L1_TCW, {DERIVED_ADD,
    {PNE_BGP_PU0_DCACHE_LINEFILLINPROG,
    PNE_BGP_PU1_DCACHE_LINEFILLINPROG,
    PNE_BGP_PU0_ICACHE_LINEFILLINPROG,
    PNE_BGP_PU1_ICACHE_LINEFILLINPROG,
    PAPI_NULL},
    {0,}}},

/* Floating Point Operations */
{PAPI_FP_OPS, {DERIVED_POSTFIX,
    {PNE_BGP_PU0_FPU_ADD_SUB_1,
    PNE_BGP_PU1_FPU_ADD_SUB_1,
    PNE_BGP_PU0_FPU_MULT_1,
    PNE_BGP_PU1_FPU_MULT_1,
    PNE_BGP_PU0_FPU_FMA_2,
    PNE_BGP_PU0_FPU_MULT_1,
    PNE_BGP_PU1_FPU_MULT_1,
    PNE_BGP_PU0_FPU_FMA_2,
    PNE_BGP_PU1_FPU_FMA_2,
    PNE_BGP_PU0_FPU_DIV_1,
    PNE_BGP_PU1_FPU_DIV_1,
    PNE_BGP_PU0_FPU_OTHER_NON_STORAGE_OPS,
    PNE_BGP_PU1_FPU_OTHER_NON_STORAGE_OPS,
    PNE_BGP_PU0_FPU_ADD_SUB_2,
    PNE_BGP_PU1_FPU_ADD_SUB_2,
    PNE_BGP_PU0_FPU_MULT_2,
    PNE_BGP_PU1_FPU_MULT_2,
    PNE_BGP_PU0_FPU_FMA_4,
    PNE_BGP_PU1_FPU_FMA_4,
    PNE_BGP_PU0_FPU_DUAL_PIPE_OTHER_NON_STORAGE_OPS,
    PNE_BGP_PU1_FPU_DUAL_PIPE_OTHER_NON_STORAGE_OPS,
    PAPI_NULL},

```

```
{ "N0|N1|+|N2|+|N3|+|N4|2|*|+|N5|2|*|+|N6|13|*|+|N7|13|*|+|N8|+|N9|+|N10|2|*|+|N11|2|*|+|N12|2|*|+|N13|2|*|+|N14|4|*|+|N15|4|*|+|N16|2|*|+|N17|2|*|+|"}},
```

The following PAPI presets return accurate results for applications running in all modes (see Example 1-4). Events from cores 0, 1, 2 and 3 will be counted.

Example 1-4 Presets for any processing mode

```
/* Level 2 Data Cache Misses */
{PAPI_L2_DCM, {DERIVED_POSTFIX,
  {PNE_BGP_PU0_L2_PREFETCHABLE_REQUESTS,
   PNE_BGP_PU1_L2_PREFETCHABLE_REQUESTS,
   PNE_BGP_PU2_L2_PREFETCHABLE_REQUESTS,
   PNE_BGP_PU3_L2_PREFETCHABLE_REQUESTS,
   PNE_BGP_PU0_L2_PREFETCH_HITS_IN_STREAM,
   PNE_BGP_PU1_L2_PREFETCH_HITS_IN_STREAM,
   PNE_BGP_PU2_L2_PREFETCH_HITS_IN_STREAM,
   PNE_BGP_PU3_L2_PREFETCH_HITS_IN_STREAM,
   PAPI_NULL},
  {"N0|N1|+|N2|+|N3|+|N4|-|N5|-|N6|-|N7|-|"}},

/* Level 3 Load Misses */
{PAPI_L3_LDM, {DERIVED_ADD,
  {PNE_BGP_L3_MO_RDO_DIRO_MISS_OR_LOCKDOWN,
   PNE_BGP_L3_MO_RDO_DIR1_MISS_OR_LOCKDOWN,
   PNE_BGP_L3_M1_RDO_DIRO_MISS_OR_LOCKDOWN,
   PNE_BGP_L3_M1_RDO_DIR1_MISS_OR_LOCKDOWN,
   PNE_BGP_L3_MO_RD1_DIRO_MISS_OR_LOCKDOWN,
   PNE_BGP_L3_MO_RD1_DIR1_MISS_OR_LOCKDOWN,
   PNE_BGP_L3_M1_RD1_DIRO_MISS_OR_LOCKDOWN,
   PNE_BGP_L3_M1_RD1_DIR1_MISS_OR_LOCKDOWN,
   PNE_BGP_L3_MO_R2_DIRO_MISS_OR_LOCKDOWN,
   PNE_BGP_L3_MO_R2_DIR1_MISS_OR_LOCKDOWN,
   PNE_BGP_L3_M1_R2_DIRO_MISS_OR_LOCKDOWN,
   PNE_BGP_L3_M1_R2_DIR1_MISS_OR_LOCKDOWN,
   PAPI_NULL},
  {0,}}},

/* Total Cycles NOTE: This value is for the time the counters are active,
 * and not for the total cycles for the job. */
{PAPI_TOT_CYC, {NOT_DERIVED,
  {PNE_BGP_MISC_ELAPSED_TIME,
   PAPI_NULL,
   PAPI_NULL},
  {0,}}},

/* Level 2 Data Cache Hit */
{PAPI_L2_DCH, {DERIVED_ADD,
  {PNE_BGP_PU0_L2_PREFETCH_HITS_IN_STREAM,
   PNE_BGP_PU1_L2_PREFETCH_HITS_IN_STREAM,
   PNE_BGP_PU2_L2_PREFETCH_HITS_IN_STREAM,
   PNE_BGP_PU3_L2_PREFETCH_HITS_IN_STREAM,
   PAPI_NULL},
  {0,}}},
```

```

/* Level 2 Data Cache Access */
{PAPI_L2_DCA, {DERIVED_ADD,
               {PNE_BGP_PU0_L2_PREFETCHABLE_REQUESTS,
                PNE_BGP_PU1_L2_PREFETCHABLE_REQUESTS,
                PNE_BGP_PU2_L2_PREFETCHABLE_REQUESTS,
                PNE_BGP_PU3_L2_PREFETCHABLE_REQUESTS,
                PNE_BGP_PU0_L2_MEMORY_WRITES,
                PNE_BGP_PU1_L2_MEMORY_WRITES,
                PNE_BGP_PU2_L2_MEMORY_WRITES,
                PNE_BGP_PU3_L2_MEMORY_WRITES,
                PAPI_NULL},
               {0,}}},

/* Level 2 Data Cache Read */
{PAPI_L2_DCR, {DERIVED_ADD,
               {PNE_BGP_PU0_L2_PREFETCHABLE_REQUESTS,
                PNE_BGP_PU1_L2_PREFETCHABLE_REQUESTS,
                PNE_BGP_PU2_L2_PREFETCHABLE_REQUESTS,
                PNE_BGP_PU3_L2_PREFETCHABLE_REQUESTS,
                PAPI_NULL},
               {0,}}},

/* Level 2 Data Cache Write */
{PAPI_L2_DCW, {DERIVED_ADD,
               {PNE_BGP_PU0_L2_MEMORY_WRITES,
                PNE_BGP_PU1_L2_MEMORY_WRITES,
                PNE_BGP_PU2_L2_MEMORY_WRITES,
                PNE_BGP_PU3_L2_MEMORY_WRITES,
                PAPI_NULL},
               {0,}}},

/* Level 3 Total Cache Accesses */
{PAPI_L3_TCA, {DERIVED_ADD,
               {PNE_BGP_L3_MO_RD0_SINGLE_LINE_DELIVERED_L2,
                PNE_BGP_L3_MO_RD1_SINGLE_LINE_DELIVERED_L2,
                PNE_BGP_L3_MO_R2_SINGLE_LINE_DELIVERED_L2,
                PNE_BGP_L3_M1_RD0_SINGLE_LINE_DELIVERED_L2,
                PNE_BGP_L3_M1_RD1_SINGLE_LINE_DELIVERED_L2,
                PNE_BGP_L3_M1_R2_SINGLE_LINE_DELIVERED_L2,
                PNE_BGP_L3_MO_RD0_BURST_DELIVERED_L2,
                PNE_BGP_L3_MO_RD1_BURST_DELIVERED_L2,
                PNE_BGP_L3_MO_R2_BURST_DELIVERED_L2,
                PNE_BGP_L3_M1_RD0_BURST_DELIVERED_L2,
                PNE_BGP_L3_M1_RD1_BURST_DELIVERED_L2,
                PNE_BGP_L3_M1_R2_BURST_DELIVERED_L2,
                BGP_L3_MO_W0_DEPOSIT_REQUESTS,
                BGP_L3_MO_W1_DEPOSIT_REQUESTS,
                BGP_L3_M1_W0_DEPOSIT_REQUESTS,
                BGP_L3_M1_W1_DEPOSIT_REQUESTS,
                PAPI_NULL},
               {0,}}},

/* Level 3 Total Cache Reads */
{PAPI_L3_TCR, {DERIVED_ADD,
               {PNE_BGP_L3_MO_RD0_SINGLE_LINE_DELIVERED_L2,

```

```

        PNE_BGP_L3_MO_RD1_SINGLE_LINE_DELIVERED_L2,
        PNE_BGP_L3_MO_R2_SINGLE_LINE_DELIVERED_L2,
        PNE_BGP_L3_M1_RD0_SINGLE_LINE_DELIVERED_L2,
        PNE_BGP_L3_M1_RD1_SINGLE_LINE_DELIVERED_L2,
        PNE_BGP_L3_M1_R2_SINGLE_LINE_DELIVERED_L2,
        PNE_BGP_L3_MO_RD0_BURST_DELIVERED_L2,
        PNE_BGP_L3_MO_RD1_BURST_DELIVERED_L2,
        PNE_BGP_L3_MO_R2_BURST_DELIVERED_L2,
        PNE_BGP_L3_M1_RD0_BURST_DELIVERED_L2,
        PNE_BGP_L3_M1_RD1_BURST_DELIVERED_L2,
        PNE_BGP_L3_M1_R2_BURST_DELIVERED_L2,
        PAPI_NULL},
    {0,}},

/* Level 3 Total Cache Writes */
{PAPI_L3_TCW, {DERIVED_ADD,
    {PNE_BGP_L3_MO_W0_DEPOSIT_REQUESTS,
    PNE_BGP_L3_MO_W1_DEPOSIT_REQUESTS,
    PNE_BGP_L3_M1_W0_DEPOSIT_REQUESTS,
    PNE_BGP_L3_M1_W1_DEPOSIT_REQUESTS,
    PAPI_NULL},
    {0,}}},

/* Torus 32B Chunks Sent */
{PAPI_BGP_TS_32B, {DERIVED_ADD,
    {PNE_BGP_TORUS_XP_32BCHUNKS,
    PNE_BGP_TORUS_XM_32BCHUNKS,
    PNE_BGP_TORUS_YP_32BCHUNKS,
    PNE_BGP_TORUS_YM_32BCHUNKS,
    PNE_BGP_TORUS_ZP_32BCHUNKS,
    PNE_BGP_TORUS_ZM_32BCHUNKS,
    PAPI_NULL},
    {0,}}},

/* Torus Packets Sent */
{PAPI_BGP_TS_DPKT, {DERIVED_ADD,
    {PNE_BGP_TORUS_XP_PACKETS,
    PNE_BGP_TORUS_XM_PACKETS,
    PNE_BGP_TORUS_YP_PACKETS,
    PNE_BGP_TORUS_YM_PACKETS,
    PNE_BGP_TORUS_ZP_PACKETS,
    PNE_BGP_TORUS_ZM_PACKETS,
    PAPI_NULL},
    {0,}}},

/* PAPI Null */
{0, {0, {PAPI_NULL, PAPI_NULL}, {0,}}}

```

Through the PAPI native event mechanism, any event that is available in the universal performance counter (UPC) counters can be programmed and controlled through PAPI. A native event is handled in the same way as the PAPI predefined events and passed through the same API calls. A complete listing of available events can be found in Table 6-3 on page 80.

1.2.4 Limitations

Due to the lack of operating system support and the nature of the intended use of the Blue Gene/P machine, the `PAPI_overflow()` function has not been implemented.

There is no notion of virtual CPU time in the Compute Node Kernel. For this reason, both `PAPI_get_real_cyc()` and `PAPI_get_virt_cyc()` are mapped to the CPU time base register. For the same reason, `PAPI_get_real_usec()` and `PAPI_get_virt_usec()` report the same amount of elapsed time.



PAPI usage

This chapter provides an example of PAPI usage. The following topics are included in this chapter:

- ▶ Headers that need to be included when running PAPI.
- ▶ How to initialize PAPI.
- ▶ Add events (both preset events and native events).
- ▶ Start PAPI and stop PAPI.
- ▶ Print the counters.

2.1 Usage example

Example 2-1 shows the headers that need to be included when linking to the PAPI library, how to initialize PAPI from within your application, add events that will collect performance data (both PAPI preset events and Blue Gene/P native events), start PAPI, stop PAPI, and print the counter data.

Example 2-1 PAPI usage sample code

```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>

#include <common/alignment.h>

#include "papi.h" //Provides definitions for base PAPI function
#include <spi/bgp_SPI.h> //The Blue Gene/P substrate for PAPI is based on the UPC
interfaces documented in this header file.
#include "papiStdEventDefs.h" //Provides a listing of all standard PAPI presets.
Please note that not all of these are available on the Blue Gene/P platform
#include "linux-bgp-native-events.h" //Provides a listing of all available
counters native to Blue Gene.

void FPUArith(void); //This method does various calculations which should saturate
many of the counters

void List_PAPI_Events(const int pEventSet, int* pEvents, int* xNumEvents);
void Print_Native_Counters();
void Print_Native_Counters_via_Buffer(const BGP_UPC_Read_Counters_Struct_t*
pBuffer);
void Print_Native_Counters_for_PAPI_Counters(const int pEventSet);
void Print_Native_Counters_for_PAPI_Counters_From_List(const int* pEvents, const
int pNumEvents);
void Print_PAPI_Counters(const int pEventSet, const long long* pCounters);
void Print_PAPI_Counters_From_List(const int* pEventList, const int pNumEvents,
const long long* pCounters);
void Print_Counters(const int pEventSet);
void Print_PAPI_Events(const int pEventSet);

int PAPI_Events[256];
long long PAPI_Counters[256];

int main(int argc, char * argv[]) {
    int i;
    int xEventSet=PAPI_NULL;

    for (i = 0; i < 255; i++)
        PAPI_Counters[i]=0;

    PAPI_library_init(PAPI_VER_CURRENT);

    BGP_UPC_Initialize_Counter_Config(BGP_UPC_MODE_DEFAULT,
BGP_UPC_CFG_EDGE_DEFAULT);
```



```

        RC = PAPI_create_eventset(&xEventSet);
    RC = PAPI_add_event(xEventSet, PAPI_L1_DCM);
    RC = PAPI_add_event(xEventSet, PNE_BGP_PU1_DCACHE_HIT);

    RC = PAPI_start(xEventSet);

    FPUArith();

    RC = PAPI_stop(xEventSet, PAPI_Counters);

    RC = PAPI_read(xEventSet, PAPI_Counters);

    Print_Counters(xEventSet);
}

```

2.1.1 Headers

Table 2-1 contains a list of the headers that are included in Example 2-1 on page 16.

Table 2-1 Headers

Header	Purpose
papi.h	Provides definitions for base PAPI functions.
spi/bgp_SPI.h	The Blue Gene/P substrate for PAPI is based on the UPC interfaces documented in the header file.
papiStdEventDefs.h	Provides a listing of all standard PAPI presets. Note that all of these presets are not available on Blue Gene/P.
linux-bgp-native-events.h	Provides a listing of all available counter native to Blue Gene.

2.1.2 Flow of events

The flow of events (when coding for SMP mode) for Example 2-1 on page 16 is as follows:

1. `PAPI_library_init` is the first PAPI call made in this code, and must be called from all nodes. This will initialize the BGP UPC environment so that other calls to the UPC interface can be made.
2. `BGP_UPC_Initialize_Counter_Config` is used to set both the user mode and the trigger for counting. Detailed information about the various modes and triggers that are available are documented in the `/bgsys/drivers/ppcfloor/arch/include/UPC.h` header file. One point of interest is that when running with the default user mode (mode 0), only events on cores 0 and 1 of a node will be counted. So, in order to get counter data from cores 2 and 3, it would be necessary to run this code again with the user mode set to *mode 1* (which counts activity on cores 2 and 3) More information about mode 1 is located in 2.2.1, "Using PAPI preset values" on page 19.
3. Before calling `PAPI_start` and running your program code, an *Event Set* should be created with `PAPI_create_eventset`. Events that you want monitored should be added with the `PAPI_add_event` method.

The example shows how to add a PAPI preset event. A preset event essentially collects various Blue Gene/P hardware counters to derive a value for some pre-determined event. In this case, the PAPI_L1_DCM preset, which counts L1 cache misses, is used. If there is not a PAPI preset available for the event you would like to count, you can add any Blue Gene/P native counter to the event set using the same interface. In our example, the native counter PNE_BGP_PU1_DCACHE_HIT, which counts cache hits on core 1, was added. A full listing of available PAPI presets can be found in Example 1-3 on page 7. The complete list of Blue Gene/P Native events is located in Table 6-3 on page 80.

4. PAPI_start will start the UPC unit. Once this call is made, counters will be active.
5. FPUArith simply does an assortment of Floating Point operations to exercise the counters. You would want to replace this call with whatever program you want to get counter data from.
6. PAPI_stop should be called once your *work* is done. You should follow this up with a PAPI_read, which will access the UPC registers and retrieve the values for each event in your event set.
7. Print_Counters is a custom method used to format and print the counters. You could do this any way you like, but some sample code is provided in “Additional methods” on page 22.

2.1.3 Running the code

This section will outline the steps to compile and run the code in Example 2-1 on page 16.

The code can be compiled with the following command:

```
/bgsys/drivers/ppcfloor/comm/bin/mpicc -I/bgusr/yourLib/PAPIinstall_ver3/include
-I/bgsys/drivers/ppcfloor/arch/include -o papi_redbook papi_redbook.c
-L/bgsys/drivers/ppcfloor/runtime/SPI -lSPI.cna
-L/bgusr/yourLib/PAPIinstall_ver3/lib -lpapi
```

Replace /bgusr/yourLib/PAPIinstall_ver3 with the path of the PAPI library you compiled and installed, as outlined in Chapter 1, “Implementing the Performance Application Programming Interface on Blue Gene/P” on page 3.

You can run the code using a command similar to this:

```
mpirun -mode smp -exe ./papi_redbook -partition R00-M0-N13-J00 -cwd
/bgusr/yourLib/PAPIpatched_ver3/papi-c-3.9.0/src/ctests/bgp/ -np 1
```

The output should be similar to that shown in Example 2-2.

Example 2-2 Sample code output

```
yourSystem:/bgusr/yourLib/PAPIpatched_ver3/papi-c-3.9.0/src/ctests/bgp> mpirun
-mode smp -exe ./papi_redbook -partition R00-M0-N13-J00 -cwd
/bgusr/yourLib/PAPIpatched_ver3/papi-c-3.9.0/src/ctests/bgp/ -nofree -np 1
```

```
==> Start: Performing arithmetic...
```

```
==> End: Performing arithmetic...
```

```
***** Start Print Counter Values *****
```

```
***** Start Print of Native Counter Values for PAPI Counters *****
```

```
*** PAPI Counter Location 000: 0x80000000 PAPI_L1_DCM
39585 15 BGP_PUO_DCACHE_MISS
```

7 50 BGP_PU1_DCACHE_MISS

```
*** PAPI Counter Location 001: 0x40000033 PNE_BGP_PU1_DCACHE_HIT
    197181      51 BGP_PU1_DCACHE_HIT
***** End Print of Native Counter Values for PAPI Counters *****

***** Start Print of PAPI Counter Values *****
Number of Counters = 2
    Calculated Value Location Event Number Event Name
-----
-----
          37916      0 0x80000000 PAPI_L1_DCM
          195792     1 0x40000033 PNE_BGP_PU1_DCACHE_HIT
***** End Print of PAPI Counter Values *****

***** End Print Counter Values *****
```

2.2 Suggestions

This section includes some tips for modifying or creating your own PAPI presets, counting events on cores 2 and 3, and running in Virtual Node mode.

2.2.1 Using PAPI preset values

In our code sample in Example 2-1 on page 16, find the following statement:

```
BGP_UPC_Initialize_Counter_Config(BGP_UPC_MODE_DEFAULT, BGP_UPC_CFG_EDGE_DEFAULT);
```

The default counter mode is mode 0, which counts events on cores 0 and 1. By changing the counter configuration to run in mode 1, you can have the UPC unit count events on cores 2 and 3. The new call would like this:

```
BGP_UPC_Initialize_Counter_Config(1, BGP_UPC_CFG_EDGE_DEFAULT);
```

Simply changing the counter mode will not be enough. As outlined in 1.2.3, “PAPI presets” on page 6, many of the PAPI presets only include counters from cores 0 and 1. Take the following preset as an example, which counts cache misses on cores 0 and 1:

```
{PAPI_L1_DCM, {DERIVED_ADD,
               {PNE_BGP_PU0_DCACHE_MISS,
                PNE_BGP_PU1_DCACHE_MISS,
                PAPI_NULL},
               {0,}}},
```

If you were to set the counter config to mode1 and add this preset to the event set, you would not automatically get the cache misses for cores 2 and 3. The easiest solution to this problem is to add a new PAPI preset to the PAPI preset list, then rebuild your PAPI library.

Referring back to the PAPI preset list for Blue Gene/P (Example 1-3 on page 7), find the path for the patched PAPI code that you created following the install instructions in 1.1, “Installing PAPI” on page 4. Go to the source directory and open the linux-bgp-preset-events.c file. This is where you can add the new preset. Looking at the example above, you notice that the two native Blue Gene/P counters that are being used are PNE_BGP_PU0_DCACHE_MISS and PNE_BGP_PU1_DCACHE_MISS, and that the values are being added together to give the value of this event (as signalled by the DERIVED_ADD tag). In Table 6-3 on page 80, you can

find a description of these native counters. PU0 and PU1 from the description tells you which cores these events are being counted for (0 and 1). Also, both of these counters are in *group 0*, which means they are only available when running with a counter configuration of mode 0. In order to get readings for cores 2 and 3, you need to find equivalent counters in the table in the group 1 section. Searching the table, locate the BGP_PU2_DCACHE_MISS and BGP_PU3_DCACHE_MISS counters, which can be used to make this new PAPI preset. The end result will look something like this:

```
{PAPI_L1_DCM_MODE_1, {DERIVED_ADD,
                      {PNE_BGP_PU2_DCACHE_MISS,
                       PNE_BGP_PU3_DCACHE_MISS,
                       PAPI_NULL}},
 {0,}}},
```

Once your library is recompiled, you can add this preset to your event set and run it.

2.2.2 Running in Dual or Virtual Node mode

PAPI_library_init must still be called from all nodes to initialize the BGP UPC environment.

Once initialization is complete, only one rank per node should set the counter config values, create the event set, and start the counters. This can be accomplished through MPI barriers and a Blue Gene/P kernel interface call.

Additionally, once your *work* is complete, only one rank per node should stop the counters, retrieve the counters, and print (if necessary).

A modified version (Example 2-3) of the main procedure provided above gives an idea of how to do this task.

Example 2-3 Running in Dual or Virtual Node mode

```
#include <spi/kernel_interface.h>
#include <mpi.h>
#include <common/bgp_personality.h>

void get_compute_node_id(int * node_number)
{
    int tx, ty, tz;
    int nx, ny, nz;
    _BGP_Personality_t personality;

    Kernel_GetPersonality(&personality, sizeof(personality));

    tx = personality.Network_Config.Xcoord;
    ty = personality.Network_Config.Ycoord;
    tz = personality.Network_Config.Zcoord;

    nx = personality.Network_Config.Xnodes;
    ny = personality.Network_Config.Ynodes;
    nz = personality.Network_Config.Znodes;

    *node_number = tx + ty*nx + tz*nx*ny;
}

void get_local_rank(int * local_rank)
{
```

```

    int node_id, procid;
    MPI_Comm node_comm;

    get_compute_node_id(&node_id);

    procid = Kernel_PhysicalProcessorID();

    PMPI_Comm_split(MPI_COMM_WORLD, node_id, procid, &node_comm);

    PMPI_Comm_rank(node_comm, local_rank);
}

int main(int argc, char * argv[]) {
    int i;
    int xEventSet=PAPI_NULL;
    int local_rank;

    for (i = 0; i < 255; i++)
        PAPI_Counters[i]=0;

    PAPI_library_init(PAPI_VER_CURRENT);

    get_local_rank(&local_rank);

    if (local_rank == 0) {

        BGP_UPC_Initialize_Counter_Config(BGP_UPC_MODE_DEFAULT,
        BGP_UPC_CFG_EDGE_DEFAULT);

        RC = PAPI_create_eventset(&xEventSet);
        RC = PAPI_add_event(xEventSet, PAPI_L1_DCM);
        RC = PAPI_add_event(xEventSet, PNE_BGP_PU1_DCACHE_HIT);

        RC = PAPI_start(xEventSet);
    }

    PMPI_Barrier(MPI_COMM_WORLD);

    FPUArith();

    PMPI_Barrier(MPI_COMM_WORLD);

    if (local_rank == 0) {

        RC = PAPI_stop(xEventSet, PAPI_Counters);

        RC = PAPI_read(xEventSet, PAPI_Counters);

        Print_Counters(xEventSet);
    }
}

```

2.2.3 Additional methods

Example 2-4 provides a sample of methods for Floating Point Arithmetic, and formatting and printing PAPI counter values.

Example 2-4 Sample methods

```
void FPUArith(void) {
    int i;

    printf("\n==> Start: Performing arithmetic...\n");

    double x[32] ALIGN_L3_CACHE;

    register unsigned int zero = 0;
    register double *x_p = &x[0];

    for ( i = 0; i < 32; i++ )
        x[i] = 1.0;

    // Single Hummer Instructions:

    #if 1

    asm volatile ("fabs      1,2");
    asm volatile ("fmr       1,2");
    asm volatile ("fnabs     1,2");
    asm volatile ("fneg      1,2");

    asm volatile ("fadd      1,2,3");
    asm volatile ("fadds     1,2,3");
    asm volatile ("fdiv      1,2,3");
    asm volatile ("fdivs     1,2,3");
    asm volatile ("fmul      1,2,3");
    asm volatile ("fmuls     1,2,3");
    asm volatile ("fres      1,2");
    asm volatile ("frsqrt    1,2");
    //asm volatile ("fsqrt    1,2");           // gives exception
    //asm volatile ("fsqrts   1,2");           // gives exception
    asm volatile ("fsub      1,2,3");
    asm volatile ("fsubs     1,2,3");

    asm volatile ("fmadd     3,4,5,6");
    asm volatile ("fmadds    3,4,5,6");
    asm volatile ("fmsub     3,4,5,6");
    asm volatile ("fmsubs    3,4,5,6");
    asm volatile ("fnmadd    3,4,5,6");
    asm volatile ("fnmadds   3,4,5,6");
    asm volatile ("fnmsub    3,4,5,6");
    asm volatile ("fnmsubs   3,4,5,6");

    //asm volatile ("fcfid    5,6");           // invalid instruction
    //asm volatile ("fctid    5,6");           // invalid instruction
    //asm volatile ("fctidz   5,6");           // invalid instruction
    asm volatile ("fctiw     5,6");
    asm volatile ("fctiwz    5,6");
```

```

asm volatile ("frsp      5,6");

asm volatile ("fcmpl   0,1,2");
asm volatile ("fcmpl   0,1,2");
asm volatile ("fscld   0,1,2,3");

#endif
#if 1

asm volatile("fpadd      9,10,11");
asm volatile("fpcmpd    9,10,11");

#endif

#if 1

asm volatile("fpmul     23,24,25");
asm volatile("fxmul     26, 27, 28");
asm volatile("fxpmul    28, 29, 30");
asm volatile("fxsmul    2, 3, 4");
#endif

#if 1

asm volatile("fpmadd    10,11,12,13");
asm volatile("fpmsub    18, 19, 20, 21");
asm volatile("fpmadd    26, 27, 28, 29");
asm volatile("fpnmsub   16,17,18,19");

asm volatile("fxmadd    10,11,12,13");
asm volatile("fxmsub    18, 19, 20, 21");
asm volatile("fxnmadd   26, 27, 28, 29");
asm volatile("fxnmsub   16,17,18,19");

asm volatile("fxcpmadd   10,11,12,13");
asm volatile("fxcpsub   18, 19, 20, 21");
asm volatile("fxcpnadd  26, 27, 28, 29");
asm volatile("fxcpnsub  16,17,18,19");

asm volatile("fxcsadd    10,11,12,13");
asm volatile("fxcsmsub  18, 19, 20, 21");
asm volatile("fxcsnadd  26, 27, 28, 29");
asm volatile("fxcsnmsub 16,17,18,19");

asm volatile("fxcpnpma   1,2,3,4");
asm volatile("fxcsnpma   5,6,7,8");
asm volatile("fxcpnsma   9,10,11,12");
asm volatile("fxcsnsma   3,4,5,6");

asm volatile("fxcxnpma   9,10,11,12");
asm volatile("fxcxnsma   8,9,10,11");
asm volatile("fxcxma     3,4,5,6");
asm volatile("fxcxnms   8,9,10,11");

```

```

#endif

#if 1

asm volatile("fpre          12, 13");
asm volatile("fprsrte      15, 16");
asm volatile("fpse1        17, 18, 19, 20");
asm volatile("fpctiw       1,2");
asm volatile("fpctiwz      3,4");
asm volatile("fprsp        5,6");
asm volatile("fscmp        1,2,3");
asm volatile("fpmr         1,2");
asm volatile("fpneg        1,2");
asm volatile("fpabs        1,2");
asm volatile("fpnabs       1,2");
asm volatile("fsmr         1,2");
asm volatile("fsneg        1,2");
asm volatile("fsabs        1,2");
asm volatile("fsnabs       1,2");
asm volatile("fxmr         1,2");
asm volatile("fsmfp        1,2");
asm volatile("fsmtp        1,2");

#endif

#if 1
asm volatile("lfdx          16,%0,%1" : "+b"(x_p) : "b"(zero));
asm volatile("lfdux        16,%0,%1" : "+b"(x_p) : "b"(zero));
asm volatile("lfsx         16,%0,%1" : "+b"(x_p) : "b"(zero));
asm volatile("lfsux        16,%0,%1" : "+b"(x_p) : "b"(zero));

asm volatile("lfsdx        16,%0,%1" : "+b"(x_p) : "b"(zero));
asm volatile("lfsdux       16,%0,%1" : "+b"(x_p) : "b"(zero));
asm volatile("lfssx        16,%0,%1" : "+b"(x_p) : "b"(zero));
asm volatile("lfssux       16,%0,%1" : "+b"(x_p) : "b"(zero));

asm volatile("lfpsx        16,%0,%1" : "+b"(x_p) : "b"(zero));
asm volatile("lfpsux       16,%0,%1" : "+b"(x_p) : "b"(zero));
asm volatile("lfxsx        16,%0,%1" : "+b"(x_p) : "b"(zero));
asm volatile("lfxsux       16,%0,%1" : "+b"(x_p) : "b"(zero));
#endif

#if 1
asm volatile("lfpdx        16,%0,%1" : "+b"(x_p) : "b"(zero));
asm volatile("lfpdux       16,%0,%1" : "+b"(x_p) : "b"(zero));
asm volatile("lfxdx        16,%0,%1" : "+b"(x_p) : "b"(zero));
asm volatile("lfxdux       16,%0,%1" : "+b"(x_p) : "b"(zero));
#endif

#if 1
asm volatile("stfdx        16,%0,%1" : "+b"(x_p) : "b"(zero));
asm volatile("stfdux       16,%0,%1" : "+b"(x_p) : "b"(zero));
asm volatile("stfsx        16,%0,%1" : "+b"(x_p) : "b"(zero));
asm volatile("stfsux       16,%0,%1" : "+b"(x_p) : "b"(zero));

```



```

asm volatile("stfsdx      16,%0,%1" : "+b"(x_p) : "b"(zero));
asm volatile("stfsdux    16,%0,%1" : "+b"(x_p) : "b"(zero));
asm volatile("stfssx     16,%0,%1" : "+b"(x_p) : "b"(zero));
//asm volatile("stfssux   16,%0,%1" : "+b"(x_p) : "b"(zero));

asm volatile("stfpsx     16,%0,%1" : "+b"(x_p) : "b"(zero));
asm volatile("stfpsux    16,%0,%1" : "+b"(x_p) : "b"(zero));
asm volatile("stfjsx     16,%0,%1" : "+b"(x_p) : "b"(zero));
asm volatile("stfjsxux   16,%0,%1" : "+b"(x_p) : "b"(zero));
#endif

#if 1
asm volatile("stfpdx     16,%0,%1" : "+b"(x_p) : "b"(zero));
asm volatile("stfpdux    16,%0,%1" : "+b"(x_p) : "b"(zero));
asm volatile("stfxdx     16,%0,%1" : "+b"(x_p) : "b"(zero));
asm volatile("stfxdux    16,%0,%1" : "+b"(x_p) : "b"(zero));
#endif
printf("==> End:   Performing arithmetic...\n");

return;
}

/* Print_Counters */
void Print_Counters(const int pEventSet) {
    printf("\n***** Start Print Counter Values *****\n");
    //
    Print_Native_Counters_via_Buffer((BGP_UPC_Read_Counters_Struct_t*)Native_Buffer);
    // Print_Native_Counters();
    Print_Native_Counters_for_PAPI_Counters(pEventSet);
    Print_PAPI_Counters(pEventSet, PAPI_Counters);
    printf("\n***** End Print Counter Values *****\n");

    return;
}

/* Print_Native_Counters */

void Print_Native_Counters() {
    printf("\n***** Start Print of Native Counter Values *****\n");
    BGP_UPC_Print_Counter_Values(BGP_UPC_READ_EXCLUSIVE);
    printf("***** End Print of Native Counter Values *****\n");

    return;
}

/* Print_Native_Counters_for_PAPI_Counters */

void Print_Native_Counters_for_PAPI_Counters(const int pEventSet) {
    printf("\n***** Start Print of Native Counter Values for PAPI Counters
*****\n");
    int xNumEvents = PAPI_num_events(pEventSet);
    if (xNumEvents) {
        List_PAPI_Events(pEventSet, PAPI_Events, &xNumEvents);
        Print_Native_Counters_for_PAPI_Counters_From_List(PAPI_Events, xNumEvents);
    }
}

```

```

    }
    else {
        printf("No events are present in the event set.\n");
    }
    printf("***** End Print of Native Counter Values for PAPI Counters *****\n");

    return;
}

/* Print_Native_Counters_for_PAPI_Counters_From_List */
void Print_Native_Counters_for_PAPI_Counters_From_List(const int* pEvents, const
int pNumEvents) {
    int i, j, xRC;
    char xName[256];
    BGP_UPC_Event_Id_t xNativeEventId;
    PAPI_event_info_t xEventInfo;

// BGP_UPC_Print_Counter_Values(); // DLH
for (i=0; i<pNumEvents; i++) {
    xRC = PAPI_event_code_to_name(PAPI_Events[i], xName);
    if (!xRC) {
        xRC = PAPI_get_event_info(PAPI_Events[i], &xEventInfo);
        if (xRC) {
            printf("FAILURE: PAPI_get_event_info failed for %s, xRC=%d\n", xName,
xRC);
            exit(1);
        }
        printf("\n      *** PAPI Counter Location %3.3d: 0x%8.8x %s\n", i,
PAPI_Events[i], xName);
        if (PAPI_Events[i] & 0x80000000) {
            // Preset event
            for (j=0; j<xEventInfo.count; j++) {
                xNativeEventId = (BGP_UPC_Event_Id_t)(xEventInfo.code[j]&0xBFFFFFFF);
//                printf("Preset: j=%d, xEventInfo.code[j]=0x%8.8x,
xNativeEventId=0x%8.8x\n", j, xEventInfo.code[j], xNativeEventId);
                BGP_UPC_Print_Counter_Value(xNativeEventId, BGP_UPC_READ_EXCLUSIVE);
            }
        }
        else {
            // Native event
            xNativeEventId = (BGP_UPC_Event_Id_t)(PAPI_Events[i]&0xBFFFFFFF);
//            printf("Native: i=%d, PAPI_Events[i]=0x%8.8x,
xNativeEventId=0x%8.8x\n", i, PAPI_Events[i], xNativeEventId);
            BGP_UPC_Print_Counter_Value(xNativeEventId, BGP_UPC_READ_EXCLUSIVE);
        }
    }
    else {
        printf("\n      *** PAPI Counter Location %3.3d: Not mapped\n", i);
    }
}
}

/* Print_PAPI_Counters */

void Print_PAPI_Counters(const int pEventSet, const long long* pCounters) {

```

```

    int i;
    char xName[256];
    printf("\n***** Start Print of PAPI Counter Values *****\n");
    // printf("Print_PAPI_Counters: PAPI_Counters*=%p, pCounters*=%p\n",
PAPI_Counters, pCounters);
    int pNumEvents = PAPI_num_events(pEventSet);
    printf("Number of Counters = %d\n", pNumEvents);
    if (pNumEvents) {
        printf("    Calculated Value Location Event Number Event Name\n");
        printf("-----\n");
        List_PAPI_Events(pEventSet, PAPI_Events, &pNumEvents);
        for (i=0; i<pNumEvents; i++) {
            if (PAPI_event_code_to_name(PAPI_Events[i], xName)) {
                printf("PAPI_event_code_to_name failed on event code %d\n",
PAPI_Events[i]);
                exit(1);
            }
            printf("%20llu    %3d    0x%8.8x %s\n", pCounters[i], i, PAPI_Events[i],
xName);
        }
        printf("***** End Print of PAPI Counter Values *****\n");

    return;
}

/* Print_PAPI_Counters_From_List */
void Print_PAPI_Counters_From_List(const int* pEventList, const int pNumEvents,
const long long* pCounters) {
    int i;
    char xName[256];
    printf("\n***** Start Print of PAPI Counter Values *****\n");
    printf("Number of Counters = %d\n", pNumEvents);
    if (pNumEvents) {
        printf("    Calculated Value Location Event Number Event Name\n");
        printf("-----\n");
        for (i=0; i<pNumEvents; i++) {
            if (PAPI_event_code_to_name(pEventList[i], xName)) {
                printf("PAPI_event_code_to_name failed on event code %d\n",
pEventList[i]);
                exit(1);
            }
            printf("%20llu    %3d    0x%8.8x %s\n", pCounters[i], i, pEventList[i],
xName);
        }
        printf("***** End Print of PAPI Counter Values *****\n");

    return;
}

/* Print_PAPI_Events */

```

```

void Print_PAPI_Events(const int pEventSet) {
    int i;
    char xName[256];
    int pNumEvents = PAPI_num_events(pEventSet);
    List_PAPI_Events(pEventSet, PAPI_Events, &pNumEvents);
    for (i=0; i<pNumEvents; i++) {
        if (!PAPI_event_code_to_name(PAPI_Events[i], xName))
            printf("PAPI Counter Location %3.3d: 0x%8.8x %s\n", i, PAPI_Events[i],
xName);
        else
            printf("PAPI Counter Location %3.3d: Not mapped\n", i);
    }

    return;
}

/* List_PAPI_Events */
void List_PAPI_Events(const int pEventSet, int* pEvents, int* pNumEvents) {
    int xRC = PAPI_list_events(pEventSet, pEvents, pNumEvents);
    if (xRC != PAPI_OK) {
        printf("FAILURE: PAPI_list_events failed, returned xRC=%d...\n", xRC);
        exit(1);
    }

    return;
}

```



Automatically Available Performance Counters

Included with the Blue Gene/P V1R3M0 release is a feature that allows system administrators or individual users to automatically enable performance counters. This feature applies to all applications compiled with any of the *mpi** compile scripts located in the `/bgsys/drivers/ppcfloor/comm/bin` directory and also applications intended for use in High Throughput Computing (HTC) mode.

3.1 Automatically Available Performance Counters for MPI applications

By providing a hook into MPI_Init and MPI_Finalize functions, counters will be enabled before an application runs, and the results will be collected and summarized before the application exits. Once this feature is enabled, no user intervention will be required to collect this performance counter data, but options will be provided at run time to change counter modes, counter triggers, and counter data output directories. It is also possible to disable the collection of performance counter data at run time.

3.1.1 Enabling Automatically Available Performance Counters

Located in the /bgsys/drivers/ppcfloor/tools/AutoPerfCounters directory is an EnableAutoPerfCounters script. By sourcing this script, the following environment variables will be set:

- ▶ MPICC_PROFILE
- ▶ MPICXX_PROFILE
- ▶ MPIF77_PROFILE
- ▶ MPIF90_PROFILE

These environment variables are used by the compile scripts located in /bgsys/drivers/ppcfloor/comm/bin to enable a profiling option for all compiles. In this case, the profiling option will be set to link to the High Performance Monitor (HPM) library. The HPM library provides a wrapper for the MPI_Init function so that any time MPI_Init is called, performance counters will be enabled. It also provides a wrapper for the MPI_Finalize function so that any time MPI_Finalize is called, performance counters will be turned off, read for all nodes, and neatly summarized into a human readable and binary output file.

The recommended method for running would be to include the following line in all users' .bashrc files, but is left up to the discretion of the system administrator or individual users:

```
source  
/bgsys/drivers/ppcfloor/tools/AutoPerfCounters/EnableAutoPerfCountersHeaders
```

3.1.2 Compiling with Automatically Available Performance Counters

Once /bgsys/drivers/ppcfloor/tools/AutoPerfCounters/EnableAutoPerfCounters has been sourced, there is no additional user intervention required for compiles. Any compiles using the scripts in /bgsys/drivers/ppcfloor/comm/bin will link to the necessary library and include directories. Take, for example, the following compile:

```
> /bgsys/drivers/ppcfloor/comm/bin/mpicc -o loopy loopy.c
```

No modification must be made for the compile. The loopy application will now run with performance counters enabled.

3.1.3 How performance counters work

In 3.1.2, "Compiling with Automatically Available Performance Counters" on page 30, the loopy application was compiled. Example 3-1 on page 31 provides a look at loopy.c.

Example 3-1 loopy.c

```
/* ....Standard Includes.... */

#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include "mpi.h"

main(argc,argv)
int argc;
char *argv[];
{
    int MyRank,Numprocs,Source, Destination, iproc;
    int tag = 0,i=300;
    int Root = 0,row,col,count,Mess;
    float addme = 0.0;
    MPI_Status status;

    MPI_Init(&argc,&argv);
    MPI_Comm_rank(MPI_COMM_WORLD,&MyRank);
    MPI_Comm_size(MPI_COMM_WORLD,&Numprocs);

    i=0;
    for (i=0;i<100000;i++) {
        addme = addme * 11.052687;
    }

    MPI_Finalize();
    exit(0);
}
```

The code above is not complicated, but there are two points of interest. On line 18, a call to `MPI_Init` is made. At this point, performance counters will be turned on. The default configuration for the counters is:

```
Counter Mode: 0
Counter Trigger: BGP_UPC_CFG_LEVEL_HIGH;
```

Instructions for changing these modes are included in 3.1.4, “Changing runtime parameters” on page 32.

On line 27, a call to `MPI_Finalize` is made. At this point, performance counters will be turned off, performance counter data will be gathered from all nodes, and two output files will be written to the user's current working directory.

It is worth noting that any work done before `MPI_Init` or after `MPI_Finalize` will not be reflected in the performance counter data.

The above example could be run with the following command:

```
mpirun -partition R00-M0-N00-J00 -exe /bgusr/userId/aapc_apps/loopy -cwd
/bgusr/userId/
```

3.1.4 Changing runtime parameters

The following environment variables can be set at run time:

- ▶ TRACE_DIR
- ▶ BGP_COUNTER_MODE
- ▶ BGP_COUNTER_TRIGGER

TRACE_DIR changes the destination directory for output files. An example of its usage would be:

```
mpirun -partition R00-M0-N00-J00 -exe /bgusr/userId/aapc_apps/loopy -cwd /bgusr/userId/ -env "TRACE_DIR=/bgusr/userId/aapc_output"
```

In this case, the hpm_summary and hpm_data files will be placed in /bgusr/userId/aapc_output rather than the current working directory. If a directory that cannot be written to is specified, the application will not fail, but an error message will be printed and no output will be written.

BGP_COUNTER_MODE changes the counter mode for the UPC. Documentation for the various modes is located in the /bgsys/drivers/ppcfloor/arch/include/spi/UPC.h header file. An example of its usage would be:

```
mpirun -partition R00-M0-N00-J00 -exe /bgusr/userId/aapc_apps/loopy -cwd /bgusr/userId/ -env "BGP_COUNTER_MODE=1"
```

In this case, the counter mode will be set to 1 (instead of the default '0'). '0', '1', '2', and '3' are the valid inputs for this environment variable. If any other inputs are provided, the counter mode will default back to '0'.

BGP_COUNTER_TRIGGER changes the trigger for the UPC. Documentation for the various triggers is located in the /bgsys/drivers/ppcfloor/arch/include/spi/UPC.h header file. An example of its usage would be:

```
mpirun -partition R00-M0-N00-J00 -exe /bgusr/userId/aapc_apps/loopy -cwd /bgusr/userId/ -env "BGP_COUNTER_TRIGGER=BGP_UPC_CFG_EDGE_RISE"
```

In this case, the counter mode will be set to BGP_UPC_CFG_EDGE_RISE (instead of the default BGP_UPC_CFG_LEVEL_HIGH). BGP_UPC_CFG_LEVEL_HIGH, BGP_UPC_CFG_EDGE_RISE, BGP_UPC_CFG_EDGE_FALL, BGP_UPC_CFG_EDGE_DEFAULT, and BGP_UPC_CFG_LEVEL_LOW are the valid inputs for this environment variable. If any other inputs are provided, the counter mode will default back to BGP_UPC_CFG_LEVEL_HIGH.

3.2 Performance counters in HTC Mode

It is also possible to gather performance counter data in High Throughput Computing (HTC) mode. Unlike Automatically Available Performance Counters for MPI applications (covered in 3.1, “Automatically Available Performance Counters for MPI applications” on page 30), obtaining counter data will require minor modifications to application code and compile commands.

3.2.1 Modifications to application code

When using Automatically Available Performance Counters for MPI applications, a hook into the MPI_Initialize function (to start counters) and MPI_Finalize function (to stop counters and summarize the data) is used. Because MPI is not available when running applications in HTC mode, and MPI_Initialize and MPI_Finalize are not used, the user is required to insert an initialize routine at the point they would like to start counting (HPM_Init), and a finalize routine (HPM_Finalize) where they would like to stop counting and write out a summary of performance counter data:

```
HPM_Init(); //Counting will start when this routine is called.
HPM_Finalize(); //Counting will stop when this routine is called, and output files
will be written containing a summary of counter data.
```

It will also be necessary to include the header file that contains definitions for HPM_Init and HPM_Finalize:

```
#include "hpm.h"
```

Example 3-2 contains a simple program that has been instrumented for performance counters in HTC Mode.

Example 3-2 counter_test.c

```
//htc_counter_test.c

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include "hpm.h"

int main(int argc, char *argv[]) {
    int i, j;
    double *a;
    double *b;
    double c;
    unsigned len;

    len = 10000000;

    if ((a=malloc(len*sizeof(double))) == NULL) {
        fprintf(stderr, "error allocating vector a\n");
    }
    if ((b=malloc(len*sizeof(double))) == NULL) {
        fprintf(stderr, "error allocating vector b\n");
    }

    HPM_Init();

    for (i=0; i<len; i++) {
        a[i] = 1.0*(i+1);
        b[i] = 1.0/(i+1);
    }

    c = 0.0;
    for (i = 0; i < 4; i++) {
        for (j=0; j<len; j++) {
```

```

        c += a[j]*b[j];
    }
}

HPM_Finalize();

}

```

3.2.2 Compiling the application

When creating the object file, ensure that the directory that contains the `hpm.h` header file (added to your instrumented code in the previous step) as well as the `arch/include` directory is included:

```

/bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc-bgp-linux-gcc
-I/bgsys/drivers/ppcfloor/tools/AutoPerfCounters
-I/bgsys/drivers/ppcfloor/arch/include -c htc_counter_test.c

```

When creating the executable, you will need to link to the `libhtchpm.a` library located in the `/bgsys/drivers/ppcfloor/tools/AutoPerfCounters` directory. The `libhtchpm.a` library contains the code that will implement `HPM_Init` and `HPM_Finalize`. You will also need to link to the `libSPI.cna.a` library located in the `/bgsys/drivers/ppcfloor/runtime/SPI` directory, which contains all of the UPC calls that are used by `HPM_Init` and `HPM_Finalize`. The link order must be the same as it is in the example:

```

bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc-bgp-linux-gcc -o htc_counter_test
htc_counter_test.o -L/bgsys/drivers/ppcfloor/tools/AutoPerfCounters -lhtchpm
-L/bgsys/drivers/ppcfloor/runtime/SPI -lSPI.cna -lrt

```

3.2.3 Running the application

Applications can be run using the `submit` command:

```

> submit -exe htc_counter_test -mode smp -location R00-M0-N05-J15-C00

```

After the application has run, an output file will be written to the user's current working directory. The output file, titled `hpm_summary.<jobid>`, contains a human readable summary of counter information collected for that job. Part of an output file is shown in Example 3-3.

Example 3-3 Sample output

```

-----
BGP counter summary, mode = 0, trigger = level high, jobid = 2221571
-----
Elapsed time = -0.000 seconds.
Partiton shape = <4,2,2>.

Measured total floating-point operation count for cores 0 and 1: 230000004

Estimated aggregate GFlops = 0.1290,
    assuming cores 2 and 3 have the same total floating-point operation count
    as cores 0 and 1, and using a weighting factor of 13 ops for fdiv.

-----
counter          value  label
-----

```

```
0      90000337.0   BGP_PUO_JPIPE_INSTRUCTIONS
1      998015779.0   BGP_PUO_JPIPE_ADD_SUB
2           2271.0   BGP_PUO_JPIPE_LOGICAL_OPS
3      2032960306.0   BGP_PUO_JPIPE_SHROTMK
...
```

The user can change the destination of the output file by using the TRACE_DIR environment variable. For example, the following command would send output to the performance_counter_data directory instead of the current working directory:

```
> submit -exe htc_counter_test -mode smp -location R00-M0-N05-J15-C00 -env
"TRACE_DIR=/bgusr/userId/performance_counter_data"
```

3.2.4 Caveats when running in HTC mode

Each node in Blue Gene/P shares a performance counter unit. Because up to four separate applications can run on any given node simultaneously, the user must control what is running on their partition if they want to ensure *pure* counter values.

The user should always run their application on core 0 of a node when using performance counters. This can be done by specifying a location ending in C00 when using the submit command.

The user should ensure that no other applications are started on the node when gathering performance counter data.

Archived



Part 2

High Performance Computing Toolkit

This part discusses the High Performance Computing Toolkit that is available for Blue Gene/P. 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.

Archived

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 4-1 provides the library file names and their usage.

Table 4-1 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.

4.1 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

4.2 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 4-1 shows how to compile and link in C, using `mpicc`, which currently is based on the GNU compiler.

Example 4-1 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 4-2 shows how to compile and link a program in Fortran.

Example 4-2 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)
```

4.3 Environment variables

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

4.3.1 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 4.5, “Configuration” on page 48.

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)`

4.3.2 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`

4.3.3 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.

4.3.4 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).

4.3.5 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 you 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.

4.3.6 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:

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

Hops_i is the distance between the processors for MPI communication, and 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.

4.4 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 4.5, “Configuration” on page 48.

4.4.1 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 4-3 provides the function.

Example 4-3 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 4-4 shows an example of *mpi profile.0*.

Example 4-4 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
1   1048576.0      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

4.4.3 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 4-2.

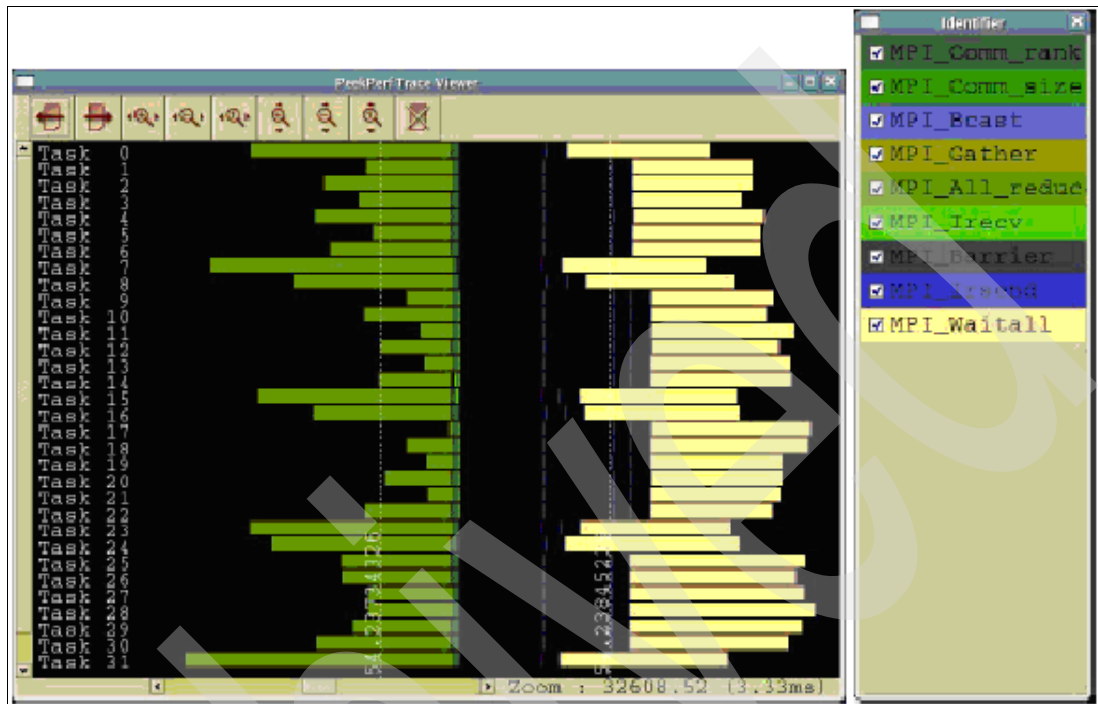


Figure 4-2 Single trace file in the Peekview utility

4.5 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.

4.5.1 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.

4.5.2 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 4-5, 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 4-5 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 4-6) 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 4-6 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 4-7) 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 4-7 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 4-8, 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 4-8 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 4-9) is used with the MT get_memoryinfo() utility function to provide memory usage information.

Example 4-9 MT_memorystruct data structure

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

4.5.3 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 4.5.3, "Utility functions" on page 51, 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 4-10, 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 4-10 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 */
}
```

4.6 Related issues

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

4.6.1 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 3×10^4 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 4.5, “Configuration” on page 48, for details.

4.6.2 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.

5.1 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 5-1.

Table 5-1 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	<pre>Xprofiler -e function1 -e function2 a.out gmon.out</pre>	<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	<pre>Xprofiler -E function1 -E function2 a.out gmon.out</pre>	<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 function1 -f function2 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 function1 -F function2 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>

5.2 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**.

5.2.1 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.

5.2.2 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 5.3, “Getting performance data for your application” on page 69.

Figure 5-1 on page 61 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.

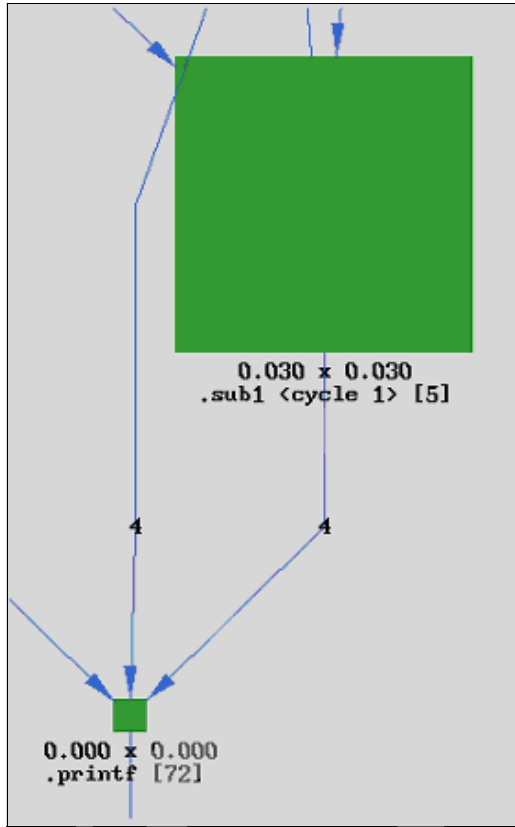


Figure 5-1 Sample functions

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.

5.2.3 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 5-2 on page 64). 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.

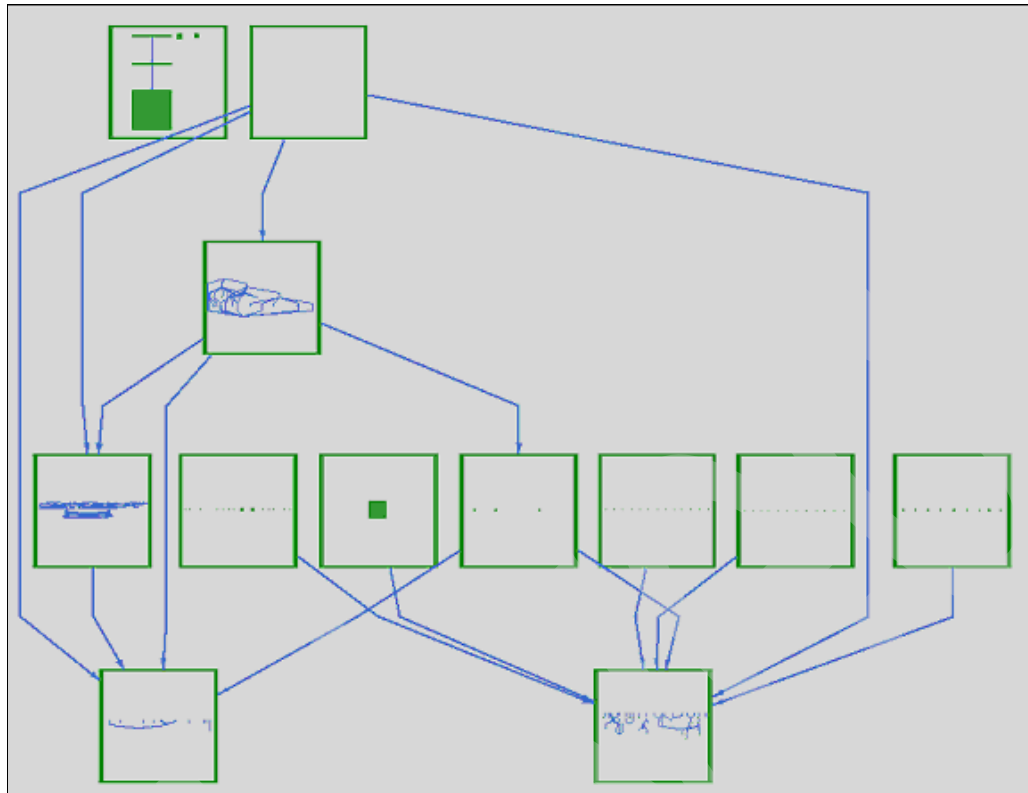


Figure 5-2 Xprofiler before filters

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 5-3).

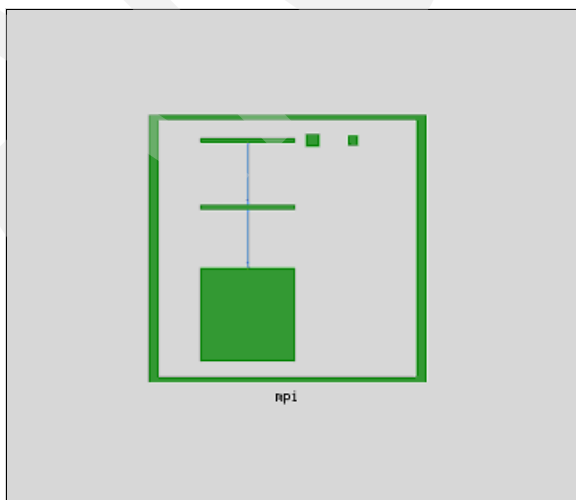


Figure 5-3 Xprofiler with library calls filtered

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 5-4). 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.

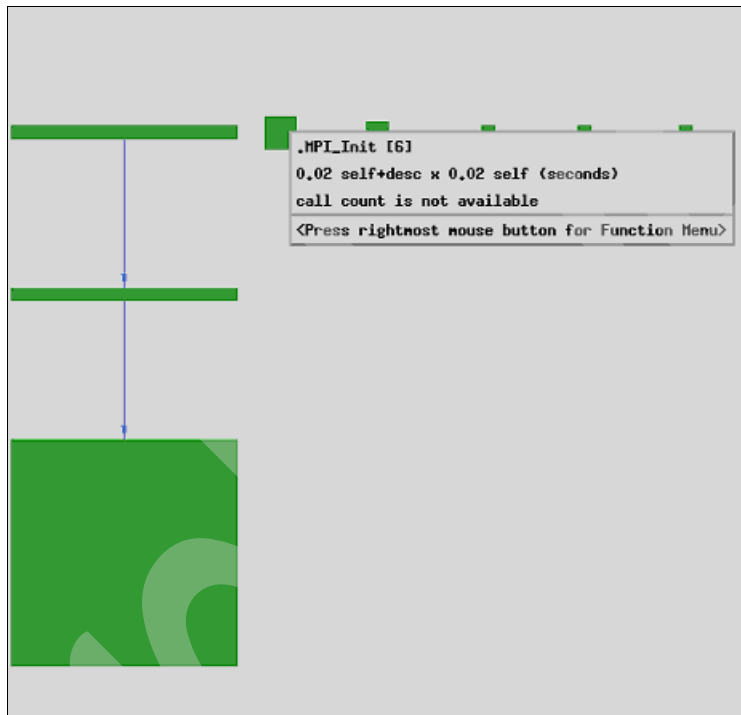


Figure 5-4 Xprofiler with the function box added

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.

5.3 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 5-5 shows the function box for the function `main` and its label.

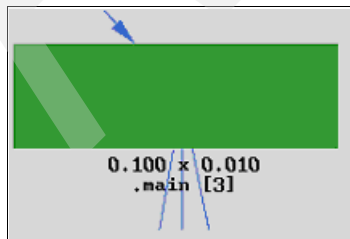


Figure 5-5 Function box

The label contains the following information:

- ▶ The name of the function, its associated cycle, if any, and its index
In the example in Figure 5-5 on page 69, 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 5-5 on page 69, 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 5-5 on page 69, 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 5-6. The label shows the number of calls that were made between the two functions (from caller to callee).

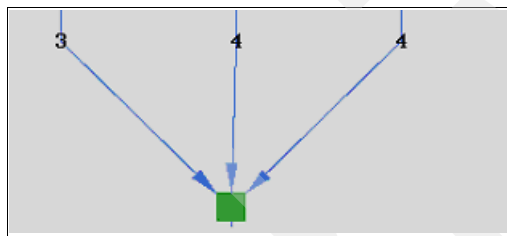


Figure 5-6 Call arc labels

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.

Archived



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.

6.1 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 6-1 shows a simple case of how an instrumented program might look.

Example 6-1 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 6-1 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 6.5, "Inclusive and exclusive values" on page 122.

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 `xlf_r`, 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.

6.2 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 6-1.

Table 6-1 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 6-3 on page 80 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 6-2 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 6-2.

Table 6-2 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 6-3 contains a list of the event names and groups on the Blue Gene/P system.

Table 6-3 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, fmul)

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, fmmadd, fmmadds, fnmsub, fnmsubs; one result generated per instruction, two flops)
0	60	BGP_PU1_FPU_DIV_1	PU1: PowerPC FP Div (fdiv, fdivs; 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, fcmpu, fcmpo, mffs, mcrrs, mtfsfi, mtfsf, mtfsb0, mtfsb1)
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	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, stfpx, stfpx, 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	54	BGP_PU1_DCACHE_OPS_UM2	PU1: D cache operations
2	55	BGP_PU1_ICACHE_MISS_UM2	PU1: Accesses to I cache that miss in I Cache
2	56	BGP_PU1_ICACHE_HIT_UM2	PU1: Accesses to I cache that hit in I Cache
2	57	BGP_PU1_FPU_ADD_SUB_1_UM2	PU1: PowerPC FP Add/Sub (fadd, fadds, fsub, fsubs)
2	58	BGP_PU1_FPU_MULT_1_UM2	PU1: PowerPC FP Mult (fmul, fmuls)
2	59	BGP_PU1_FPU_FMA_2_UM2	PU1: PowerPC FP FMA (fmadd, fmadds, fmsub, fmsubs, fnmadd, fnmadds, fnmsub, fnmsubs; one result generated per instruction, two flops)
2	60	BGP_PU1_FPU_DIV_1_UM2	PU1: PowerPC FP Div (fdiv, fdivs; Single Pipe Divide)
2	61	BGP_PU1_FPU_OTHER_NON_STORAGE_OPS_UM2	PU1: PowerPC FP remaining non-storage instructions (fabs, fnabs, frsp, fctiw, fctiwz, fres, frsqrte, fsel, fmr, fneg, fcmpu, fcmpo, mffs, mcrrs, mtfsfi, mtfsf, mtfsb0, mtfsb1)
2	62	BGP_PU1_FPU_ADD_SUB_2_UM2	PU1: Dual pipe Add/Sub (fpadd, fpsub)
2	63	BGP_PU1_FPU_MULT_2_UM2	PU1: Dual pipe Mult (fpmul, fxmul, fxpmul, fxsmul)
2	64	BGP_PU1_FPU_FMA_4_UM2	PU1: Dual pipe FMAs (fpmadd, fpmnadd, fpmsub, fpnmsub, fxmadd, fxnmadd, fxmsub, fxnmsub, fxcpmadd, fxcsmadd, fxcnmpadd, fxcnsmadd, fxcpsub, fxcnpsub, fxcnpsma, fxcnsma, fxcnpsma, fxcnsma, fxcnpsma, fxcnsma; two results generated per instruction, four flops)
2	65	BGP_PU1_FPU_DUAL_PIPE_OTHER_NON_STORAGE_OPS_UM2	PU1: Dual pipe remaining non-storage instructions (fpmr, fpneg, fsmr, fsneg, fxmr, fsmfp, fsmtp, fpabs, fpnabs, fsabs, fsnabs, fprsp, fpctiw, fpctiwz, fpre, fprsqrte, fpsel, fscmp)
2	66	BGP_PU1_FPU_QUADWORD_LOADS_UM2	PU1: Quad Word Loads (ffpdx, lfpdux, lfxdx, lfxdux)
2	67	BGP_PU1_FPU_OTHER_LOADS_UM2	PU1: All other Loads (lfs, lfsx, lfsu, lfsux, lfpsx, fpsux, lfsdx, lfsdux, lfssx, lfssux, lfd, lfdx, lfdu, lfdux, lfxsx, lfxsux)
2	68	BGP_PU1_FPU_QUADWORD_STORES_UM2	PU1: Quad Word Stores (stfpdx, stfpdux, stfxdx, stfxdux)
2	69	BGP_PU1_FPU_OTHER_STORES_UM2	PU1: All other FPU Stores (stfs, stfsx, stfsu, stfsux, stfd, stfdx, stfdu, stfdx, stfiwx, stfpx, stfpxsux, stfpiwx, stfsdx, stfsdux, stfssx, stfssux, stfxsx, stfxsux)

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	20	BGP_PU2_ICACHE_MISS_UM3	PU2: Accesses to I cache that miss in I Cache
3	21	BGP_PU2_ICACHE_HIT_UM3	PU2: Accesses to I cache that hit in I Cache
3	22	BGP_PU2_FPU_ADD_SUB_1_UM3	PU2: PowerPC FP Add/Sub (fadd, fadds, fsub, fsubs)
3	23	BGP_PU2_FPU_MULT_1_UM3	PU2: PowerPC FP Mult (fmul, fmuls)
3	24	BGP_PU2_FPU_FMA_2_UM3	PU2: PowerPC FP FMA (fmadd, fmadds, fmsub, fmsubs, fnmadd, fnmadds, fnmsub, fnmsubs; one result generated per instruction, two flops)
3	25	BGP_PU2_FPU_DIV_1_UM3	PU2: PowerPC FP Div (fdiv, fdivs; Single Pipe Divide)
3	26	BGP_PU2_FPU_OTHER_NON_STORAGE_OPS_UM3	PU2: PowerPC FP remaining non-storage instructions (fabs, fnabs, frsp, fctiw, fctiwz, fres, frsqtr, fsel, fmr, fneg, fcmpu, fcmpo, mffs, mcrrs, mtfsfi, mtfsf, mtfsb0, mtfsb1)
3	27	BGP_PU2_FPU_ADD_SUB_2_UM3	PU2: Dual pipe Add/Sub (fpadd, fpsub)
3	28	BGP_PU2_FPU_MULT_2_UM3	PU2: Dual pipe Mult (fpmul, fxmul, fxpmul, fxsmul)
3	29	BGP_PU2_FPU_FMA_4_UM3	PU2: Dual pipe FMAs (fpmadd, fpmnadd, fpmsub, fpnmsub, fxmadd, fxnmadd, fxmsub, fxnmsub, fxcpmadd, fxcsnadd, fxcpsub, fxcsnsub, fxcpsnsub, fxcpsnsub, fxcpsnma, fxcpsnma, fxcpsnsma, fxcpsnsma, fxcpsnms, fxcpsnms; two results generated per instruction, four flops)
3	30	BGP_PU2_FPU_DUAL_PIPE_OTHER_NON_STORAGE_OPS_UM3	PU2: Dual pipe remaining non-storage instructions (fpmr, fpneg, fsmr, fsneg, fxmr, fsmfp, fsmtp, fpabs, fpnabs, fsabs, fsnabs, frsp, fpctiw, fpctiwz, fpre, frsqtr, fpsel, fscmp)
3	31	BGP_PU2_FPU_QUADWORD_LOADS_UM3	PU2: Quad Word Loads (ffpdx, lfpdux, lfxdx, lfxdux)
3	32	BGP_PU2_FPU_OTHER_LOADS_UM3	PU2: All other Loads (lfs, lfsx, lfsu, lfsux, lfpsx, fpsux, lfdsx, lfdsux, lfssx, lfssux, lfd, lfdx, lfdu, lfdux, lfxsx, lfxsux)
3	33	BGP_PU2_FPU_QUADWORD_STORES_UM3	PU2: Quad Word Stores (stfpdx, stfpdux, stfxdx, stfxdux)
3	34	BGP_PU2_FPU_OTHER_STORES_UM3	PU2: All other FPU Stores (stfs, stfsx, stfsu, stfsux, stfd, stfdx, stfdu, stfdx, stfiwx, stfpx, stfpsux, stfpiwx, stfsdx, stfsdux, stfssx, stfssux, stfxsx, stfxsux)
3	35	BGP_PU3_JPIPE_INSTRUCTIONS_UM3	PU3: Number of J-pipe instructions

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, fmmadd, fmmadds, fnmsub, fnmsubs; one result generated per instruction, two flops)
3	60	BGP_PU3_FPU_DIV_1_UM3	PU3: PowerPC FP Div (fdiv, fdivs; Single Pipe Divide)

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

6.3 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.

6.4 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.

6.5 Inclusive and exclusive values

For a motivating example of the term *exclusive values*, refer to Example 6-1 on page 78. 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.

6.5.1 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.

6.5.2 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.

6.5.3 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.

6.6 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 ≤ 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 ≤ 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 6.5, “Inclusive and exclusive values” on page 122.
 - `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 6.10, “Multithreaded program instrumentation issues” on page 127, 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.

6.7 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.

6.8 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 6.6, “Function reference” on page 123.
- ▶ 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 window 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.

6.9 Examples of libhpm for C and C++

Example 6-2 shows the syntax for C and C++, which are the same. The libhpm routines are declared as having external C linkage in C++.

Example 6-2 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 6-3) should call the functions with the prefix `f_`. Also, in Example 6-3, notice that the declaration is required on all source files that have instrumentation calls.

Example 6-3 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 )

```

6.10 Multithreaded program instrumentation issues

When placing instrumentation inside of parallel regions, use different ID numbers for each thread, as shown in Example 6-4 for Fortran.

Example 6-4 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
```

6.11 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 6.4, “Inheritance” on page 121). 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.

6.11.1 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.

6.11.2 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.

6.11.3 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.

6.11.4 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.

6.11.5 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 6-5 describes the types of parameters that are used in a call to a function aggregator.

Example 6-5 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 6.5, “Inclusive and exclusive values” on page 122, 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.

6.11.6 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_architecture>
```

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.

Archived



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.

7.1 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.

7.2 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 7-1 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.

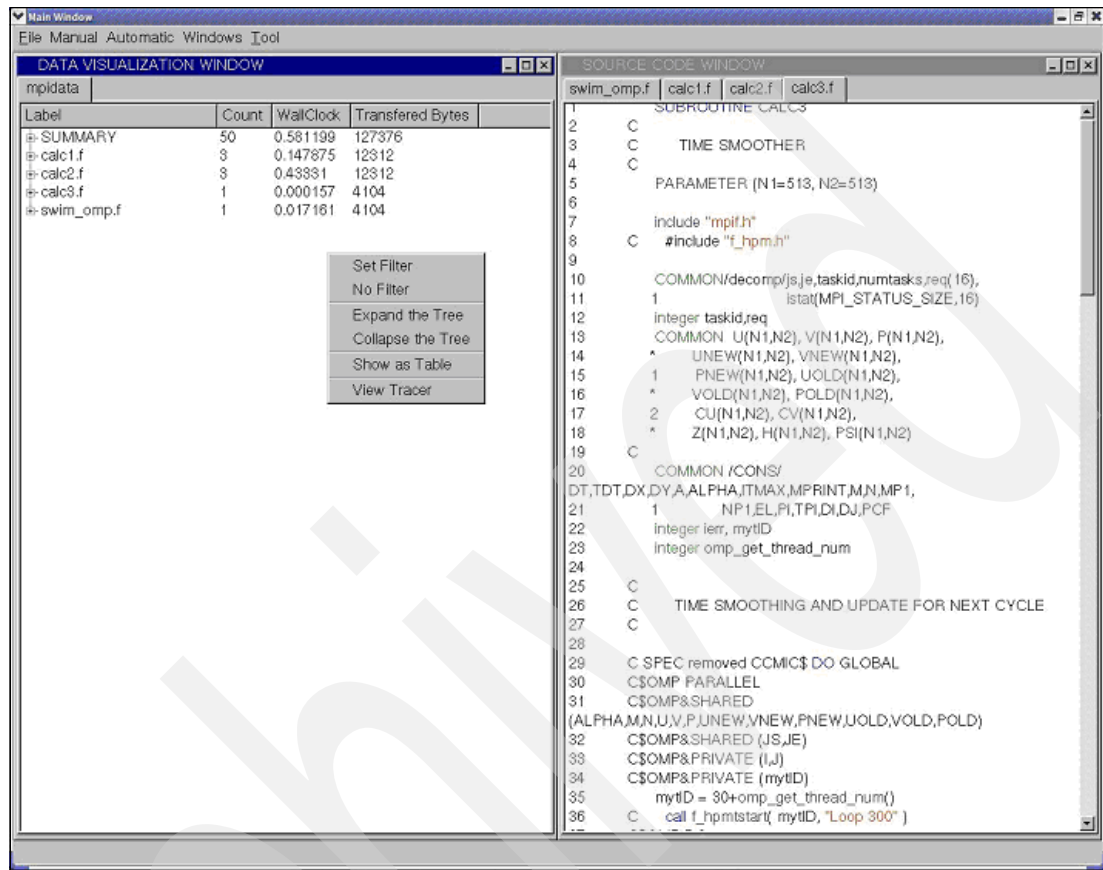


Figure 7-1 HPCT GUI

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 7-2) 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.

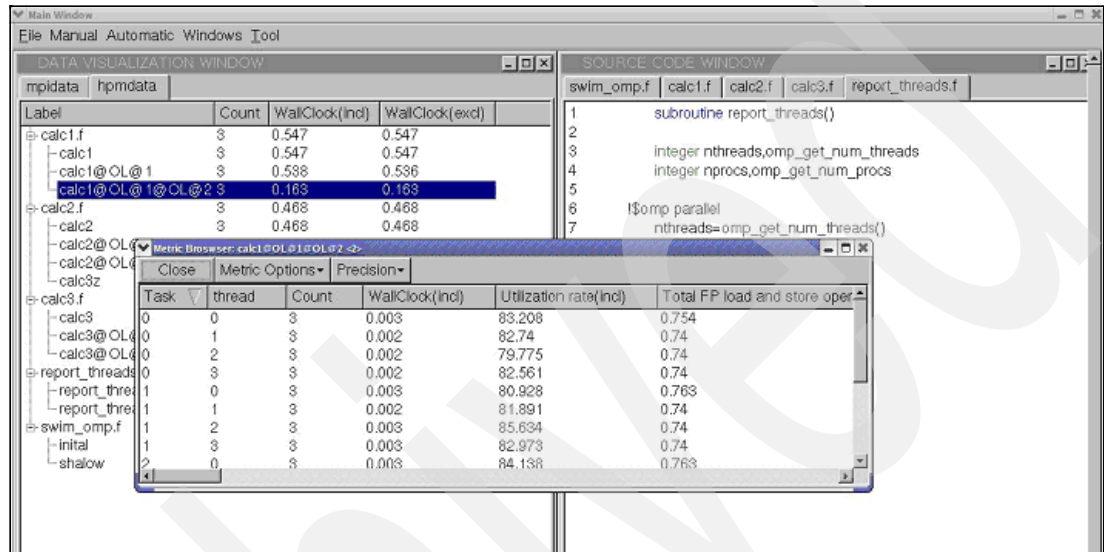


Figure 7-2 All performance data for specified node

If you right-click a non-leaf node or other empty space, a context menu opens as shown in Figure 7-3 (also shown in Figure 7-1 on page 135). 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.

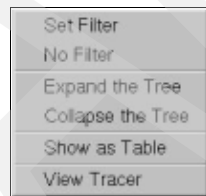


Figure 7-3 Context menu

View Tracer

The MPI Trace Viewer window (Figure 7-5) 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.

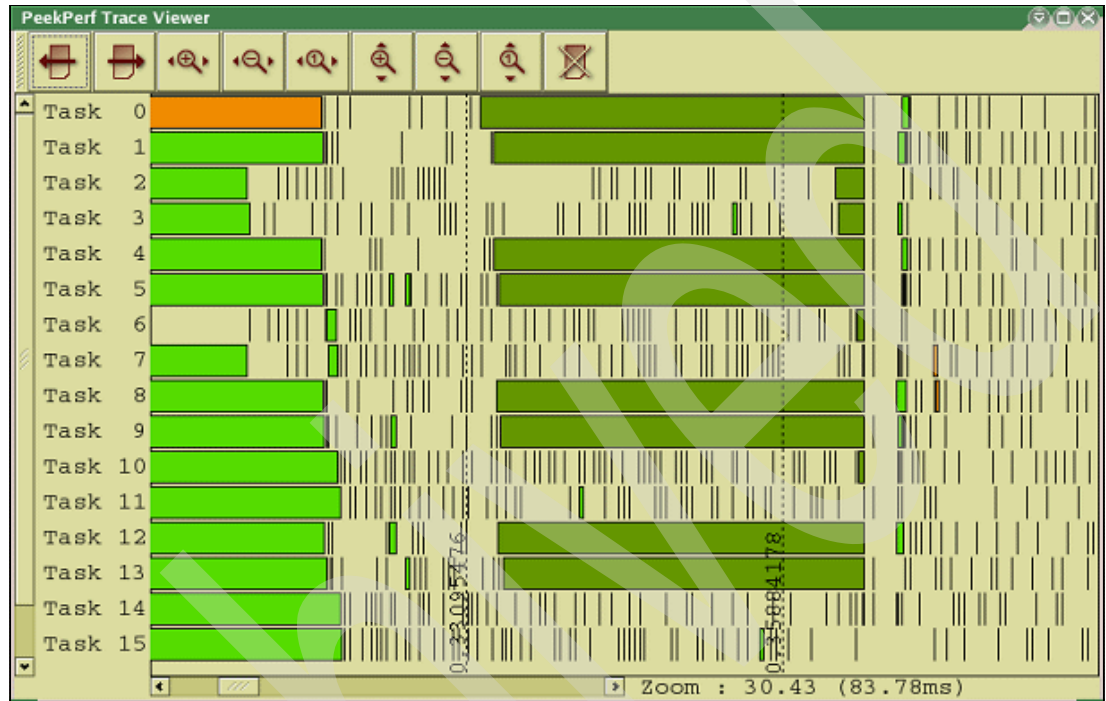


Figure 7-5 MPI Trace Viewer

Figure 7-6 shows the same trace with a different zoom level.

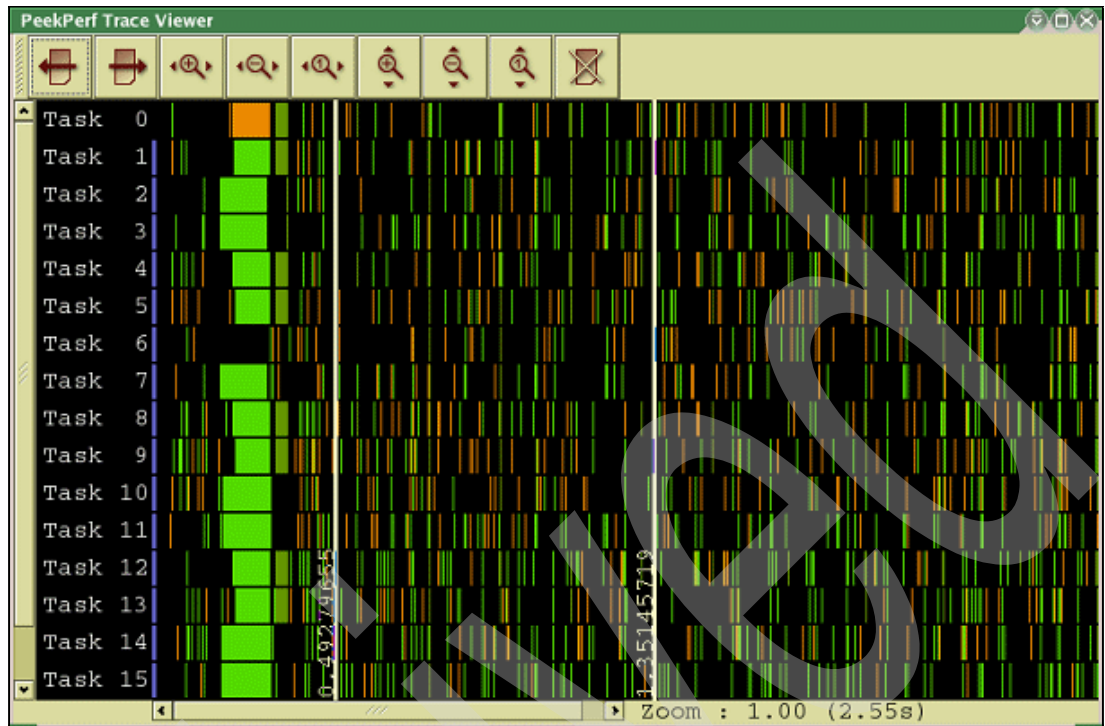


Figure 7-6 Trace Viewer with a new zoom level

Figure 7-7 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.

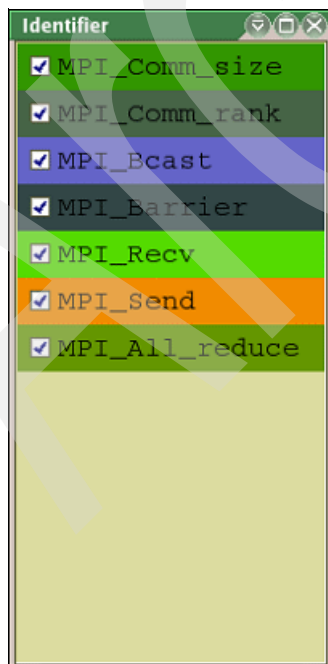


Figure 7-7 Trace Viewer Identifier window

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 time frame.

When you right-click an event in the Trace Viewer window, a box opens as shown in Figure 7-8. This window represents a summary of the collected data for the selected event.

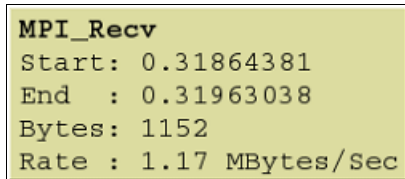


Figure 7-8 Summary of data collected

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 7-9). This toolbar can be undocked from the window. You can zoom in and out vertically and horizontally.



Figure 7-9 Navigation toolbar

You can also use your keyboard to navigate through the trace. Table 7-1 provides a list of keys and their corresponding action.

Table 7-1 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 7-10 opens. The I/O Trace Viewer is the window in the right side of the figure.

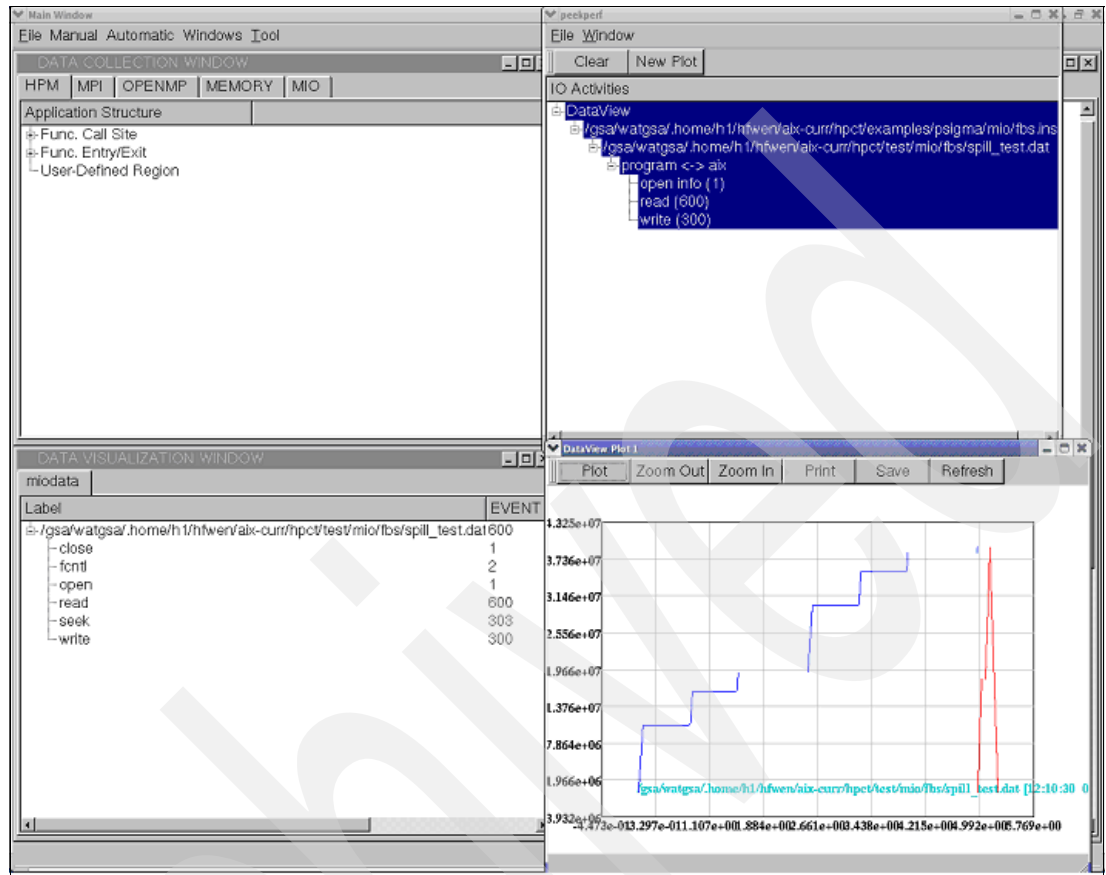


Figure 7-10 I/O Trace Viewer

7.3 HPCT GUI Main Window with instrumentation

When the binary is given to the HPCT GUI, the Data Collection Window (Figure 7-11) 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.

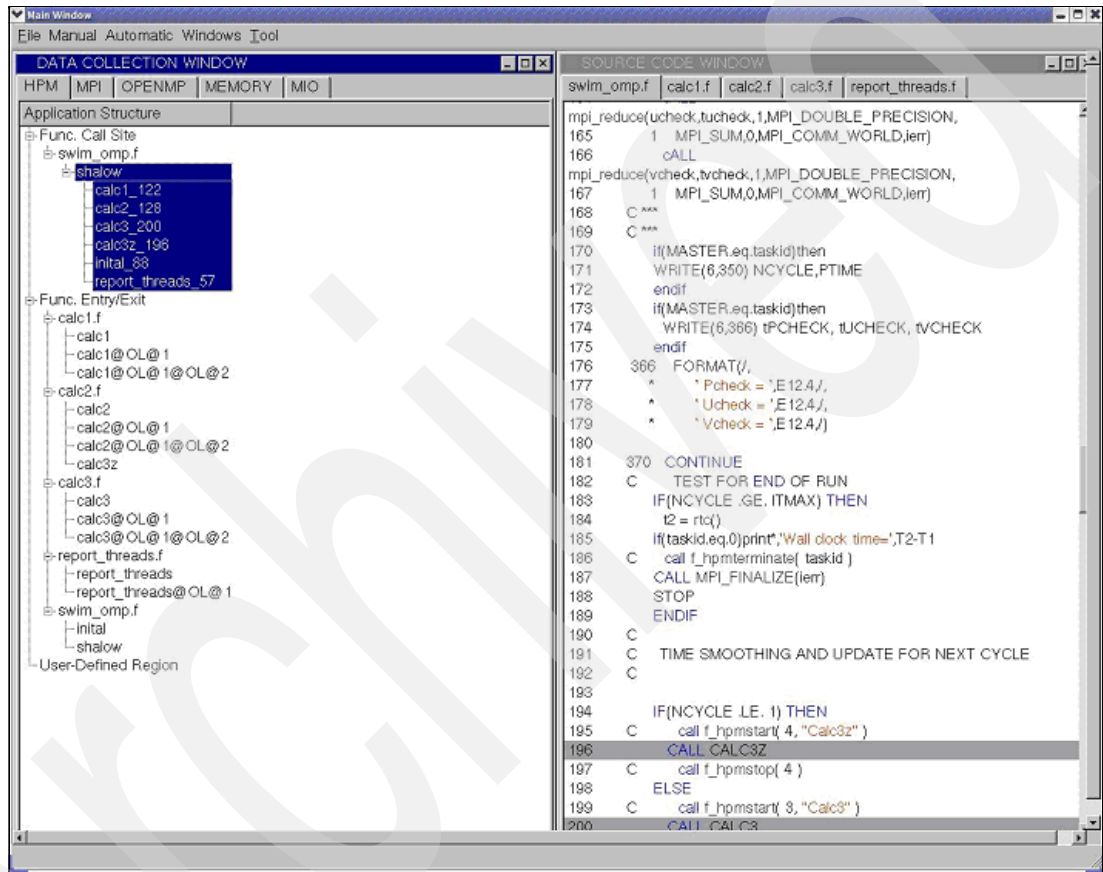


Figure 7-11 Main window With instrumentation

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 7-12 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	shallow	calc1_122

Figure 7-12 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 7-11 on page 142). 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 7-13 shows an example after the instrumented application has run. The HPM and MPI data are collected in a single run.

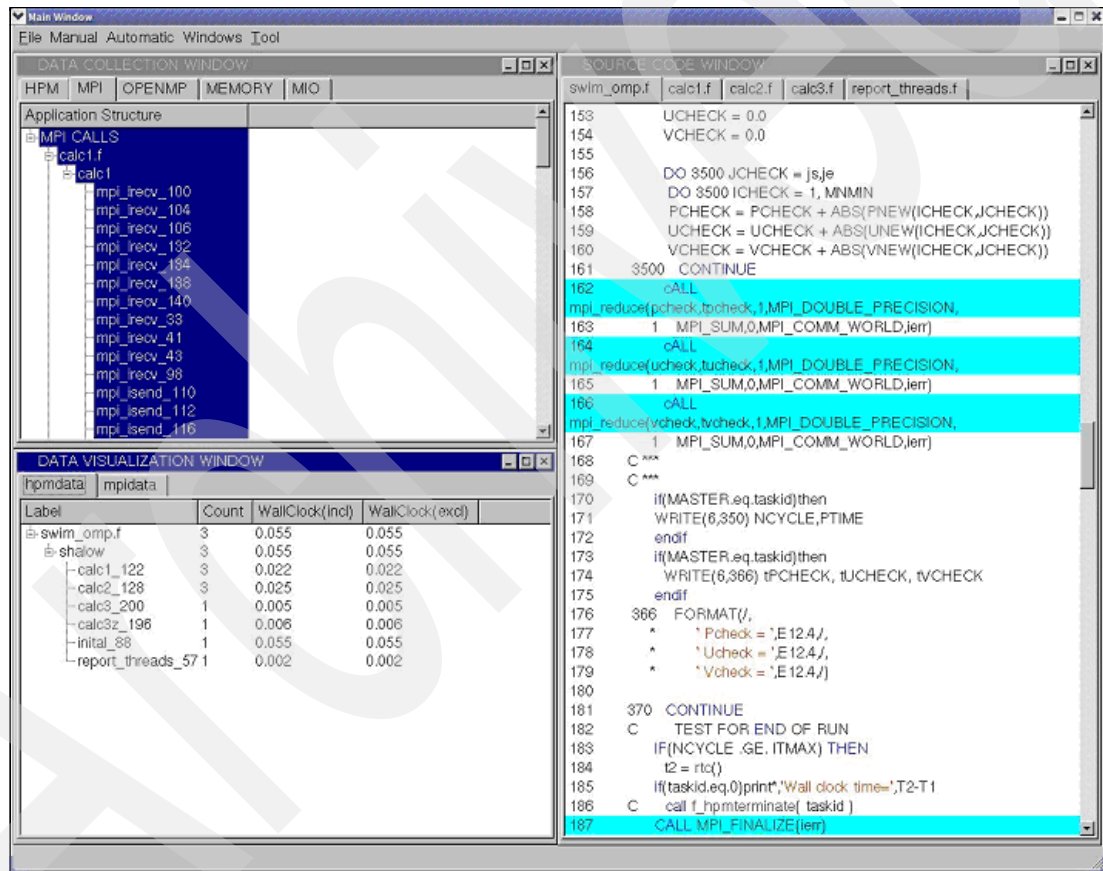


Figure 7-13 Instrumented application

For MPI, the tree is displayed in two different ways. One way is to group by MPI function classes (see Figure 7-14).

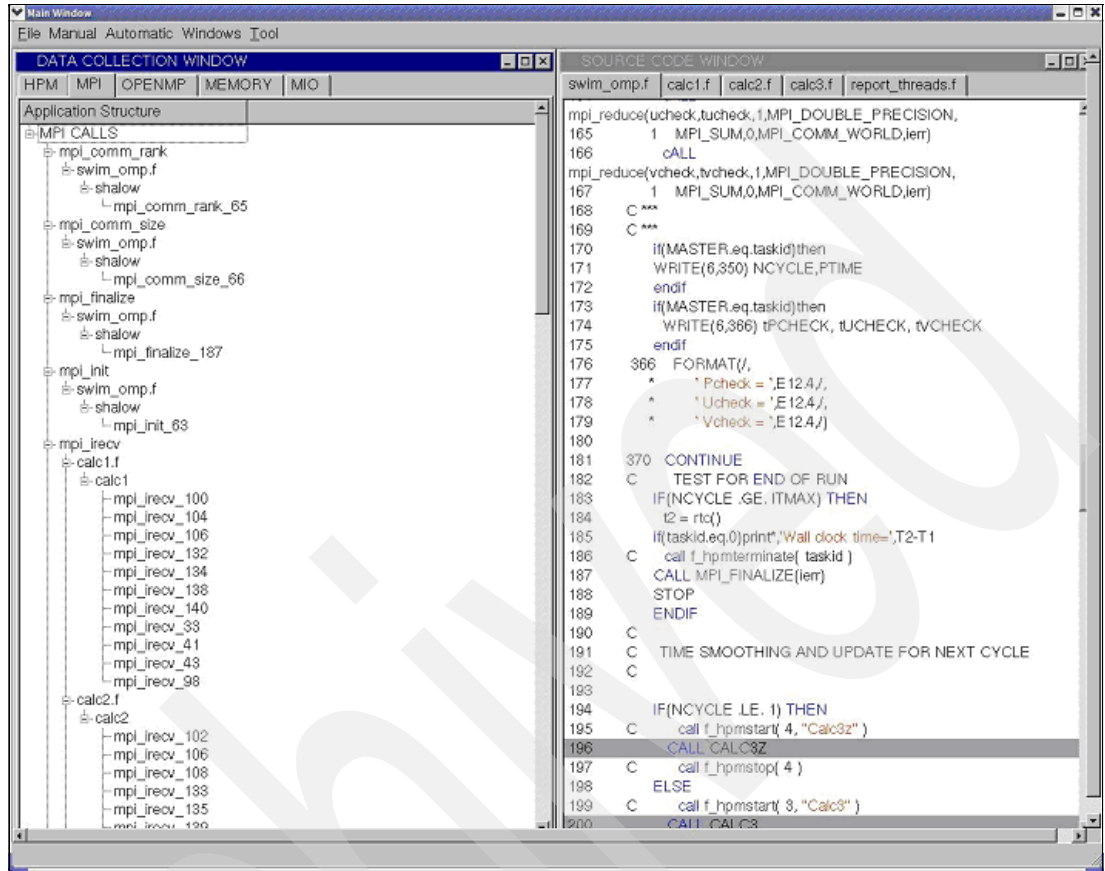


Figure 7-14 Grouped by MPI functions

The other way is to group by File and Function (see Figure 7-15). The display for MPI can be switched by the option in the context menu of the MPI panel.

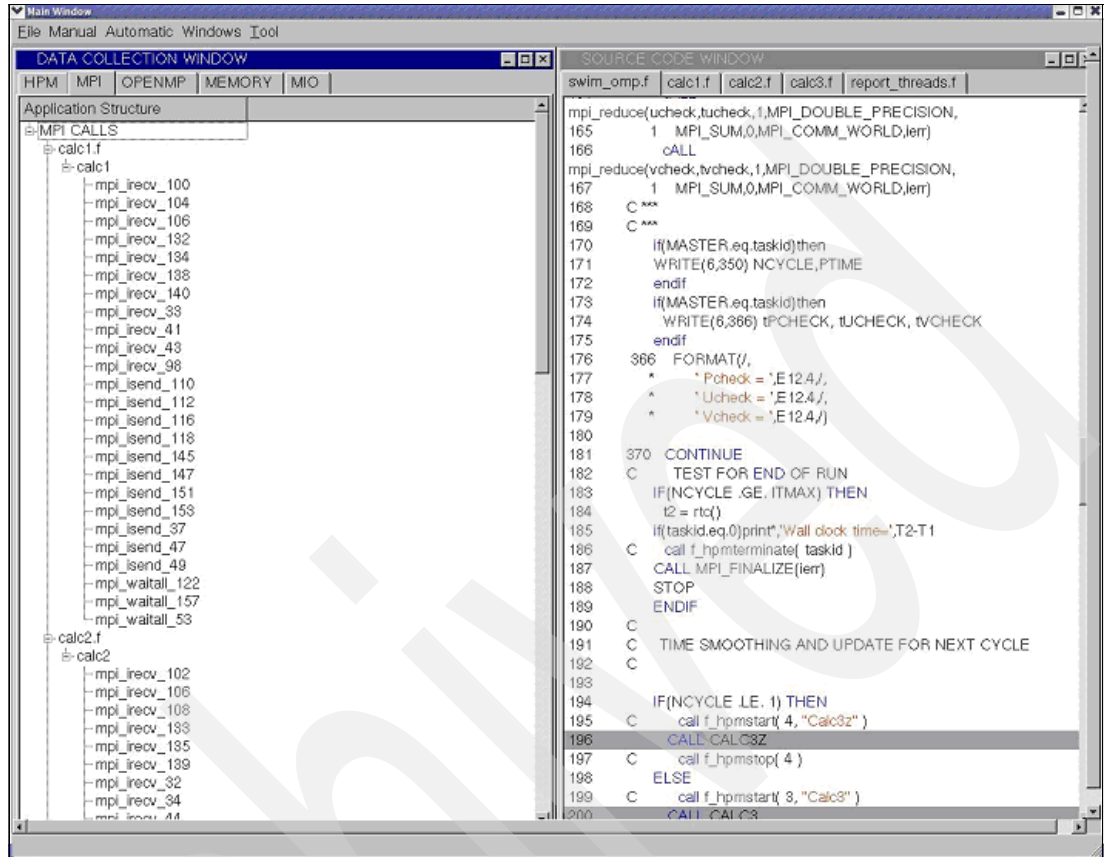


Figure 7-15 Grouped by file or function

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 7-16), 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.

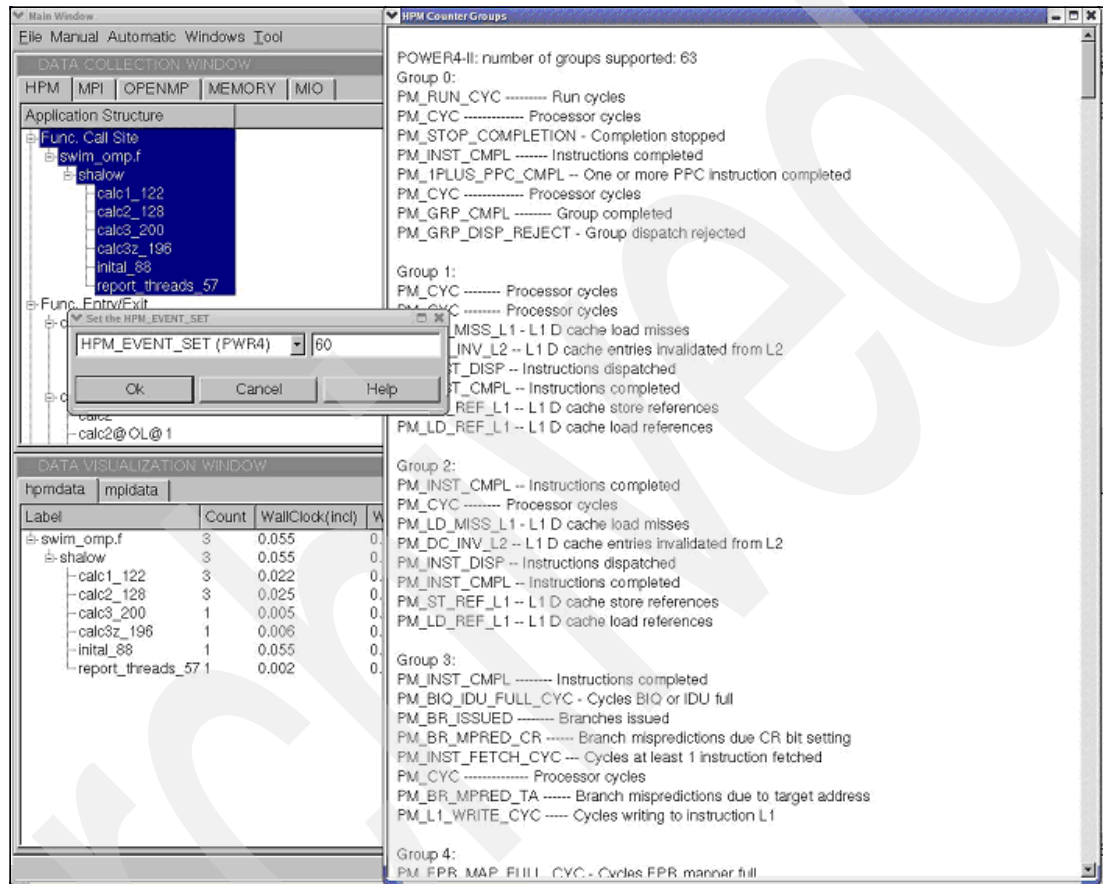


Figure 7-16 Setting the HPM_EVENT_SET

7.4 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**.

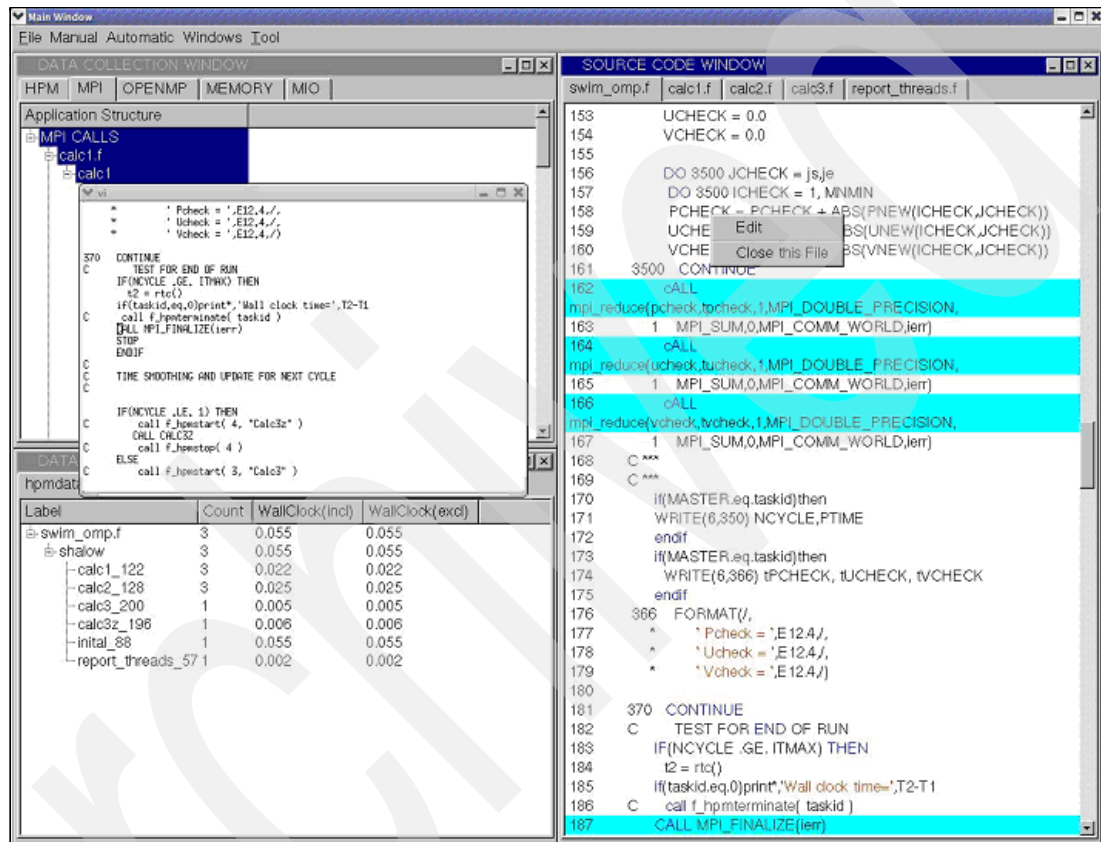


Figure 7-17 Editing the source code

Archived

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.

8.1 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:

mio	The interface to the user program
pf	A data prefetching module
trace	A statistics gathering module
aix	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.

8.2 Runtime control of MIO

MIO is controlled via four environment variables:

- ▶ MIO_STATS
- ▶ MIO_FILES
- ▶ MIO_DEFAULTS
- ▶ MIO_DEBUG

8.2.1 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.

8.2.2 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 default options for a default invocation. The user might override the default 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.

8.2.3 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.

8.2.4 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:

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

8.3 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:

mode=(0,0,0)	Override the default mode
nomode	Do not override the mode
direct	Set the O_DIRECT bit
nodirect	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:

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

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

mbytes	Output statistics in units of megabytes
gbytes	Output statistics in units of gigabytes
tbytes	Output statistics in units of terabytes
inter=(30,30,30)	Output intermediate trace statistics on kill -30
nointer	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:

debug	Print debug statements for open and close
nodebug	Do not print debug statements for open and close

8.4 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`.

8.5 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 8-1) 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 8-1 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 8-2 shows the `makefile.bgp` file.

Example 8-2 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 8-3, we ran the program using `mpirun` on partition `N11_32_1`.

Example 8-3 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 8-4 contains the sample MIO header file.

Example 8-4 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_nwait64(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 8-5 shows how to return the summary results as statistics.

Example 8-5 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 8-6 provides an example of returning the results in a figure.

Example 8-6 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 8-1 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.

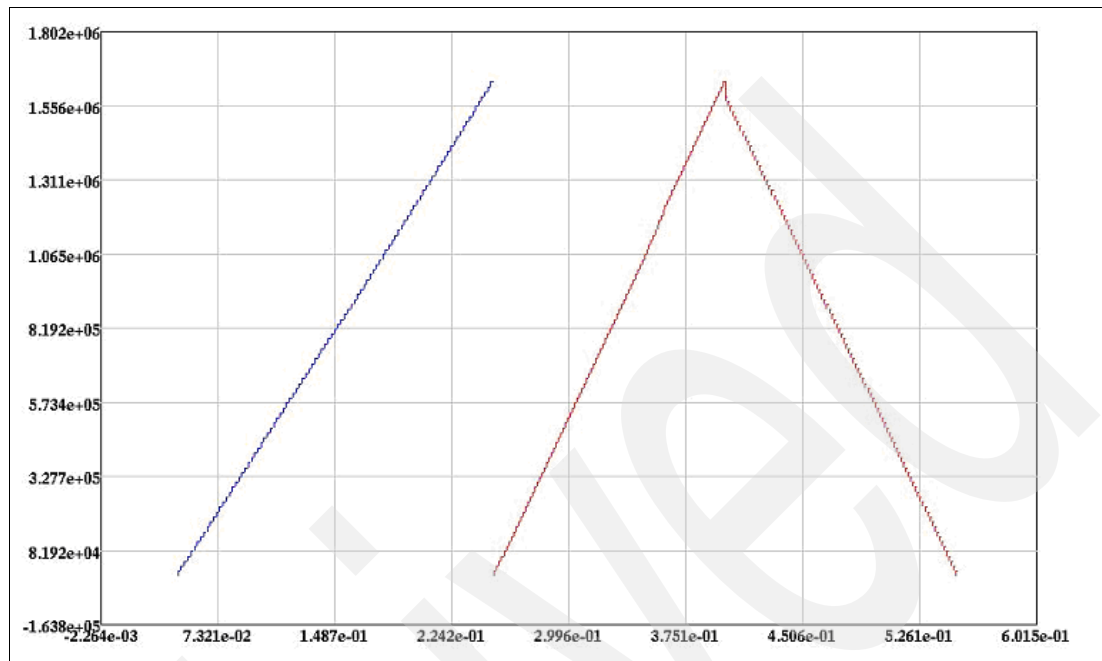


Figure 8-1 I/O access pattern

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 8-2 shows the POSIX I/O profiling for the IOR application.

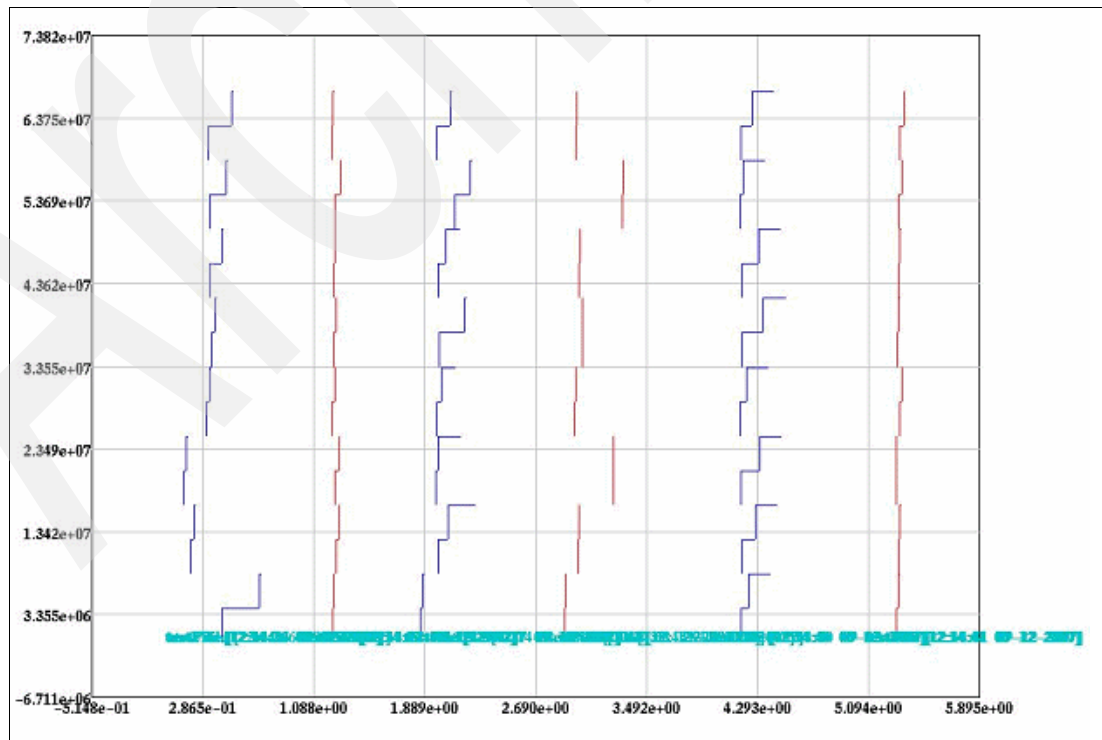


Figure 8-2 POSIX

Figure 8-3 is the Fortran I/O for the BTIO application.

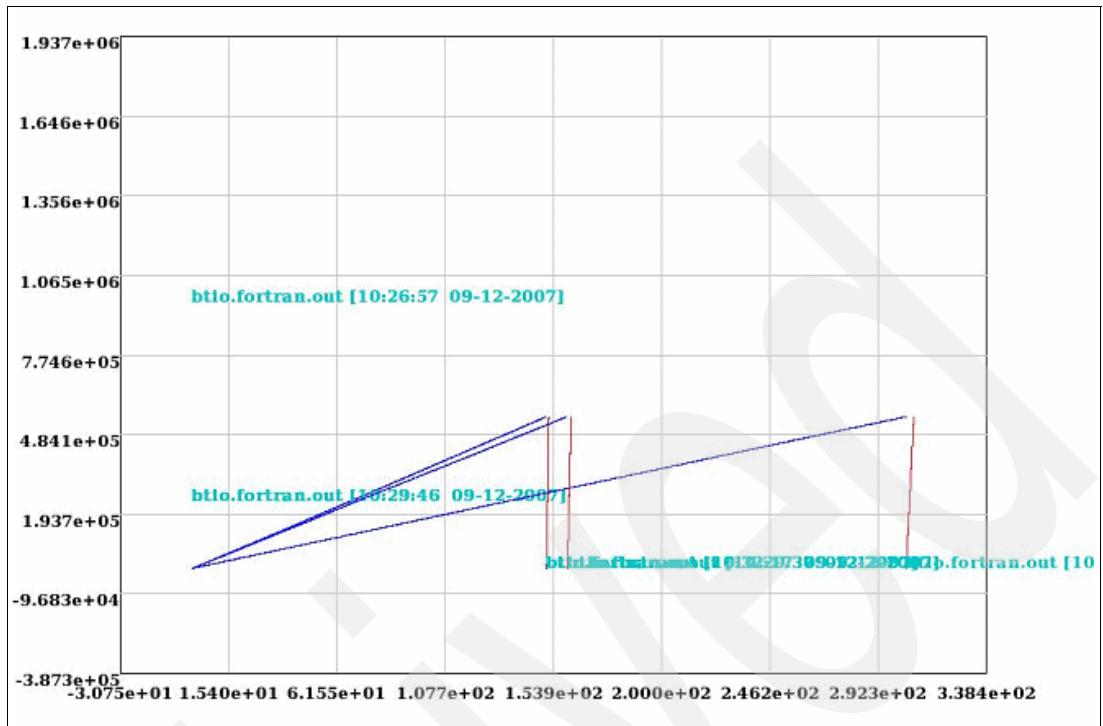


Figure 8-3 Fortran

Figure 8-4 shows EPIO mode for the BTIO application.

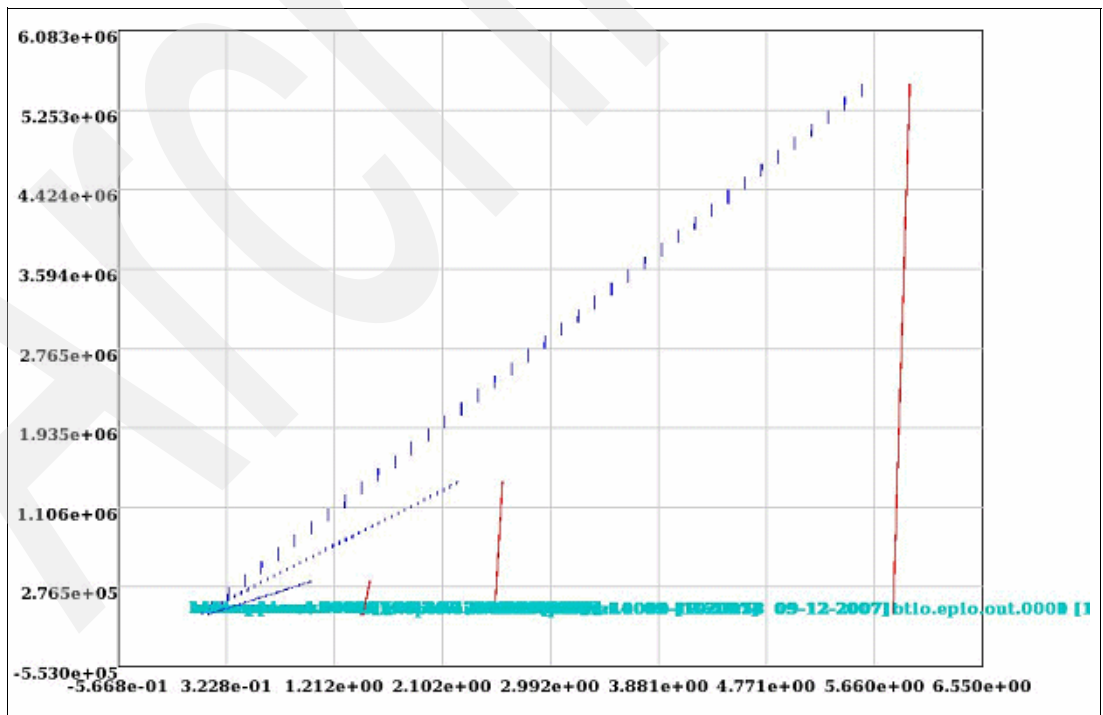


Figure 8-4 EPIO

Archived

GPFS and I/O node

This part provides details on performance tuning your GPFS configuration and the Blue Gene I/O nodes.

Archived



IBM General Parallel File System and I/O node performance tuning

This chapter provides information to assist you in tuning the I/O infrastructure supporting Blue Gene systems to provide optimal performance. Topics in the chapter include

- ▶ Recommended General Parallel File System (GPFS) configuration settings for the Network Shared Disk (NSD) servers and I/O nodes.
- ▶ Instructions on how to run various performance tests to evaluate and tune the complete IO system.
- ▶ Real world results to provide baseline reference points for designing and verifying a Blue Gene/P GPFS solution.

9.1 Cluster architecture

GPFS uses multi-cluster (remote mount) to provide the file system to the Blue Gene cluster. Typically, there are two clusters: the I/O cluster and the Blue Gene cluster. The I/O cluster hosts the file systems and contains the NSD servers and storage arrays. The Blue Gene cluster is made up of the service node and the Blue Gene/P I/O nodes. The Blue Gene cluster mounts the file system from the I/O cluster and uses the GPFS Network Shared Disk (NSD) mechanism to access the data.

The GPFS I/O cluster is designed in scalable units called *building blocks*. A building block is a set of servers and storage arrays that provide a certain I/O performance level. An I/O solution for Blue Gene/P includes one or more of these building blocks based on storage performance and capacity requirements. Some tested example building blocks include:

Linux Servers with IBM DCS Storage

▶ Example 1

- Four IBM System x3655 servers: Four to eight core, 8 GB memory, InfiniBand® (IB) Storage connection, 10 GbE
- DDN 9550 with 48 tiers
- Sequential performance approximately 2.1 GBps read/write

▶ Example 2

- Eight IBM System x3450 servers: Four core, 8 GB memory, IB Storage connection, 10 GbE
- 1xDDN 9900 with 48 tiers
- Sequential performance approximately: 5 GBps read/write

9.2 System configuration and settings

Tuning the I/O subsystem requires tuning all data paths, including the I/O node configuration, TCP/IP network, GPFS NSD servers, the NSD server connection to the storage, and the storage itself. This section describes the setup of your GPFS environment with Blue Gene. Topics covered are:

- ▶ TCP settings
- ▶ GPFS NSD server configuration
- ▶ Blue Gene/P I/O node setup
- ▶ GPFS Service and Login node settings
- ▶ Storage array settings

9.2.1 TCP settings

The TCP/IP network should be tuned for optimal sequential throughput. The following network parameters worked well in various 10 GbE network configurations. To apply these tuning suggestions to the GPFS NSD servers, add these parameters to `/etc/sysctl.conf` and restart the network:

```
net.ipv4.tcp_wmem = 10000000 10000000 10000000
net.ipv4.tcp_mem = 10000000 10000000 10000000
net.core.rmem_max = 10000000
net.core.wmem_max = 10000000
```

To restart the network on the GPFS NSD servers, use this command:

```
service network restart
```

To to apply these changes to the I/O nodes, add the following lines to the file `/bgsys/iofs/etc/init.d/rc3.d/S10sitefs` and then reboot the Blue Gene/P block:

```
echo 10000000 10000000 10000000 > /proc/sys/net/ipv4/tcp_wmem
echo 10000000 10000000 10000000 > /proc/sys/net/ipv4/tcp_mem
echo 10000000 > /proc/sys/net/core/rmem_max
echo 10000000 > /proc/sys/net/core/wmem_max
```

9.2.2 GPFS NSD server configuration and setup

Note: The parameters used in this setup were tested on storage building blocks made up of four IBM System x3655 servers, each with one 10 GbE Myricom Network card and one InfiniBand connection to a DCS storage array.

The following steps provide configuration tuning information for the GPFS NSD servers in a building block for I/O support of a Blue Gene/P system. These steps and settings should apply to most levels of supported releases of Linux, but may vary slightly from release to release. When setting up a GPFS NSD server, the procedure is:

1. If you use an InfiniBand link between the NSD servers and the storage servers, be sure that your Open Fabrics Enterprise Distribution (OFED) driver level V1.3.1 or later is installed. The driver level on the NSD servers can be determined by running the following command:

```
rpm -qa | grep -i ofed
```

2. Next, configure the InfiniBand driver using the steps outlined below.

- a. The first step is to configure the driver to start. Often there is only a single connection to the storage from the NSD server. In this case, there is no need to use the multi-path driver. If you decide not to use the multi-path driver, you should stop the `openibd` driver from creating the multi-path (`/dev/dm-*`) devices. This is done by editing the `/etc/init.d/openibd` startup script. To stop the script from creating the devices, change the following line in `/etc/init.d/openibd` from:

```
ACTION=="add", KERNEL=="sd* [!0-9]", RUN+="/sbin/multipath %M:%m"
```

to:

```
ACTION=="add", KERNEL=="sd* [!0-9]"
```

This step removes the command that loads the multi-path driver on device creation.

- b. To start OpenSM on boot, ensure that the `onboot` parameter is set in the configuration file. In the file `/etc/opensm.conf`, add the parameter:

```
ONBOOT=yes
```

- c. Verify that the drivers are set to start using the `chkconfig` command:

- **#chkconfig openibd**

If the drivers are set to start, the command will return `openibd on`.

- **chkconfig opensmd**

If the drivers are set to start, the command will return `opensmd on`.

- In the next step, you create scripts to set the SCSI, InfiniBand, and network device parameters. The SCSI device parameters should be modified from the defaults to obtain optimal sequential throughput. For example, the default maximum I/O size on Linux is 512 KB, so if these parameters are not changed, the largest I/O that GPFS can send and receive from the storage is 512 KB. If the GPFS file system block size is greater than 512 KB, which is common in Blue Gene installations, these parameters need to be increased to take advantage of the performance gain when using larger I/O sizes. Table 9-1 contains a list of SCSI device parameters that need to be changed from the defaults to take advantage of larger GPFS file system block size settings.

Table 9-1 Device parameter settings

Parameter	Default value	Required value	Description
max_hw_sectors_kb	512	32767	Maximum IO size, in sectors, that can be issued to a device
max_hw_segments	255	256	Size of the scatter gather table available to be used to coalesce I/O requests
max_phys_segments	128	128	Maximum scatter gather list length
max_sectors_kb	512	8192	Largest possible I/Os going to disk
nr_requests	128	4	Number of outstanding requests to disk subsystem
read_ahead_kb	512	2048	Maximum size of read ahead operations
scheduler	cfq	Noop	Linux I/O Elevator

The device parameters are set by a script in boot /etc/init.d/boot.local. The script sets the configuration parameters for the DCS storage devices. These should be seen labeled /dev/sd*. These parameters can be set and verified in /sys/block/sdXX/queue/*. The parameters can be set on boot by creating or modifying the file /etc/init.d/boot.local to add the following:

```
#!/bin/bash
for i in `cat /proc/partitions | grep sd | tail +2 | egrep -v sda[0-9] | awk '{print $4}'`
do
    echo 2048 > /sys/block/${i}/queue/read_ahead_kb
    echo 4 > /sys/block/${i}/queue/nr_requests
    echo noop > /sys/block/${i}/queue/scheduler
    echo 8192 > /sys/block/${i}/queue/max_sectors_kb
done
```

- Reboot the system.
- Make sure all logical unit numbers (LUNs) are available by issuing the command:


```
#cat /proc/partitions
```
- Verify the device parameters were set properly by checking the contents of the following files for the available SCSI device:
 - cat /sys/block/[sd device]/queue/max_sectors_kb
 - cat /sys/block/[sd device]/queue/nr_requests
 - cat /sys/block/[sd device]/queue/read_ahead_kb
 - cat /sys/block/[sd device]/queue/nr_requests

7. Start the GPFS daemon using the following command:

```
# mmstartup
```

After setting up each node, reboot the server and confirm that the boot process is clean. This means that all appropriate drivers are starting and all tuning parameters are set. Until the boot process is consistently clean, GPFS should not be set to start automatically. Ensure all the LUNs are available before starting GPFS on the NSD server. Once the boot is clean (all devices come up and parameters are set), GPFS can be started automatically.

Debugging notes

If OpenSM does not start and `/var/log/messages` does not indicate what the problem is, you can collect debugging information by adding the following lines to the `/etc/init.d/opensmd` file:

```
#!/bin/bash -x
exec 1> /tmp/opensm.out
exec 2>&1
```

One error you may encounter is database corruption. The error message, found in the `/var/log/messages` file, is similar to the following:

```
Dec  5 15:19:50 fs1 OpenSM[12374]: OpenSM Rev:openib-3.0.14
Dec  5 15:19:50 fs1 OpenSM[12374]: FATAL: Error restoring Guid-to-Lid persistent
database
```

This message is caused by the database being corrupted or deleted. To fix this error, find the following line in `/etc/init.d/opensmd`:

```
local START_FLAGS=""
```

Add a `-y` between the quotes, which will cause the database to be rebuilt:

```
local START_FLAGS="-y"
```

9.3 Verifying startup

Once you have installed and configured the OpenSM driver, verify the environment is set up properly:

- ▶ If you rebooted your machine after the installation process completed, InfiniBand (IB) interfaces should be up. If you did not reboot your machine, enter the following command to start the driver:

```
# service openibd start
```

- ▶ Verify that the IB driver is running on all nodes. You can check the status of each machine by using the `ibv_devinfo` command. It should output `hca_id: <linux device name>` on the first line, as shown in Example 9-1.

Example 9-1 `ibv_devinfo` output

```
Example:
#ibv_devinfo
hca_id: mthca0
      fw_ver:                1.2.0
      node_guid:              0008:f104:0399:a7d0
      sys_image_guid:         0008:f104:0399:a7d3
      vendor_id:              0x08f1
      vendor_part_id:         25204
```

```

hw_ver:                0xA0
board_id:              VLT0050010001
phys_port_cnt:        1
port: 1
state:                 PORT_ACTIVE (4)
ax_mtu:                2048 (4)
active_mtu:           2048 (4)
sm_lid:                2
port_lid:              2
port_lmc:              0x00

```

- ▶ Make sure that a Subnet Manager (SM) is running by invoking the SMinfo utility. If a Subnet Manager is not running, SMinfo outputs the following line:

```
sminfo: iberror: query failed
```

- If an SM is running, SMinfo outputs the LID and other SM node information:

```
sminfo: sm lid 0x1 sm guide 0x2c9010b7c2ae1, activity
count 20 priority 1
```

- To check if OpenSM is running on the management node, enter:

```
service opensmd status
```

- To start OpenSM, enter:

```
service opensmd start
```

Tip: OpenSM parameters are set in the `/etc/opensm.conf` file. OpenSM can be configured to run upon boot by setting `ONBOOT=yes` in the `opensm.conf` file.

- ▶ Verify the status of the ports by using the `ibv_devinfo` command. All connected ports should report a `PORT_ACTIVE` state.
- ▶ Check the network connectivity status by running the `ibchecknet` command to see if the subnet is *clean* and ready for ULP/application use. The following tools display additional information about the InfiniBand ports:
 - `ibnetdiscover`
 - `ibhosts`
 - `ibswitches`
- ▶ Rather than performing the previous tasks individually, you can use the `ibdiagnet` utility to perform a set of tests on your network. Upon finding an error, `ibdiagnet` prints a message starting with an `-E-`. For a more complete report of the network features, you should run `ibdiagnet -r`. If you have a topology file describing your network, you can supply this file to the `ibdiagnet` command as a parameter using the `-t <file>` option. All the resulting reports will use the names that appear in the file instead of the default format (LIDs, GUIDs, and directed routes).
- ▶ To run an application over SDP, set the following variables:

```

env LD_PRELOAD='stack_prefix'/lib/libsdp.so
LIBSDP_CONFIG_FILE='stack_prefix'/etc/libsdp.conf <application name>
(or LD_PRELOAD='stack_prefix'/lib64/libsdp.so on 64 bit machines)

```

The default 'stack_prefix' is /usr.

9.4 Performance tuning the InfiniBand interface

Tuning the InfiniBand link between the NSD servers and disk storage array is critical for high performance of the storage subsystem. This section demonstrates how to set and verify the `srp_sg_tablesize`, `max_sect`, and `max_cmd_per_lun` parameters.

srp_sg_tablesize

The scatter gather table size should be at the maximum size to provide the best chance of coalescing I/O requests. The `srp_sg_tablesize` setting is loaded as part of the `ib_srp` module. To set this parameter, create a file named `/etc/modprobe.d/srp` that contains the following options:

```
options ib_srp srp_sg_tablesize=256
```

It can also be set manually using **modprobe**:

```
modprobe ib_srp srp_sg_tablesize=256
```

To verify the configuration, look at the `srp` file to verify that it is set upon boot:

```
cat /etc/modprobe.d/srp
```

To verify the current value, run:

```
cat /sys/module/ib_srp/parameters/srp_sg_tablesize
```

max_sect and max_cmd_per_lun

The `max_sect` parameter must be set to 65535 because there are two sectors per 1 KB and we recommend that you set `max_hw_sectors_kb` to 32767 in the SCSI Device Parameters section. To ensure that the IB device queue is not overrun, we recommend that the `max_cmd_per_lun` be limited to 3.

- ▶ To set these parameters, add the following line to the end of the `/etc/srp_daemon.conf` file:

```
a max_sect=65535,max_cmd_per_lun=3
```

- ▶ To verify the `max_sect` settings, use the following command:

```
cat /sys/block/sdc/queue/max_hw_sectors_kb
```

The returned result should be a number one half the value you set for `max_sect` because the results are in KB and there are two sectors per 1 KB.

- ▶ To verify the `max_cmd_per_lun`, issue the following commands (in order):

```
cat /proc/scsi/sg/device_hdr
cat /proc/scsi/sg/devices
```

The returned results should look like those in Example 9-2. The column `qdepth` shows the current value for `max_cmd_per_lun`.

Example 9-2 Verify max_cmd_per_lun

```
jalien@fs1:/proc/scsi/sg> cat device_hdr
host      chan    id      lun     type    opens   qdepth  busy    online
jalien@fs1:/proc/scsi/sg> cat devices
0         0       0       0       0       1       3       0       1
0         0       1       0       0       1       3       0       1
5         0       0       0       0       1       3       0       1
5         0       0       1       0       1       3       0       1
5         0       0       2       0       1       3       0       1
5         0       0       3       0       1       3       0       1
```

9.5 GPFS Blue Gene/P I/O node settings

The configuration of GPFS on the I/O nodes in a Blue Gene configuration is different than the NSD servers. These parameters have been tested to work specifically with the unique Blue Gene architecture.

Pagepool

Set the pagepool to 384 MB. The I/O nodes are diskless so memory use is at a premium. The maximum GPFS pagepool setting (for GPFS Version 3.2) on an I/O node (maximum for Blue Gene/P) is 384 MB. You can set the pagepool with this command:

```
mmchconfig pagepool=384m -N ionode.lst
```

ionode.lst is a file containing a list of all I/O nodes (either IP address or host name) with one entry per line. The service node may have a larger pagepool setting.

TCP send and receive buffer sizes

The default GPFS TCP buffer size is dynamic and can get large. On I/O nodes, we recommend that the TCP send and receive settings are set to 256 KB to control memory usage and packet size:

```
mmchconfig socketRcvBufferSize=262144 -N ionode.lst
mmchconfig socketSndBufferSize=262144 -N ionode.lst
```

You can verify these parameters are set by using the `mm1sconfig` command with no options.

GPFS I/O node parameters

These values have been tested on Blue Gene/P and have shown good results on multiple systems with 16 I/O nodes per Blue Gene/P rack. These parameters can be changed by using the `mmchconfig` command on both the NSD server and the I/O nodes.

maxMBpS=150

This may seem counterintuitive, but when configuring for sequential I/O, `maxMBpS` is increased. In this case the regular rules do not apply because the I/O node usage mode is not common. Though the I/O node has only a few cores, it processes the I/O operations for up to 256 compute node cores. So, from the GPFS perspective, it is a very large SMP server with very little memory (because GPFS can only use the memory on the I/O node). Keeping `maxMBpS` at the default of 150 ensures each process uses less memory, thus reducing contention.

prefetchThreads=8 This value is similar to the one above; reducing prefetchThreads from the default of 72 reduces contention and improves sequential IO performance.

worker1Threads=24 The worker1Threads parameter controls the maximum number of concurrent file operations at any one time.

Pinned GPFS processes to cores 1 and 2

One possible tuning option when the Blue Gene/P system has a high ratio of compute nodes to I/O nodes would be to pin the GPFS processes to cores 1 and 2 of the I/O nodes. This option has been measured to help in some workloads, but there has not been enough testing done to recommend it for all applications. The following instructions describe how to pin the GPFS process to cores 1 and 2 of the I/O node.

1. Create a S40gpfs script on the service node by copying the file `/bgsys/drivers/ppcfloor/ramdisk/etc/init.d/gpfs` to `/bgsys/iofs/etc/init.d/rc3.d/S40gpfs`.
2. Next, edit the newly created S40gpfs file. Find the line:
`/usr/lpp/mmfs/bin/mmautoLoad`
3. Replace it with the following line:
`taskset 0x6 /usr/lpp/mmfs/bin/mmautoLoad`

Pinning CIOD processes to specific I/O node cores

Another possible tuning option is to pin the I/O node process to specific cores. This option may help in some applications, but there has not been enough testing done to recommend it for all workloads. To place the main threads on core 0, ioproxy threads on cores 1 and 2, and the reader on core 3, create a file named `/bgsys/iofs/etc/sysconfig/ciod` and add the following lines:

```
export CIOD_MAIN_AFFINITY_MASK=1
export CIOD_IOPROXY_AFFINITY_MASK=6
export CIOD_READER_AFFINITY_MASK=8
```

Increased CIOD buffer size

The CIOD buffer size should be increased to match the GPFS file system block size. To set the CIOD buffer size on the service node, create the file `/bgsys/iofs/etc/sysconfig/ciod` and add the line:

```
export CIOD_RDWR_BUFFER_SIZE=4194304
```

To verify the CIOD buffer size is set correctly, use telnet to connect to the I/O nodes (see Example 9-3).

Example 9-3 Telnet to an I/O node

```
#telnet [ioNodeIP] 9000
ciod running with 256 threads and 0 active processes
> show_config_info
CIOD_RDWR_BUFFER_SIZE: 4194304
CIOD_REPLY_MSG_SIZE: 4194304
CIOD_TREE_RECV_RETRY: 50
CIOD_NEW_JOB_EXIT_PGM: (null)
CIOD_NEW_JOB_EXIT_PGM_TIMEOUT: 3600
CIOD_TRACE_DIRECTORY: (null)
CIOD_LOG_LEVEL: 0
CIOD_END_JOB_TIMEOUT: 4
```

The CIOD_RDWR_BUFFER_SIZE directly affects the total I/O node memory used. Care must be taken to ensure that the total memory used does not exceed the total memory currently available on the I/O node. The total memory used is calculated by multiplying the CIOD RDWR buffer size times four processes per compute node times the number of compute nodes per I/O node. For example, a buffer size of 256 KB with 32 compute nodes per I/O node requires 32 MB of storage (256 KB x 4 processes per node x 32 compute nodes). A buffer size of 4 MB with 64 compute nodes per I/O node will require 1 GB of storage (4 MB x 4 x 64).

9.6 GPFS Service and login node settings

In general, the Blue Gene/P service node and login nodes are configured differently than the I/O nodes. The service node and login nodes should be tuned for human interaction to the file system. This means that parameters, including `maxFilesToCache` and `Pagepool`, should be increased to enhance the performance of user commands, such as `ls`, and to support a different usage model. Many of the parameters can be the same as the rest of the cluster.

service node

The service node is the cluster configuration server and quorum node for the Blue Gene/P GPFS cluster,

Pagepool

The pagepool can be set larger than on the I/O nodes. A setting of 2 GB or more is appropriate, depending on the available memory. In addition, larger values of pagepool can be helpful on login nodes.

maxFilesToCache

Increasing the `maxFilesToCache` parameter on the login nodes is highly recommended. The default of 1000 is often too low on a system where users are interacting with the file system. The `maxFilesToCache` parameter should be increased to 5,000 or 10,000, depending on the level of activity. One indication that this value should be increased is when interactive commands like `ls` are taking a very long time to complete.

Note: Only increase `maxFilesToCache` on login nodes. Increasing this value on all of the I/O nodes can cause an unnecessarily large number of tokens that need to be managed. This can affect the performance of the entire cluster.

The `-N` parameter of the `mmchconfig` command is used to set the `maxFilesToCache` parameter only on the service and login nodes:

```
mmchconfig maxFilesToCache=1000 -N <Service_Node1>,<Login_Node1>,...
```

9.7 Storage array settings

When implementing a solution using GPFS with DCS 9550 or DCS 9900 storage, there are a few items to take into consideration. Some of these recommendations impact performance, while others are for reliability.

- ▶ We recommend using separate Fibre Channel or SAS based metadata storage.
- ▶ Disable read prefetch.
- ▶ Create one GPFS LUN per tier.

Use separate metadata storage

There are multiple reasons to use separate metadata storage in a configuration using the DCS 9550 or the DCS 9900. The first is reliability and the second is performance.

Neither the DCS 9550 or DCS 9900 have a mirrored or battery backed write cache. If there is, for example, a power failure during a metadata write, your file system could be corrupted. In some situations, this risk may be acceptable for data storage. When used for data storage, the effect may be a corrupt file. For metadata storage, if a write fails, the entire file system could be affected. For this reason, we recommend either disabling write cache on metadata LUNS, which could effect performance, or use another storage server, for example, a DS4800, to store metadata.

The DCS 9550 and 9900 are highly optimized for sequential large block data access. Metadata operations are very small (between 4 KB and 8 KB) random I/O requests. In addition, the DCS storage is often configured with SATA disk, which is not optimal for small random I/O patterns. Placing the metadata on alternate storage servers, for example, DS4800 with Fibre drives, can greatly improve overall solution performance. Because the percentage of metadata storage required is very small, the additional cost of adding the DS4800 is often minimal.

Disable read prefetch on the controller

GPFS prefetch patterns cannot be easily detected by storage controller prefetch logic, so you will achieve better performance by disabling read prefetch on the DCS controllers.

To disable read prefetch, use the `cache` command from DDN admin prompt:

```
> cache prefetch=0
```

Use dedicated tiers for GPFS

This is common for all types of storage when used with GPFS. For optimal sequential I/O performance, it is best to allow GPFS to manage the whole LUN to reduce disk thrashing.

9.8 Performance verification

Validating the performance of a GPFS storage infrastructure should be done in steps. It is often a complex solution and breaking it down into measurable sub-components can greatly ease implementation. There are three areas of performance on which to focus while verifying and tuning the solution:

- ▶ Storage performance
- ▶ Network performance
- ▶ System performance

9.8.1 Storage performance

The goal is to tune and verify the performance of disk I/O within each building block. This means running the performance test directly on the GPFS NSD server nodes and measuring the aggregate performance across all nodes to all the storage in a single building block. When running these tests, look for balance across all NSD servers as well as overall storage throughput.

dd/sgp_dd

A quick way to get a good idea of sequential storage performance is to use the **dd** or **sgp_dd** commands. The **sgp_dd** command can be used to verify read and write disk performance.

Important: Take care when running **sgp_dd** write tests. Ensure you are not writing to a LUN used by another application. This could cause critical file system data to be overwritten and lost.

Start by measuring the performance of one NSD server to one DCS using as many LUNS as possible. Start a **sgp_dd** process for each LUN and run the test in parallel. When a single node is performing as expected, apply and verify any changes required on the other nodes and then move to testing the entire building block. Alternatively, you can create a GPFS file system on each building block for testing or use the GPFS **mmchdisk** command to suspend disks to isolate storage servers.

Be sure to verify the I/O performance on each NSD server and storage array. To verify the entire building block, run **sgp_dd** on a group of servers (for example, eight) to one storage array.

An example **sgp_dd** read command is:

```
sgp_dd of=/dev/null if=/gpfs/ddn22/jalien/ddtest/file133 bs=4M count=10000 time=1
```

An example **sgp_dd** write command is:

```
sgp_dd if=/dev/zero of=/gpfs/ddn22/jalien/ddtest/file134 bs=4M count=10000 time=1
```

See 9.11.1, “dd scripts” on page 184 for more example dd scripts.

Storage performance strategy

The strategy of the storage performance tests is to verify that you have acceptable performance from the GPFS NSD servers to the storage array. To do this, start by verifying the performance of one NSD server to one storage server. If that test yields acceptable results, then scale up to several (for example, eight) NSD servers all reading/writing the same storage array. Keep scaling up until you have verified all servers and disks in the building block are performing to expectations. If the building block does perform to expectations, go back and do a read and write test on each server by itself to identify the issue.

If an individual server shows poor performance, verify the device and hardware configuration for that particular server. If all servers in a building block show evenly degraded performance, then verify the storage and operating system configurations.

Storage performance results

We tested dd write and read performance with a 4 MB GPFS file system block size. The results for dd write are shown in Table 9-2 on page 177.

Table 9-2 dd write results

NSD server	DDN	Results (in MBps)
8 IBM System x3450 servers: 4 core, 8 GB memory, IB storage connection, 10 GbE	1 DDN S2A9900	5300 to 5600
4 IBM System x3655 servers: InfiniBand storage connection, 10 GbE	1 DDN S2A9550	2300 to 2400

Table 9-3 contains the results from our dd read tests.

Table 9-3 dd read results

NSD server	DDN	Results (in MBps)
8 IBM System x3450 servers: 4 core, 8 GB memory, IB storage connection, 10 GbE	1 DDN S2A9900	5100 to 5600
4 IBM system x3655 servers: InfiniBand storage connection, 10 GbE	1 DDN S2A9550	2100 to 2200

9.8.2 Network performance tools

Network performance is essential to success when implementing a high performance Blue Gene/P I/O solution. GPFS uses an all clients to all servers network connection model, which means that all NSD clients (I/O nodes) have sockets open concurrently to all NSD servers. This type of workload can be a challenge for any network infrastructure, so this should be tested thoroughly.

nsdperf

The tool *NSDPerf* is provided by the GPFS development group and can be used for testing a *many to many* node network performance. NSDPerf simulates the network I/O pattern for a set of NSD clients and servers. NSDPerf can be obtained by e-mailing a request to the GPFS development team at gdfs@us.ibm.com.

Building NSDPERF

Here are the steps required to build NSDPerf. These steps should be followed on a login server (front end node) for the I/O nodes and on the NSD server for the file servers.

1. Expand the tar file.
2. Change into the NSDPerf directory and enter the following command to build NSDPerf:

```
g++ -o nsdperf -O2 -lpthread nsdperf.cc
```

Additional instructions and examples can be found in the README file.

Running NSDPerf

NSDPerf verifies the network performance between a group of clients and servers. To accomplish this task, the NSDPerf daemon must be started on all machines, then each machine must be designated as client or server, and finally, the test is run.

1. Start all the machines to be tested in server mode. This can be done by creating a *testnodes* file that contains a list of all the systems participating in the test. Either the IP address or system name can be used. Use the following command to start the NSDPerf processes on all machines:

```
nsdperf -s
```

An example of running the **nsdperf** command from a script is shown in Example 9-7 on page 187.

2. Verify all the NSDPerf processes started by issuing the command:

```
server status
```

See Example 9-7 on page 187.

3. The actual testing is done from a *control node*, which can be any machine that can connect to all of the servers. From the *control node*, run the command **nsdperf** with no parameters to get to an NSDPerf prompt.

Example 9-4 provides an example of the command used to start a NSDPerf run.

Example 9-4 Starting an nsdperf run

```
% nsdperf -v
nsdperf> server fin25 fin26 fin27
Connected to fin25
Connected to fin26
Connected to fin27

nsdperf> client fin28 fin29
Connected to fin28
Connected to fin29

nsdperf> connect

nsdperf> status
test time: 5 sec
data buffer size: 4194304
socket send/receive buffer size: 10000000
clients:
  fin28 9.1.72.237
  fin25 9.1.72.234
  fin26 9.1.72.235
  fin27 9.1.72.236
  fin29 9.1.72.238
  fin25 9.1.72.234
  fin26 9.1.72.235
  fin27 9.1.72.236
servers:
  fin25 9.1.72.234
  no connections
  fin26 9.1.72.235
  no connections
  fin27 9.1.72.236
  no connections

nsdperf> test
write rate 119 MB/sec
read rate 158 MB/sec

nsdperf> killall
Connection from 9.1.72.234 broken
Connection from 9.1.72.235 broken
Connection from 9.1.72.236 broken
```

```
Connection from 9.1.72.237 broken
Connection from 9.1.72.238 broken
```

```
nsdperf> quit
```

Alternatively, you can place the commands in a file and call NSDPerf with that command file:

```
nsdperf -I commandfile
```

From that prompt, you can type **help** to get the list of available commands. There are commands used to designate servers and clients, tell clients to connect, and run the tests. For more info, see the README file supplied in the NSDPerf download.

iperf

Another useful tool to that can measure network performance is *iperf*. Iperf is an open source tool that can be downloaded from:

<http://sourceforge.net/projects/iperf/>

Iperf is good for measuring node to node or parallel stream peak network performance. However, it is difficult to run iperf in an all client to all server configuration on large systems.

Building iperf for the NSD server

Use these steps to build iperf on one of the NSD servers:

1. Place the downloaded iperf tar file on one of the NSD servers.
2. Run `./configure` and make in the iperf-X.Y.Z directory, but use the supplied makefile (as is) from the tar file.
3. Copy the executable into the desired bin directory on the NSD servers to be tested.

Building iperf for the I/O node

Either the service node or front end node (login node) can be used to build the executables to be run on the I/O nodes:

1. On the Blue Gene/P service node, expand the downloaded tar file and **cd** to the main directory (that is, **cd iperf-X.Y.Z**).
2. Run the configuration script to configure your machine.
`./configure`
3. Run the following command to update the Makefile for BGP I/O nodes:
`patch -b Makefile -i iperf_make.diff`
The iperf_make.diff file is included in the Blue Gene/P build directory `/bgsys/drivers/ppcfloor/tools/perf_diags/iperf`.
4. Run **make** to build the executable. The iperf executable (called *iperf*) will be generated in the iperf-X.Y.Z/src directory.
5. Copy the iperf executable into the iperf home directory created in “Running iperf”.

Running iperf

Scripts for running iperf can be found in the Blue Gene/P build in the directory `/bgsys/drivers/ppcfloor/tools/perf_diags/iperf`. Here are the steps to run iperf:

1. Create an iperf home directory and an output directory, for example, `/gpfs1/jalien/iperf` and `/gpfs1/jalien/iperf/output`.

2. Export the home directory:


```
export IPERF_HOME=/gpfs1/yourHomeDir/iperf
```
3. Create an iperf configuration file that contains your system's I/O nodes and NSD server names or IP addresses. See the iperf.config file in /bgsys/drivers/ppcfloor/tools/perf_diags/iperf for an example.

Note: You will need to make and install iperf on the NSD servers if this has not been already done. See “Building iperf for the NSD server” on page 179.

4. Start the iperf server jobs on the GPFS NSD servers under test by running the command:


```
iperf -s
```
5. Allocate and boot a Blue Gene block under test.
6. Use the runIperf script to run the different iperf tests. Here is the format of the command:


```
runIperf <blockname> <testname> <configfile>
```

The command parameters are as follows:

blockname	Blue Gene/P partition name under test
testnames	
1to1	Runs a 1 I/O node to 1 NSD server iperf
1to16	Runs a 1 I/O node to 16 NSD servers iperf
16to1	Runs a 16 I/O nodes to 1 NSD server iperf
16to16	Runs a 16 I/O nodes to 16 NSD servers iperf
alltoall	Runs a all I/O nodes to 16 NSD servers iperf
configfile	Configuration file for Blue Gene/P partition

Network performance strategy

The object of this test is to verify the network is performing as expected. Start by running one iperf or NSDPerf client connected to one server, then two clients to two servers, and build up to 16 to 16. If the network is performing properly, the performance should scale up evenly to 16 times the one to one performance.

For one I/O node to one NSD server, performance should be around 800 MBps. If the one to one results are not within 10% of that number, stop here and verify that the network settings for I/O node, switch, and NSD server are set correctly.

If the one to one results are as expected, but the results do not scale up with 16 or more iperf client/servers, then there is probably a switch or cabling issue. To isolate the source of the slowdown, try running each of the client/server pairs individually. If you obtain poor results with any of the pairs, then more investigation will be required in the network settings and hardware for the affected client and server. If running each pair individually shows good results, then re-run the 16 to 16 test again and check the individual results to see if all connections slow down evenly or if just a certain few are slower than the rest. If all connections slow down evenly, for example, verify that the I/O nodes under test are all evenly distributed across different line cards in the switch. If only a few connections are performing poorly, verify that they are not cabled through a switch port that may be getting blocked by another I/O node's connection and that there are no errors on the port.

The *alltoall* test can be used to check the performance on all I/O nodes to all NSD servers. For Blue Gene/P systems, the target I/O throughput for an alltoall test is based on the target

performance of the network infrastructure of the system. For example, in an alltoall test on one network vendor, the maximum performance target is 58% of the total available bandwidth (10 Gbit * number of nodes * 58% = expected throughput).

Network performance results

We tested network performance using 16 I/O nodes as clients and one to eight GPFS NSD servers as servers with a Myricom switch. The results for iperf and NSDPerf are located in the following tables. Table 9-4 contains the iperf results.

Table 9-4 iperf results

Iperf client command	Number of clients	Number of servers	Results (in MBps)
<code>iperf -c -w 800k -P 16 -t 30</code>	16	1	1229
<code>iperf -c -w 800k -P 16 -t 30</code>	16	2	2292
<code>iperf -c -w 800k -P 16 -t 30</code>	16	4	4413
<code>iperf -c -w 800k -P 16 -t 30</code>	16	6	5320
<code>iperf -c -w 800k -P 16 -t 30</code>	16	8	5993

Table 9-5 shows the NSDPerf network performance results.

Table 9-5 NSDPerf results

NSDPerf time	NSDPerf threads	Number of clients	Number of servers	Write results (in MBps)	Read results (in MBps)
40	4	512	126	74100	74000
40	4	512	135	76200	76300
120	4	512	136	76800	79600
300	4	512	136	76700	79300

9.9 Total system performance

The IOR tool can be used to measure the total system (compute node to disk) file system I/O performance. IOR is open source and can be downloaded from the following Web site:

<http://sourceforge.net/projects/ior-sio/>

Building IOR

Use the following steps to build IOR:

1. After expanding the downloaded tar file, change directories into the main directory:

```
cd /IOR-X.YY.Z/src/C
```

Where X.YY.Z is the version of IOR that you are building.

2. Patch the /src/C/Makefile.config so it includes a Blue Gene/P target. Use the command:

```
patch -e -b Makefile.config -i ior_make.diff
ior_make.diff
```

3. `cd` back to the main directory:

```
cd ../...
```

4. From that prompt, run:

```
gmake posix
```

5. The executable generated is called IOR and it is put in the src/C directory.

Running IOR

Use the following steps to run IOR:

1. Modify the testFile parameter in the supplied arguments files to target the GPFS file system under performance evaluation. See the example IN.file.XXX files in the /bgsys/drivers/ppcfloor/tools/perf_diags/ior directory.
2. Copy the IOR executable and all modified argument files to an established current working directory (cwd) on a file system accessible by the Blue Gene/P system under test.
3. On the service node, execute this command to run IOR:

```
mpirun -partition R00-M0-N02 -cwd /bgusr/jalien -exe ./IOR.rts -args -f  
IN.file.1G -nofree -verbose 1 -label 2>&1 | tee -a IOR.out
```

where the parameters -partition, -cwd, -exe, and -args all must be updated to values specific to the particular system and setup:

-partition	The allocated BGP block under test
-cwd	The chosen current working directory
-exe	The IOR executable built earlier
-args	Used to specify the IN.file with all the test settings

More information about input parameters can be found in the README and USER_GUIDE files from the IOR tar file expansion.

System performance strategy

The strategy for running IOR is to allocate a fairly large Blue Gene/P block (one rack or more if possible) so you can get a good idea of the throughput from a large number of compute nodes to the file system and underlying storage. Start with a smaller IOR blockSize parameter like 64 or 16 MB to do a short IOR run. This will verify everything is set up and working correctly. It will also complete faster and provide a quick result to let you know if performance is in the expected range. If this tests yields acceptable results, then increase the IOR blockSize parameter to a larger value and rerun the test. If the short test does not yield acceptable results, then more investigation needs to be done with the network and storage performance.

Two different variations of this test can be run by setting different values for the file per process (filePerProc) parameter in the IOR configuration file. Using filePerProc=0 will cause all compute nodes to write and read one common file. With filePerProc=1, each compute node creates a unique file to write and read. In the file per process case, there will be a wait for file creation time, which is reported by IOR. This creation time is included in the final throughput number. If file creation time is not part of the measurement, or if you wish to run without it for testing, you can reuse the directories and files created by setting the IOR parameter useExistingTestFile = 1 and keepFile = 1 in the configuration file.

System performance results

The results in Table 9-6 on page 183 were obtained using a 4 MB GPFS file system block size to match the IOR transferSize parameter. The Blue Gene/P I/O nodes are connected to 16 IBM p520 NSD servers with two DDN DCS9550 couplets through a 10 GbE Force 10 Switch. The NSD servers are connected to the DDN couplets through 4 Gb Fibre Channel.

Table 9-6 IOR one file per process (compute node), POSIX API

Number of compute/I/O nodes	IOR transferSize parameter	IOR blockSize parameter	Aggregate file size	Write results (in MBps)	Read results (in MBps)
2048/64	4 MiB	1 GiB	2048 GiB	4122	4976
2048/64	4 MiB	1 GiB	2048 GiB	4185	4976
2048/64	4 MiB	1 GiB	2048 GiB	4194	4971
2048/64	4 MiB	1 GiB	2048 GiB	4192	4965

The results in Table 9-7 were obtained using a 4 MB GPFS file system block size to match the IOR transferSize parameter. The Blue Gene/P I/O nodes are connected to 136 IBM x3455 NSD servers with 17 DDN S2A9900 couplets through a Myricom 2 Tier CLOS network. The NSD servers are connected to the DDN couplets through an InfiniBand 4X connection.

Table 9-7 IOR one file per process (compute node), POSIX API

Number of compute/I/O nodes	IOR transferSize parameter	IOR blockSize parameter	Aggregate file size	Write results (in MBps)	Read results (in MBps)
32768/512	4 MiB	2 GiB	64 TiB	71762	75974
8192/128	4 MiB	2 GiB	16 TiB	37236	47404
4096/128	4 MiB	2 GiB	8 TiB	35622	Not Done

The results in Table 9-8 were obtained using a 4 MB GPFS file system block size to match the IOR transferSize parameter. The Blue Gene/P I/O nodes are connected to 128 IBM System x3455 NSD servers with 16 DDN S2A9900 couplets through a Myricom 2 Tier CLOS network. The NSD servers are connected to the DDN couplets through an InfiniBand 4X connection. GPFS level 3.2.1-5 is required for single shared file performance fixes.

Table 9-8 IOR single shared file results, POSIX API

Number of compute/I/O nodes	IOR transferSize parameter	IOR blockSize parameter	Aggregate file size	Write results (in MBps)	Read results (in MBps)
32768/512	4 MiB	2 GiB	64 TiB	67105	N/A
32768/512	4 MiB	2 GiB	64 TiB	62779	53551
32768/512	4 MiB	2 GiB	64 TiB	61831	57835

9.10 Troubleshooting I/O size

The I/O size is the size of the request that is being issued by GPFS to the disk. If the I/O size is smaller than expected, performance can be impacted. You can determine the I/O size that is being issued to the disk by using a system monitoring tool such as iostat or nmon. Using the `iostat` command with the `-x` parameter shows the I/O request size sent to the disk. The `nmon` command shows similar information using the `-D` option. If the average I/O size is much smaller than expected, verify the device parameters and alternately try rebooting the NSD server.

Two known issues that can cause small I/O sizes:

- ▶ SCSI device parameters

Verify that `max_hw_sectors_kb` and `man_sectors_kb` are set properly. If not, rerun `/etc/init.d/boot.local`.

- ▶ Memory fragmentation

The theory is that multiple restarts of the GPFS daemon on a Linux server can cause fragmentation of memory pages used for I/O processing, effectively reducing the maximum I/O size that can be issued to the storage. Rebooting the NSD servers cleans up the memory.

9.11 Example scripts

The following section contains sample scripts for topics previously discussed in this chapter.

9.11.1 dd scripts

This section contains sample `ddwrite` and `ddread` scripts. These example scripts are for a system with 136 file servers named `fs1-data` to `fs136-data` and 17 `ddn` building blocks named `ddn6` to `ddn22`. Given a `ddn` building block as input, the scripts will do either **dd write** or **dd read** commands from the eight file servers to that `ddn` unit.

Example 9-5 contains a sample `ddwrite` script.

Example 9-5 ddwrite

```
#!/bin/bash

if [ "${1}" == "ddn6" ]; then
    nodes="seq 1 8"
    fs=sddn6
elif [ "${1}" == "ddn7" ]; then
    nodes="seq 9 16"
    fs=sddn7
elif [ "${1}" == "ddn8" ]; then
    nodes="seq 17 24"
    fs=sddn8
elif [ "${1}" == "ddn9" ]; then
    nodes="seq 25 32"
    fs=sddn9
elif [ "${1}" == "ddn10" ]; then
    nodes="seq 33 40"
    fs=sddn10
elif [ "${1}" == "ddn11" ]; then
    nodes="seq 41 48"
    fs=sddn11
elif [ "${1}" == "ddn12" ]; then
    nodes="seq 49 56"
    fs=sddn12
elif [ "${1}" == "ddn13" ]; then
    nodes="seq 57 64"
    fs=sddn13
elif [ "${1}" == "ddn14" ]; then
```



```

    nodes="seq 65 72"
    fs=sddn14
elif [ "${1}" == "ddn15" ]; then
    nodes="seq 73 80"
    fs=sddn15
elif [ "${1}" == "ddn16" ]; then
    nodes="seq 81 88"
    fs=sddn16
elif [ "${1}" == "ddn17" ]; then
    nodes="seq 89 96"
    fs=sddn17
elif [ "${1}" == "ddn18" ]; then
    nodes="seq 97 104"
    fs=sddn18
elif [ "${1}" == "ddn19" ]; then
    nodes="seq 105 112"
    fs=sddn19
elif [ "${1}" == "ddn20" ]; then
    nodes="seq 113 120"
    fs=sddn20
elif [ "${1}" == "ddn21" ]; then
    nodes="seq 121 128"
    fs=sddn21
elif [ "${1}" == "ddn22" ]; then
    nodes="seq 129 136"
    fs=sddn22
else
    echo "Unknown building block"
    exit
fi

if [[ $2 == "" ]]; then
    echo "parm 2 must be dd count value"
    exit
fi

# Set up a directory to collect the results
dt=`date +%h%d%y-%H_%M_%S`
results=/tmp/${dt}.${1}.wr
mkdir -p ${results}

for i in `-${nodes}`
do
    ping -c1 fs${i}-data >/dev/null 2>&1
    (( $? != 0 )) && continue
    ssh fs${i}-data mount | grep /gpfs/${fs} >/dev/null 2>&1
    (( $? != 0 )) && echo Aborting: /gpfs/${fs} not mounted on fs${i}-data && exit 1
done

((first=1))

for i in `-${nodes}`
do
    ((first)) && ssh fs${i}-data mkdir -p /gpfs/${fs}/ddtest && ((first=0))

```

```
ssh fs${i}-data time dd if=/dev/zero of=/gpfs/${fs}/ddtest/file${i} bs=4M
count=$2 > $results/fs${i}-data.out 2>&1 &
done
wait
echo results are in ${results}
grep copied ${results}/*.out
```

Example 9-6 contains a sample ddread script.

Example 9-6 ddread

```
#!/bin/bash

if [ "${1}" == "ddn6" ]; then
    nodes="seq 1 8"
    fs=sddn6
elif [ "${1}" == "ddn7" ]; then
    nodes="seq 9 16"
    fs=sddn7
elif [ "${1}" == "ddn8" ]; then
    nodes="seq 17 24"
    fs=sddn8
elif [ "${1}" == "ddn9" ]; then
    nodes="seq 25 32"
    fs=sddn9
elif [ "${1}" == "ddn10" ]; then
    nodes="seq 33 40"
    fs=sddn10
elif [ "${1}" == "ddn11" ]; then
    nodes="seq 41 48"
    fs=sddn11
elif [ "${1}" == "ddn12" ]; then
    nodes="seq 49 56"
    fs=sddn12
elif [ "${1}" == "ddn13" ]; then
    nodes="seq 57 64"
    fs=sddn13
elif [ "${1}" == "ddn14" ]; then
    nodes="seq 65 72"
    fs=sddn14
elif [ "${1}" == "ddn15" ]; then
    nodes="seq 73 80"
    fs=sddn15
elif [ "${1}" == "ddn16" ]; then
    nodes="seq 81 88"
    fs=sddn16
elif [ "${1}" == "ddn17" ]; then
    nodes="seq 89 96"
    fs=sddn17
elif [ "${1}" == "ddn18" ]; then
    nodes="seq 97 104"
    fs=sddn18
elif [ "${1}" == "ddn19" ]; then
    nodes="seq 105 112"
    fs=sddn19
```

```

elif [ "${1}" == "ddn20" ]; then
    nodes="seq 113 120"
    fs=sddn20
elif [ "${1}" == "ddn21" ]; then
    nodes="seq 121 128"
    fs=sddn21
elif [ "${1}" == "ddn22" ]; then
    nodes="seq 129 136"
    fs=sddn22
else
    echo "Unknown building block"
    exit
fi

if [[ $2 == "" ]]; then
    echo "parm 2 must be dd count value"
    exit
fi

# Set up a directory to collect the results
dt=`date +%h%d%y-%H_%M_%S`
results=/tmp/${dt}.${1}.rd
mkdir -p ${results}

for i in `${nodes}`
do
    ping -c1 fs${i}-data >/dev/null 2>&1
    ((? != 0)) && continue
    ssh fs${i}-data mount | grep /gpfs/${fs} >/dev/null 2>&1
    ((? != 0)) && echo Aborting: /gpfs/${fs} not mounted on fs${i}-data && exit 1
done

for i in `${nodes}`
do
    ssh fs${i}-data time dd of=/dev/null if=/gpfs/${fs}/ddtest/file${i} bs=4M
count=$2 > ${results}/fs${i}-data.out 2>&1 &
done
wait
echo results are in ${results}
grep copied ${results}/*.out

```

9.11.2 NSDPerf scripts and commands

Example 9-7 contains a sample NSDperf server script. This script will start, stop, or return the status of the NSDPerf server for all systems listed in the testnodes file. It assumes ssh keys for the root ID have been added to all servers, so passwords are not required.

Example 9-7 NSDperf server script

```

#!/bin/bash

if [ "${1}" == "start" ]; then
    for i in `cat testnodes`
    do

```

```

    echo "Starting server on ${i}"
    ssh ${i} "nsdperf -s < /dev/null > /dev/null 2>&1 &"
done
elif [ "${1}" == "status" ]; then
    for i in `cat testnodes`
    do
        echo -n "${i} - "
        nprocs=`ssh ${i} "ps -ef | grep nsdperf | egrep -v grep | wc -l " `
        if (($nprocs == 0 )); then
            echo The server is not running: $nprocs
        elif (($nprocs == 1 )); then
            echo The server is running: $nprocs
        elif (($nprocs > 1 )); then
            echo Error - there are $nprocs servers running.
        fi
    done
elif [ "${1}" == "stop" ]; then
    for i in `cat testnodes`
    do
        echo Stopping server on ${i}
        ssh ${i} "ps -ef | grep nsdperf | awk '{ print \$2 }' | xargs kill -9 "
    done
fi

```

Example 9-8 provides a sample NSDPerf commands file. The commands file can be used to execute a list of NSDPerf commands by using the following command at the NSDPerf prompt:

```
source <command file name>
```

Example 9-8 NSDperf commands file

```

server fs1-data
server fs2-data
server fs3-data
server fs4-data
server fs5-data
server fs6-data
server fs7-data
server fs8-data
server fs17-data
server fs18-data
server fs19-data
server fs20-data
server fs21-data
server fs22-data
server fs23-data
server fs24-data
client 172.16.8.1
client 172.16.8.2
client 172.16.8.3
client 172.16.8.4
client 172.16.8.5
client 172.16.8.6
client 172.16.8.7
client 172.16.8.8
client 172.16.8.17

```

```
client 172.16.8.18  
client 172.16.8.19  
client 172.16.8.20  
client 172.16.8.21  
client 172.16.8.22  
client 172.16.8.23  
client 172.16.8.24  
tt 120  
test  
quit
```

Archived

Archived

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 191. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *Blue Gene/L: Performance Analysis Tools*, SG24-7278
- ▶ *IBM System Blue Gene Solution: Blue Gene/P 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

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Archived



IBM System Blue Gene Solution: Performance Analysis Tools



Redpaper™

PAPI installation and usage

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.

Tools to visualize and analyze your performance data

The first section of this paper provides an overview of the Performance Application Programming Interface (PAPI). In this section, we describe the Blue Gene implementation of PAPI and usage.

I/O node and GPFS performance tuning

The second part of the paper gives a detailed description of the IBM High Performance Computing Toolkit for the Blue Gene/P system. Included in this section is a discussion of the MPI Profiler and Tracer, CPU profiling, the Hardware Performance Monitor, and the Modular I/O library.

In the final section of the book are tips for tuning I/O performance and the IBM General Parallel File System (GPFS).

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