

ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

Expert team | Trust guarantee | 100% satisfaction

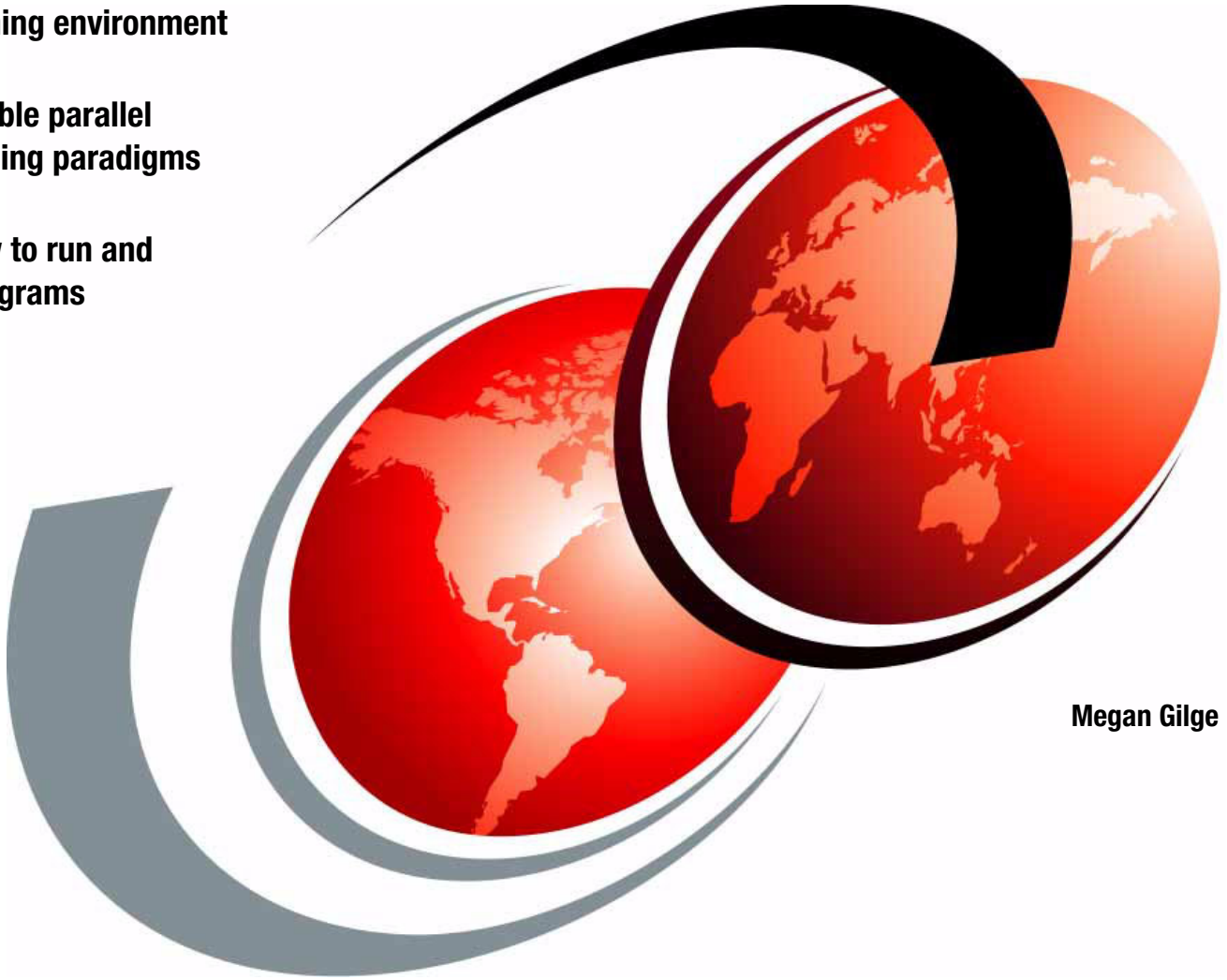
All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Find the *IBM 00E5867* at our website: *Click HERE*

IBM

IBM System Blue Gene Solution Blue Gene/Q Application Development

- Understand the Blue Gene/Q programming environment
- See available parallel programming paradigms
- Learn how to run and debug programs



Megan Gilge

Redbooks

ibm.com/redbooks

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



International Technical Support Organization

IBM System Blue Gene Solution: Blue Gene/Q Application Development

June 2013

SG24-7948-01

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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

Second Edition (June 2013)

This edition applies to Version 1, Release 1, Modification 2 of IBM Blue Gene/Q (product number 5733-BGQ).

© Copyright International Business Machines Corporation 2012, 2013. All rights reserved.
Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Contents

Notices vii
Trademarks ix

Preface xi
Author. xi
Now you can become a published author, too! xii
Comments welcome. xii
Stay connected to IBM Redbooks xiii

Summary of changes xv
June 2013, Second Edition xv

Chapter 1. System overview 1
1.1 Blue Gene/Q environment overview 2
1.2 Blue Gene/Q hardware overview 3
1.3 Blue Gene/Q software overview 4
1.3.1 System administration and management 4
1.3.2 Compute Node Kernel and services 4
1.3.3 I/O node kernel and services 5
1.3.4 Message Passing Interface. 5
1.3.5 Compilers 6
1.3.6 Application development and debugging 6

Chapter 2 Kernel functionality 9
2.1 Compute Node Kernel. 10
2.1.1 Stateless compute nodes 11
2.1.2 Firmware 11
2.2 Role of the I/O node kernel 12

Chapter 3. Processes 13
3.1 Importance of process count. 14
3.2 Process creation 14
3.3 Processes per node 14
3.4 Determining how many processes per node to use 15
3.5 Specifying process count 15
3.6 Support for 64-bit applications 16
3.7 Object identifiers 16
3.7.1 Process identifier 16
3.7.2 Thread identifier 17
3.7.3 Thread group identifier 17
3.7.4 T coordinate 17
3.8 Sub-node jobs. 17
3.9 Threading overview. 17
3.9.1 Hardware thread over-commitment. 17
3.10 Thread scheduler 18
3.10.1 Thread preemption 18
3.10.2 Thread yield 19
3.10.3 Round-robin dispatch 19
3.11 Thread affinity 19

© Copyright IBM Corp. 2013. All rights reserved.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

3.11.1 Breadth-first assignment . . . 19
3.11.2 Depth-first assignment . . . 19
3.11.3 Thread affinity control . . . 19
3.11.4 Setting affinity with the pthread attribute . . . 20
3.11.5 Setting affinity with the system call . . . 20
3.11.6 Extended thread affinity control . . . 20
3.12 Thread priority . . . 21
3.12.1 Setting priority through the pthread attribute . . . 22
3.12.2 Explicit setting of priority . . . 23
3.12.3 Hardware thread priority . . . 23

Chapter 4. Memory . . . 25
4.1 Memory system overview . . . 26
4.1.1 L1 prefetch cache overview . . . 26
4.1.2 L2 cache functional overview . . . 28
4.1.3 Boot eDRAM overview . . . 28
4.2 Memory management . . . 29
4.3 Memory protection . . . 29
4.4 Shared memory . . . 29
4.5 Persistent memory . . . 30
4.6 Compute node ramdisk . . . 30
4.7 Support for the /proc file system . . . 31
4.8 L1P prefetcher . . . 32
4.8.1 Linear stream prefetcher overview . . . 33
4.8.2 Perfect prefetcher overview . . . 34
4.8.3 L1P prefetcher API descriptions . . . 38
4.8.4 Performance considerations . . . 52
4.9 L2 atomic operations . . . 53
4.10 Speculative execution . . . 53
4.11 Support for dynamic linking . . . 54
4.12 Transactional memory . . . 54

Chapter 5. Compute Node Kernel interfaces . . . 55
5.1 Lightweight principles . . . 56
5.2 Kernel access . . . 56
5.2.1 Application programming interfaces . . . 56
5.2.2 System programming interface . . . 62
5.3 System calls . . . 63

Chapter 6. Parallel paradigms . . . 65
6.1 Programming model . . . 66
6.2 Blue Gene/Q MPI implementation . . . 67
6.2.1 High-performance network for efficient parallel execution . . . 67
6.2.2 Forcing MPI to allocate too much memory . . . 69
6.2.3 Not waiting for the MPI_Test function . . . 70
6.2.4 Flooding the network with messages . . . 70
6.2.5 Deadlocking the system . . . 70
6.2.6 Violating MPI buffer ownership rules . . . 71
6.2.7 Buffer alignment sensitivity . . . 71
6.3 Blue Gene/Q MPI extensions . . . 72
6.3.1 Changing class-route usage at run time . . . 72
6.3.2 Determining hardware properties . . . 73
6.4 MPI functions . . . 73
6.5 Compiling MPI programs on the Blue Gene/Q system . . . 74

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

6.6 OpenMP 77
6.6.1 OpenMP implementation for Blue Gene/Q 77
6.7 Multiple Program, Multiple Data 77

Chapter 7. Developing applications with Blue Gene/Q compilers 79
7.1 Programming environment overview 80
7.2 Compilers for the Blue Gene/Q system 80
7.2.1 IBM XL compilers 80
7.2.2 GNU Compiler Collection 81
7.2.3 Python interpreter 81
7.2.4 Toolchain tools 81
7.3 Compiling and linking applications on the Blue Gene/Q system 81
7.4 Compiler options specific to the Blue Gene/Q system 82
7.4.1 Options for the Blue Gene/Q system 82
7.4.2 Unsupported compiler options 83
7.5 Support for pthreads and OpenMP 83
7.5.1 Thread stack size for the Blue Gene/Q system 84
7.6 Creating libraries on the Blue Gene/Q system 85
7.7 Running dynamically linked applications on the Blue Gene/Q system 86
7.7.1 Creating a program 86
7.7.2 Creating a shared library 87
7.7.3 Running a Blue Gene/Q dynamically linked program on a front end node 87
7.7.4 Running a dynamically linked program on the Blue Gene/Q system 87
7.7.5 Tools for dynamic linking 88
7.8 Mathematical Acceleration Subsystem Libraries 92
7.9 Engineering and Scientific Subroutine Libraries 92
7.10 Cross-compilation on the Blue Gene/Q system 92
7.10.1 Configuring and building on an I/O node used as a front end node 93
7.10.2 Using implicit program launching from a front end node 93
7.11 Python support 95
7.11.1 Using the Python interpreter in a cross-compiled environment 95
7.11.2 Running the Python interpreter on the Blue Gene/Q system 96
7.12 Using the QPX floating-point unit 97
7.12.1 Using SIMD instructions in applications 98

Chapter 8. Running and debugging applications 103
8.1 Running applications 104
8.1.1 IBM LoadLeveler 104
8.2 Debugging applications 104
8.2.1 General debugging architecture 105
8.2.2 GNU Project Debugger 105
8.2.3 Coreprocessor debugger 108
8.2.4 The addr2line utility 109
8.3 What to do when a job fails 112
8.4 Debugging jobs 113
8.4.1 The snapbug tool 113
8.4.2 The Coreprocessor tool 113

Appendix A. Mapping 115
Mapping overview 116
General guidance 117

Appendix B. Blue Gene/Q personality 121
Personality of Blue Gene/Q nodes 122

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Examples of retrieving Blue Gene/Q personality information 122

Appendix C. PAMI and MPI header files and libraries 125
Blue Gene/Q applications 126

Appendix D. MPI and CNK environment variables 129
Message Passing Interface environment variables 130
Compute Node Kernel environment variables 142
Setting environment variables 145

Appendix E. Using GNU profiling 147
Using the Blue Gene/Q gmon tool 148
Specifying which ranks generate gmon.out files 148
Functions to disable gmon.out files for some nodes 148
Profiling for threads 148
Profiling with the GNU toolchain 148
Using timer tick (machine instruction level) profiling 149
Collecting call count information 149

Appendix F. Hardware performance counters 151
Blue Gene Hardware Performance Monitoring API 152
Performance Application Programming Interface 153

Appendix G. Requirements for C++ programming in a failover environment 155

Abbreviations and acronyms 157

Related publications 159
IBM Redbooks 159
Other publications 159
Online resources 160
How to get IBM Redbooks 160
Help from IBM 161

References 163

Index 165

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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.

IBM DOES NOT WARRANT OR REPRESENT THAT THE CODE PROVIDED IS COMPLETE OR UP-TO-DATE. IBM DOES NOT WARRANT, REPRESENT OR IMPLY RELIABILITY, SERVICEABILITY OR FUNCTION OF THE CODE. IBM IS UNDER NO OBLIGATION TO UPDATE CONTENT NOR PROVIDE FURTHER SUPPORT.

ALL CODE IS PROVIDED "AS IS," WITH NO WARRANTIES OR GUARANTEES WHATSOEVER. IBM EXPRESSLY DISCLAIMS TO THE FULLEST EXTENT PERMITTED BY LAW ALL EXPRESS, IMPLIED, STATUTORY AND OTHER WARRANTIES, GUARANTEES, OR REPRESENTATIONS, INCLUDING, WITHOUT LIMITATION, THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF PROPRIETARY AND INTELLECTUAL PROPERTY RIGHTS. YOU UNDERSTAND AND AGREE THAT YOU USE THESE MATERIALS, INFORMATION, PRODUCTS, SOFTWARE, PROGRAMS, AND SERVICES, AT YOUR OWN DISCRETION AND RISK AND THAT YOU WILL BE SOLELY RESPONSIBLE FOR ANY DAMAGES THAT MAY RESULT, INCLUDING LOSS OF DATA OR DAMAGE TO YOUR COMPUTER SYSTEM.

IN NO EVENT WILL IBM BE LIABLE TO ANY PARTY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY OR CONSEQUENTIAL DAMAGES OF ANY TYPE WHATSOEVER RELATED TO OR ARISING FROM USE OF THE CODE FOUND HEREIN, WITHOUT LIMITATION, ANY LOST PROFITS, BUSINESS INTERRUPTION, LOST SAVINGS, LOSS OF PROGRAMS OR OTHER DATA, EVEN IF IBM IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THIS EXCLUSION AND WAIVER OF LIABILITY APPLIES TO ALL CAUSES OF ACTION, WHETHER BASED ON CONTRACT, WARRANTY, TORT OR ANY OTHER LEGAL THEORIES.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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.

© Copyright IBM Corp. 2013. All rights reserved.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Trademarks

IBM, the IBM logo, and 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®
Blue Gene/P®
Blue Gene/Q®
Blue Gene®
GPFS™
IBM®
LoadLeveler®
PowerPC®
Power®
Redbooks®
Redpapers™
Redbooks (logo) ®
Tivoli®

The following terms are trademarks of other companies:

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries 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.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Preface

This IBM® Redbooks® publication is one in a series of IBM books written specifically for the IBM System Blue Gene® supercomputer, Blue Gene/Q®, which is the third generation of massively parallel supercomputers from IBM in the Blue Gene series. This document provides an overview of the application development environment for the Blue Gene/Q system. It describes the requirements to develop applications on this high-performance supercomputer.

This book explains the unique Blue Gene/Q programming environment. This book does not provide detailed descriptions of the technologies that are commonly used in the supercomputing industry, such as Message Passing Interface (MPI) and Open Multi-Processing (OpenMP). References to more detailed information about programming and technology are provided.

This document assumes that readers have a strong background in high-performance computing (HPC) programming. The high-level programming languages that are used throughout this book are C/C++ and Fortran95. For more information about the Blue Gene/Q system, see “IBM Redbooks” on page 159.

Author

This book was produced by a team working at the International Technical Support Organization (ITSO), Rochester Center.

Megan Gilge is a Technical Writer in the IBM International Technical Support Organization. Before joining the ITSO one year ago, Megan was an Information Developer in the IBM Semiconductor Solutions and User Technologies areas. Megan holds a B.A. in Liberal Arts from Michigan Technological University.

Thanks to the following people for their contributions to this project:

Robert E. Walkup
IBM Research

John Attinella
Mike Blocksome
Lynn Boger
Thomas Budnik
Kristan Davis
Mitchell Felton
Thomas Gooding
Jerrold Heyman
Kerry Kaliszewski
Gary Lakner
Tom Liebsch
Mike Nelson
Jeffrey Parker

© Copyright IBM Corp. 2013. All rights reserved.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Brian Smith
IBM Systems & Technology Group

Annette Bauerle
International Technical Support Organization

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

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 books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

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

  `ibm.com/redbooks`

- Send your comments in an email 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

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Stay connected to IBM Redbooks

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks

- Follow us on Twitter:

  http://twitter.com/ibmredbooks

- Look for us on LinkedIn:

  http://www.linkedin.com/groups?home=&gid=2130806

- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm

- Stay current on recent Redbooks publications with RSS Feeds:

  http://www.redbooks.ibm.com/rss.html

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Summary of changes

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

Summary of Changes
for SG24-7948-01
for IBM System Blue Gene Solution: Blue Gene/Q Application Development
as created or updated on February 12, 2014.

June 2013, Second Edition

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

New information

- Added information 3.11.6, “Extended thread affinity control” on page 20.
- Added 4.6, “Compute node ramdisk” on page 30
- Added 4.7, “Support for the /proc file system” on page 31
- Added 4.8, “L1P prefetcher” on page 32
- Added information about kernel SPI documentation to 5.2.2, “System programming interface” on page 62.
- Added information about rdma.h and sendx.h to Table 5-4 on page 63.
- Added 5.3, “System calls” on page 63.
- Added 6.7, “Multiple Program, Multiple Data” on page 77.
- Added 8.4, “Debugging jobs” on page 113.
- Added Appendix G, “Requirements for C++ programming in a failover environment” on page 155.

Changed information

- Removed PAMID_COLLECTIVES_SELECTION from Table D-1 on page 130. Updated information for the PAMI_GLOBAL_SHMEMSIZE, PAMI_ROUTING, PAMID_DISABLE_INTERNAL_EAGER_TASK_LIMIT, PAMID_EAGER_LOCAL, PAMID_RZV_LOCAL, PAMID_EAGER, PAMID_RZV, PAMID_PT2PT_LIMITS, PAMID_RMA_PENDING, PAMID_SHORT variables.
- Updated information about the BG_COREDUMPBINARY, BG_COREDUMPMAXNODES, BG_COREDUMPRANKS, BG_MAPCOMMONHEAP, BG_MAPNOALIASES, BG_MAPALIGN16, BG_THREADLAYOUT, and BG_THREADMODEL variables in Table D-5 on page 142.

© Copyright IBM Corp. 2013. All rights reserved.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

1

System overview

This chapter provides an overview of the IBM Blue Gene/Q system and its software environment. It includes the following sections:

- Blue Gene/Q environment overview
- Blue Gene/Q hardware overview
- Blue Gene/Q software overview

© Copyright IBM Corp. 2013. All rights reserved.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

1.1 Blue Gene/Q environment overview

The Blue Gene/Q system, shown in Figure 1-1, is the third-generation computer architecture in the Blue Gene family of supercomputers.

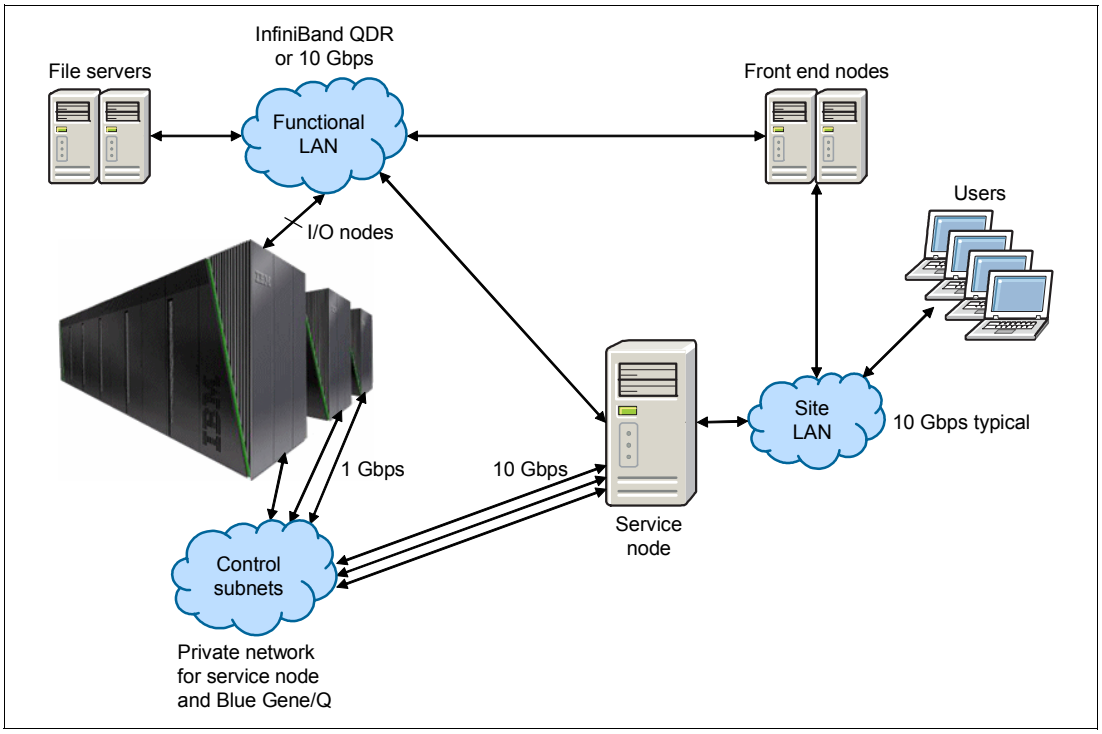


Figure 1-1 Blue Gene/Q system architecture

The Blue Gene/Q system comprises multiple components including one or more compute racks and optionally I/O racks. The system contains densely packaged compute nodes, I/O drawers, and service cards. Additional hardware is associated with the storage subsystem, the primary service node (SN), the front end nodes (FENs), and the communications subsystem. The I/O drawers containing I/O nodes connect to the functional local area network (LAN) to communicate with file servers, FENs, and the SN. The service cards connect to the control subnets and are used by the SN to control the Blue Gene/Q hardware.

A service node provides a single point of control and administration for the Blue Gene/Q system. It is possible to operate a Blue Gene/Q system with a single service node. However, the system environment can also be configured to include distributed subnet service nodes (SSN) for high scalability. System administration is outside the scope of this book and is covered in the *IBM System Blue Gene Solution: Blue Gene/Q System Administration*, SG24-7869 Redbooks publication.

A front end node, also known as a login node, comprises the system resources that application developers log in to for access to the Blue Gene/Q system. Application developers edit and compile applications, create job control files, launch jobs on the Blue Gene/Q system, post-process output, and perform other interactive activities.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

1.2 Blue Gene/Q hardware overview

Figure 1-2 shows the primary hardware components of the Blue Gene/Q system.

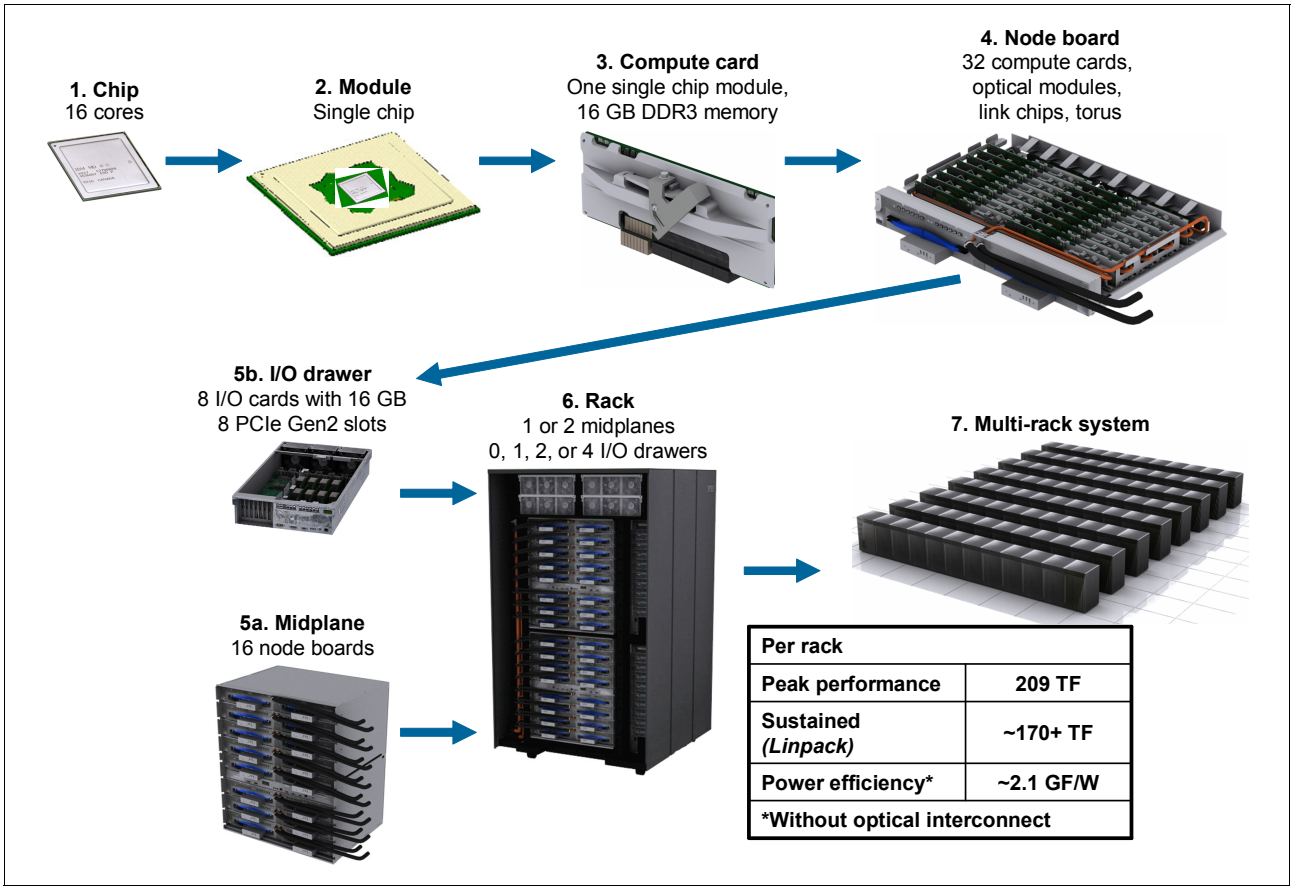


Figure 1-2 Blue Gene/Q hardware overview

Compute cards contain 16 IBM Blue Gene/Q PowerPC® A2 core processors and 16 GB of memory. Thirty-two such cards plug into a node board and 16 node boards are contained in a midplane. A Blue Gene/Q compute rack has either one (half rack configuration) or two fully populated midplanes. The system can be scaled to 512 compute racks.

Compute racks components are cooled either by water or air. Water is used for the processing nodes. Air is used for the power supplies and the I/O drawers mounted in the Blue Gene/Q rack.

I/O drawers are either in separate racks or in I/O enclosures on top of the compute racks, sometimes described as *top hats*. Eight I/O nodes are housed in each I/O drawer. In the compute rack, up to four I/O drawers, two per midplane, can be configured using the I/O enclosure (top hat). The placement of I/O drawers in the I/O rack configuration is advisable in a large system installation where the number of I/O nodes cannot be accommodated in the compute racks.

For an introduction to the Blue Gene/Q hardware components, see the *Blue Gene/Q Hardware Overview and Installation Planning Guide*, SG24-7822 Redbooks publication.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

1.3 Blue Gene/Q software overview

The Blue Gene/Q software includes the following features:

- Scalable Blue Gene/Q system administration and management services running on service nodes, subnet service nodes, and front end nodes
- Compute Node Kernel (CNK) running on the compute nodes
- Full Linux kernel running on I/O nodes
- Message Passing Interface (MPI) between compute nodes through MPI library support
- Open multi-processing (OpenMP) application programming interface (API)
- Support for the standard IBM XL family of compilers with XLC/C++, XLF, and the GNU Compiler Collection
- Software support that includes IBM Tivoli® Workload Scheduler LoadLeveler®, IBM General Parallel File System (GPFS™), and Engineering and Scientific Subroutine Library (ESSL)
- Support for running Python applications
- Support for debuggers including GNU Project Debugger (GDB)

1.3.1 System administration and management

The responsibilities of a Blue Gene/Q system administrator can be wide-ranging, but the administrator typically maintains and monitors the health of the Blue Gene/Q system. Most of the system administrator tasks are performed from the service node. The Navigator web application that runs on the service node plays an important role in helping administrators perform their job. The *IBM System Blue Gene Solution: Blue Gene/Q System Administration*, SG24-7869 Redbooks publication provides a comprehensive description of administering a Blue Gene/Q system, including how to use the key features of Navigator, manage compute and I/O blocks, run diagnostics, perform service actions, use the console, handle alerts, manage various servers, submit and manage jobs, and configure I/O nodes.

1.3.2 Compute Node Kernel and services

The Compute Node Kernel (CNK) software is an operating system that is similar to Linux and provides an environment for running user processes on compute nodes. The CNK includes the following services:

- Process creation and management
- Memory management
- Process debugging
- Reliability, availability, and serviceability (RAS) management
- File I/O
- Network

The Blue Gene/Q software stack includes a standard set of runtime libraries for C, C++, and Fortran. To the extent that is possible, the supported functions maintain open standard Portable Operating System Interface (POSIX)-compliant interfaces. The CNK has a robust threading implementation on the Blue Gene/Q system that supports pthread, XL OpenMP, and GNU OpenMP implementations. The Native POSIX Thread Library (NPTL) pthreads implementation in the GNU C Library (GLIBC) runs without modification.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Although statically linked executable programs provide optimal performance, the CNK also has support for dynamically linked executable programs. This support enables dynamically linked scripting languages, such as Python, to be used in CNK environments.

For more information about the Compute Node Kernel, see 2.1, “Compute Node Kernel” on page 10.

1.3.3 I/O node kernel and services

The I/O node kernel is a patched Red Hat Enterprise Linux 6 kernel running on I/O nodes. The patches provide support for the Blue Gene/Q platform and contain modifications to improve performance.

The I/O node software provides I/O services to compute nodes. For example, applications that are running on compute nodes can access file servers and communicate with processes in other machines. The I/O nodes also play an important role in starting and stopping jobs and in coordinating activities with debug and monitoring tools.

Blue Gene/Q is a diskless system, so file servers must be present. A high-performance parallel file system is expected. The Blue Gene/Q system is flexible and accepts various file systems that are supported by Linux. Typical parallel file systems are the IBM General Parallel File System (GPFS) and Lustre.

The I/O node includes a complete internet protocol (IP) stack with Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) services. A subset of these services is available to user processes running on the compute nodes that are associated with an I/O node. Application processes communicate with processes that are running on other systems with client-side sockets. Support for server-side sockets is also provided. The I/O node implements the sockets so that a group of compute nodes behave as though the compute tasks are running on the I/O node. In particular, this means that the socket port number is a single address space within the group. The compute nodes share the IP address of the I/O node.

The I/O node kernel is designed to be booted as infrequently as possible. The bootstrap process includes loading a ramdisk image and booting the Linux kernel. The ramdisk image is extracted to provide the initial file system. This system contains minimal commands to mount the file system on the service node using the Network File System (NFS). The boot continues by running startup scripts from the NFS. It also runs customer-supplied startup scripts to perform site-specific actions, such as logging configuration and mounting high-performance file systems.

Toolchain shared libraries and all of the basic Linux text and shell utilities are local to the ramdisk. Packages, such as GPFS, and customer-provided scripts are NFS mounted for administrative convenience.

A complete description of the I/O node software is provided in the *IBM System Blue Gene Solution: Blue Gene/Q System Administration*, SG24-7869 Redbooks publication.

1.3.4 Message Passing Interface

The implementation of the Message Passing Interface (MPI) on the Blue Gene/Q system is the MPICH2 standard that was developed by Argonne National Labs. For more information about MPICH2, see the Message Passing Interface (MPI) standard website at:

`http://www-unix.mcs.anl.gov/mpi/`

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The dynamic process management function (creating new MPI processes) of the MPI-2 standard is not supported by the Blue Gene/Q system. However, the various thread modes are supported.

1.3.5 Compilers

The Blue Gene/Q toolchain compilers and the IBM XL compilers for Blue Gene/Q compute nodes are available for use on the Blue Gene/Q system. Because compilation occurs on the front end node and not the Blue Gene/Q system, the compilers for the Blue Gene/Q system are cross-compilers. See 7.2, “Compilers for the Blue Gene/Q system” on page 80 for more information about compilers.

GNU compilers

The compilers in the Blue Gene/Q toolchain are based on the GNU compilers. When installing the Blue Gene/Q software, RPM package managers (RPMs) are provided so that the user can build and install the Blue Gene/Q toolchain into the *gnu-linux* directory of the software stack. The Blue Gene/Q toolchain compilers are used to build much of the Blue Gene/Q system software and provide the base libraries for user applications. They can be used to build applications to run on the Blue Gene/Q compute nodes. See 7.2.2, “GNU Compiler Collection” on page 81 for more information about the GNU compilers.

IBM XL compilers

The IBM XL compilers for Blue Gene/Q can be used to build applications that run on the Blue Gene/Q system. The IBM XL compilers can provide higher levels of optimization than the Blue Gene/Q toolchain compilers. The XL compilers for Blue Gene/Q support single instruction, multiple data (SIMD) vectorization (*simdization)*. Simdization enables automatic code generation to use the quad floating-point unit (FPU) of the Blue Gene/Q system. This unit can handle four simultaneous floating-point instructions. The Blue Gene/Q XL compilers also provide support for source code syntax to use transactional memory and speculative threads. See 7.2.1, “IBM XL compilers” on page 80 for more information about the IBM XL compilers.

MPI wrapper scripts for Blue Gene/Q compilers

The MPI wrapper scripts are compiler wrapper scripts that are provided in the Blue Gene/Q driver. These scripts can be used to compile and link programs that use MPI. Various MPI scripts are available, depending on which compiler is used to compile the code and the version of the libraries to be linked. The wrapper scripts start the appropriate compiler and add all necessary directories, libraries, and options that are required to compile programs for MPI. For each compiler language and standard that is provided for the Blue Gene/Q system, there is a corresponding MPI wrapper script. There are also thread-safe versions for each of the IBM XL compilers. The MPI wrapper scripts are described in 6.5, “Compiling MPI programs on the Blue Gene/Q system” on page 74.

For more detailed compiler information, see Chapter 7, “Developing applications with Blue Gene/Q compilers” on page 79.

1.3.6 Application development and debugging

Application developers access front end nodes to compile and debug applications, submit Blue Gene/Q jobs, and perform other interactive activities.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Debuggers

The Blue Gene/Q system includes support for running GNU Project Debugger (GDB) with applications that run on compute nodes. Other third-party debuggers are also available. See 8.2, “Debugging applications” on page 104.

Running applications

Blue Gene/Q applications can be run in several ways. The most common method is to use a job scheduler that supports the Blue Gene/Q system, such as the LoadLeveler scheduler. Another less common option is to use the **`runjob`** command directly. All Blue Gene/Q job schedulers use the **`runjob`** interface for job submission, but schedulers can wrap it with another command or job submission interface. The **`runjob`** command is described in the *IBM System Blue Gene Solution: Blue Gene/Q System Administration*, SG24-7869 Redbooks publication.

For more information about the LoadLeveler scheduler, see 8.1.1, “IBM LoadLeveler” on page 104.

Application memory considerations

On the Blue Gene/Q system, the entire physical memory of a compute node is 16 GB, so careful consideration of memory is required when writing applications. Some of that space is allocated for the CNK. Shared memory space is also allocated to the user process at the time the process is created.

The CNK tracks collisions of the stack and heap as the heap is expanded with brk() and mmap() system calls. The CNK and its private data are protected from reads and writes by the user process or threads. The code space of the process is protected from writing by the process or threads. Code and read-only data are shared between the processes that share each node.

The amount of memory required by the application is an important topic for Blue Gene/Q. The memory used by an application falls into one of the following classifications:

bss	Uninitialized static and common variables
data	Initialized static and common variables
heap	Controlled allocatable arrays
stack	Controlled automatic arrays and variables
text	Application text (instructions) and read-only data

The Blue Gene/Q system implements a 64-bit memory model. You can use the Linux **`size`** command to display the memory size of the program. However, the **`size`** command does not provide any information about the runtime memory usage of the stack or heap.

The memory that is available to the application depends on the number of processes per node. The 16 GB of available memory is partitioned as evenly as possible among the processes on each node. Because memory is a limited resource, it is generally advisable to conserve memory in the application. In some cases, the memory requirement can be reduced by distributing data that was replicated in the original code. However, additional communication might be required. On Blue Gene/Q systems, the total number of processes can be large. Consider the memory that is required to store arrays that have the number of processes as one or more of the array dimensions.

Other considerations

It is important to understand that the operating system present on the compute node, the CNK, is not a full version of the Linux operating system. Therefore, use care in the areas

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

explained in the following sections when writing applications for the Blue Gene/Q system. For a full list of supported system calls, see 5.3, “System calls” on page 63.

Input and output

Pay special attention to I/O in your application. The CNK does not perform I/O. I/O is managed by the I/O node.

File I/O

A limited set of file I/O is supported. Do *not* attempt to use asynchronous file I/O because it causes runtime errors.

Standard input

Standard input (stdin) is supported on the Blue Gene/Q system.

Socket calls

Socket calls are supported on the Blue Gene/Q system. For more information, see Chapter 5, “Compute Node Kernel interfaces” on page 55.

Linking

Dynamic linking is supported on the Blue Gene/Q system. You can statically link all code into your application or use dynamic linking.

Shell scripts

The CNK does not provide a mechanism for a command interpreter or shell when applications start on the Blue Gene/Q system. Only the executable program can be started. Therefore, if the application includes shell scripts that control workflow, the workflow must be adapted. For example, an application workflow shell script cannot be started with the **`runjob`** command. Instead, run the application workflow scripts on the front end node and start the **`runjob`** command only at the innermost shell script level where the main application binary is called.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

2

Kernel functionality

The kernel provides the glue that makes all components in Blue Gene/Q system work together. This chapter provides an overview of the functionality that is implemented as part of the Compute Node Kernel (CNK) and the I/O node kernel, which includes information about the following topics:

- Compute Node Kernel
- Role of the I/O node kernel

© Copyright IBM Corp. 2013. All rights reserved.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

2.1 Compute Node Kernel

The CNK is a flexible, lightweight kernel for Blue Gene/Q compute nodes that can support diagnostic modes and user applications. It provides an operating system that is similar to the Linux operating system and supports a large subset of Linux-compatible system calls. This subset is based on the IBM Blue Gene/P® system, which demonstrated good compatibility and portability with the Linux operating system. The CNK is tuned for the capabilities and performance of the Blue Gene/Q application-specific integrated circuit (ASIC).

As part of the Blue Gene/Q system, the CNK supports threads and dynamic linking for further compatibility with the Linux operating system. Figure 2-1 shows the interaction between the application space and the kernel space.

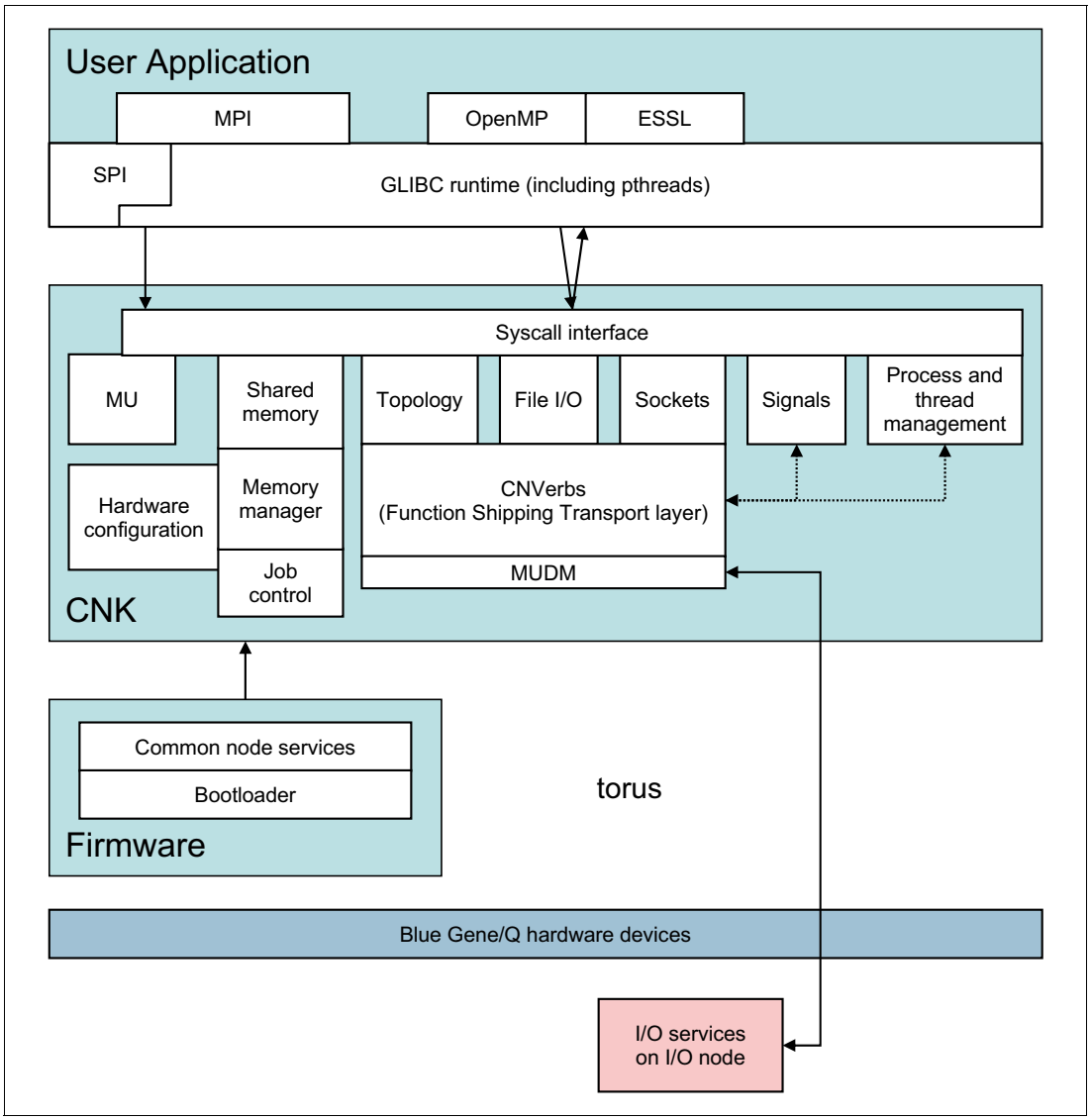


Figure 2-1 Compute Node Kernel overview

When running a user application, the CNK connects to the I/O node through the torus network. This connection communicates to a set of processes called *Control and I/O Services (CIOS)* that run on the I/O node. All function-shipped system calls are forwarded to the CIOS process and executed on the I/O node.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

At the user-application level, the Compute Node Kernel supports the following application programming interfaces (APIs) among others:

- Message Passing Interface (MPI) support between nodes using MPI library support
- Open multi-processing (OpenMP) API
- Standard IBM XL family of compilers support with XLC/C++, XLF, and GNU Compiler Collection
- Highly optimized mathematical libraries, such as IBM Engineering and Scientific Subroutine Library (ESSL)
- GNU Compiler Collection (GCC) C Library, or GLIBC, which is the C standard library and the GCC interface.

The CNK provides the following services:

- Torus direct memory access (DMA), which provides memory access for reading, writing, or doing both independently of the processing unit. The DMA torus interfaces are made available to the user space, which allows communication libraries to send messages directly from the application without involving the kernel. The kernel, with the hardware, implements secure limit registers that prevent the DMA from targeting memory outside the application. These constraints, along with the electrical partitioning of the torus, provide security between applications.
- Shared-memory access on a local node
- Hardware configuration
- Memory management
- MPI topology
- File I/O
- Sockets connection
- Signals
- Thread management
- Transport layer through the torus network

2.1.1 Stateless compute nodes

The Blue Gene/Q hardware is a stateless system with no embedded read-only memories (ROMs) or resident basic input/output system (BIOS). When the hardware is reset, the Control System must load the operating system into the memory of each compute node. It accomplishes this process in two phases:

- Phase 1 loads a small *firmware* component into the embedded random access memory (RAM) on each compute node. This firmware starts executing and initializes critical pieces of the Blue Gene/Q chip.
- Phase 2 communicates over a custom protocol to download the rest of the kernel images. These kernel images are then executed, allowing for connectivity over the torus network.

2.1.2 Firmware

The firmware component provides low-level services that are both specific to Blue Gene and common to the Linux operating system and the Compute Node Kernel. As such, these services provide a consistent implementation across node types while insulating the kernels from the details of the Control System. The common node services provide the same

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

low-level hardware initialization and setup interfaces to both the Linux operating system and the Compute Node Kernel.

2.2 Role of the I/O node kernel

The I/O node kernel provides I/O services to compute nodes and runs on I/O nodes. The I/O nodes also play an important role in starting and stopping jobs and in coordinating activities with debug and monitoring tools.

The operating system that runs on the I/O nodes is a distribution of Red Hat Enterprise Linux 6 for IBM PowerPC. *IBM System Blue Gene Solution: Blue Gene/Q System Administration*, SG24-7869 describes setup and configuration of the operating system. The I/O node is not apparent to the application, but it is important to consider the I/O node when tuning I/O performance.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

3

Processes

The compute nodes on the Blue Gene/Q system comprise 17 cores on a single chip with 16 GB of dedicated physical memory. Applications run on 16 of the cores with the 17th core reserved for system software. Nearly the full 16 GB of physical memory is dedicated to application usage.

Within each job, processes (also known as tasks) are distributed to all of the compute nodes. Each node runs separate instantiations of the Compute Node Kernel (CNK). Each CNK can run multiple tasks or processes. This chapter describes some of the characteristics of those processes and how to configure them.

© Copyright IBM Corp. 2013. All rights reserved.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

3.1 Importance of process count

When submitting a job to the Blue Gene/Q system with the **runjob** command or another job scheduler command, it is important to decide how many processes or tasks to run on each node. This decision significantly impacts performance and the memory that is allocated to each process, for example:

- Running 16 processes per node divides the total number of cores and the total physical memory into 16ths. Thus, one core and roughly 1 GB is available to each task.
- Running one process with 16 threads might yield equivalent performance but causes no subdivision of the memory layout.

When a job is submitted to the Control System, the user specifies the number of processes to create. This value is used in the configuration of memory and number of cores assigned to the process.

3.2 Process creation

Jobs are submitted to the CNK through the Control System using the functional network path through the I/O node to load the application. A job can be a subset of the block. Multiple jobs comprising one or more node jobs can exist per block. However, a single job cannot span blocks. From a CNK perspective, jobs comprise one or more processes to the node. With each job, the Control System provides information that describes the environment of the job (environment variables, program arguments, and so on).

For statically linked applications, the CNK loads the applications into memory at process creation time. To ensure that the application load scales to large system configurations, the application is loaded from a job leader node and broadcast to the other nodes in the job.

For dynamically linked applications, the CNK loads the dynamic interpreter (ld.so) into memory. The interpreter then pulls the target application and associated dynamic libraries into main memory by using standard system calls.

Processes are only created at the time of job initialization. The CNK does not support the fork system call and therefore cannot dynamically spawn processes.

3.3 Processes per node

When a job is submitted to the Control System, the user specifies the number of processes to create. This value is used to configure memory and the number of cores that are assigned to the process. To specify the process count, use the **runjob** -p (or --ranks-per-node) option.

When the user specifies the process count per node, the memory on that node is divided evenly among the processes. Generally, each process has roughly the same amount of memory, although there can be slight variations. Variations can be because of the size of the application text, shared memory size, kernel storage, and so on. The CNK configures the hardware to avoid memory page translations while the application is running. This configuration is known as a *static memory map*.

To achieve a static memory map, the process count must be a power of two. Thus, the valid numbers are 1, 2, 4, 8, 16, 32, or 64 processes per node. User submitted jobs run on 16 of the 17 cores.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The CNK allocates a number of cores to a process. Therefore, as shown in Table 3-1, the number of threads that each process can have active at a given moment is dictated by the processes per node value.

Table 3-1 Processes per node

Processes per node	Number of A2 cores per process	Maximum number of active hardware threads per process
1	16	64
2	8	32
4	4	16
8	2	8
16	1	4
32	2 processes per core	2
64	4 processes per core	1

3.4 Determining how many processes per node to use

The best configuration of processes per node depends on the type of application, the memory requirement, and the parallel paradigm that is implemented. There might be several options for applications that use a hybrid paradigm with both MPI and OpenMP or pthreads, depending on the memory footprint. Hybrid applications that support a high degree of threading might work well with a single process per node, but scenarios with 2, 4, 8, or 16 processes per node are more common. For single-threaded applications, the memory requirement per process is the main consideration. If possible, use all 16 of the cores with 16, 32, or 64 processes per node.

One trade off to consider is that each additional process that is running on a node has a fixed amount of overhead. Overhead consists of a replicated data segment (for example, the global storage for the process), storage for the main stack, and storage for the heap.

3.5 Specifying process count

The default mode for the **runjob** command is one process per node. To specify other values for processes per node, use the following commands:

```
runjob ... -p 2 ...
runjob ... --ranks-per-node 2 ...
runjob ... -p 16 ...
runjob ... -p 64 ...
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

3.6 Support for 64-bit applications

The Blue Gene/Q system only supports processes compiled for the 64-bit PowerPC application binary interface (ABI). 32-bit processes are not supported:

- For compilations with the GCC compiler, the -m64 flag is the default.
- For compilations with the XL compiler, the -q64 flag is the default.

3.7 Object identifiers

There are various identifiers related to the process objects on the compute node. Each of the 16 physical cores that support the application processes is assigned a processor core identifier. Each of the four hardware threads within that core is assigned a processor thread identifier. There is also a unique identifier for each hardware thread within the node termed the processor identifier. Table 3-2 describes the interrelationship between these hardware identifiers. The first 16 cores are used to run applications. The 17th core is reserved for system use.

Table 3-2 Physical core ID, thread ID, and processor ID

Processor core ID	Processor thread ID	Processor ID
0	0, 1, 2, 3	0, 1, 2, 3
1	0, 1, 2, 3	4, 5, 6, 7
2	0, 1, 2, 3	8, 9, 10, 11
3	0, 1, 2, 3	12, 13, 14, 15
4	0, 1, 2, 3	16, 17, 18, 19
5	0, 1, 2, 3	20, 21, 22, 23
6	0, 1, 2, 3	24, 25, 26, 27
7	0, 1, 2, 3	28, 29, 30, 31
8	0, 1, 2, 3	32, 33, 34, 35
9	0, 1, 2, 3	36, 37, 38, 39
10	0, 1, 2, 3	40, 41, 42, 43
11	0, 1, 2, 3	44, 45, 46, 47
12	0, 1, 2, 3	48, 49, 50, 51
13	0, 1, 2, 3	52, 53, 54, 55
14	0, 1, 2, 3	56, 57, 58, 59
15	0, 1, 2, 3	60, 61, 62, 63
16	0, 1, 2, 3	64, 65, 66, 67

3.7.1 Process identifier

The process identifier (PID) is a 4-byte signed number that identifies a process. Each process on a compute node has a unique PID. The PID value is not unique across compute nodes.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The PID can be passed to various pthreads, signal APIs, and system calls as a thread group identifier (TGID) for the process. The thread identifier (TID) that corresponds to the primary thread of the process is the same value as the PID for the process.

3.7.2 Thread identifier

The thread identifier (TID) is a number assigned by the kernel used to uniquely identify a thread within the node. The TID of the process' main thread is the same as the PID of the process. See 3.7.1, "Process identifier" on page 16 for more information about PID number generation.

3.7.3 Thread group identifier

The thread group identifier (TGID) is an input parameter on several pthread and signal APIs and system calls. The PID (that is, the main thread TID of the process) serves as a valid TGID.

3.7.4 T coordinate

Multiple processes within a node for a given job are assigned a "T" coordinate value. That value can be used with the A, B, C, D, and E coordinates to uniquely identify a specific process or rank in the block or sub-block. The "T" coordinates begin at 0 and are assigned in sequential order to a value that is equal to the number of processes minus one. For example, if the node is configured to contain four processes, the "T" coordinates range from 0 to 3. The coordinate T = 0 corresponds to the first process containing processor IDs 0 - 15. The coordinate T = 3 corresponds to the last process containing processor IDs 48 - 63. Each unique rank within a job is identified by a corresponding set of A, B, C, D, E, T coordinates.

3.8 Sub-node jobs

The compute node kernel supports sub-block jobs within a node. A sub-node, sub-block job is known as a sub-node job. A sub-node job occupies a subset of the 16 cores available for assignment to applications. Jobs that are running in a subset of the 16 cores in the node can be started and ended asynchronously. Only one core per sub-node job is supported. Only one process per node in a sub-node job is supported. Sub-node jobs are restricted to a single user per node.

3.9 Threading overview

The CNK provides a threading model based on the Native Portable Operating System Interface (POSIX) Thread Library (NPTL) available in the glibc library. The NPTL package is the default threading package for Linux applications. The NPTL threading package implements the POSIX pthread API. The NPTL package allows the same POSIX pthread API library that Linux uses (-lpthreads) to function on the CNK without special parameters.

3.9.1 Hardware thread over-commitment

More than one pthread can be assigned to a given hardware thread. These additional pthreads are supported by additional kernel thread structures (that is, an M:N threading

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

model of 1:1). By default, five pthreads can be assigned to one hardware thread. In the Blue Gene/Q threading model, pthreads have absolute affinity to the hardware threads they are associated with. There is no time-quantum driven preemption of pthreads running on a hardware thread. After a pthread begins to run on a hardware thread, it continues to run until one of the following occurs:

- The thread calls pthread_yield(), and an equal or higher-priority thread available for dispatch is found.
- A signal is being delivered to a higher-priority pthread on the same hardware thread.
- The thread enters a futex wait condition.
- The thread enters a nanosleep system call.
- A new pthread is created on this hardware thread or is migrated to this hardware thread. Its priority is higher than the currently running thread.
- The priority of the running thread is lower or the priority of a thread that is ready to run on the same hardware thread is raised such that a more eligible thread is now available to be dispatched.
- The thread exits.
- A nanosleep previously initiated by a higher-priority pthread on the same hardware thread expires.
- A timed futex wait previously initiated by a higher-priority pthread on the same hardware thread expires.

3.10 Thread scheduler

The kernel scheduler runs on each hardware thread independently. Each local dispatcher handles the dispatching of the software threads assigned to the one hardware thread that it controls. There is no global dispatcher. Therefore, no global locks or blocking conditions are required to manage the dispatching of threads.

3.10.1 Thread preemption

A pthread is preempted when and only when a pthread with a strictly higher software priority is available to be run on the same hardware thread. This scenario can occur for the following reasons:

- A futex-wait by a higher-priority pthread is satisfied.
- A signal is delivered to a higher-priority pthread.
- A new pthread with a higher software priority is created on, or is migrated to, this hardware thread.
- The software priority of the current pthread is lowered, or the priority of another pthread on the same hardware thread is raised.
- A nanosleep initiated by a higher-priority pthread on the same hardware thread expires.
- A timed futex wait initiated by a higher-priority pthread on the same hardware thread expires.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

3.10.2 Thread yield

When a pthread executes a pthread_yield() function and another pthread with the same software priority is available to be dispatched, the current thread relinquishes control. The other thread is dispatched. If there is no other runnable thread of equal or higher-priority, control returns to the thread that executed the pthread_yield() function.

3.10.3 Round-robin dispatch

A thread relinquishes control due to a yield or a futex wait. If there are other pthreads with the same software priority, those pthreads are selected over the current thread for the next dispatch in a round-robin order. In other words, when there are multiple equal-priority pthreads on a hardware thread, and each pthread issues frequent yields, each of the pthreads makes progress. There is no guarantee that each thread will make equal progress. Interrupt conditions presented to the hardware thread might cause unbalanced thread dispatching within the scheduler's simple, light-weight, round-robin algorithm.

3.11 Thread affinity

When a pthread is created within a process, the CNK must select a hardware thread for the pthread. The kernel supports two layout algorithms for assigning pthreads to hardware threads. The number of hardware threads that are available to the process is dependent on the number of processes in the node. See 3.3, “Processes per node” on page 14. The layout types in the following sections can be activated through the use of an environment variable, BG_THREADLAYOUT. If required, additional layout algorithms can be added. When possible, the even-numbered processor IDs within the process are assigned before the odd-numbered processor IDs because of the configuration limitations that are imposed by the hardware universal performance counter implementation.

3.11.1 Breadth-first assignment

Breadth-first is the default thread layout algorithm. This algorithm corresponds to BG_THREADLAYOUT = 1. With breadth-first assignment, the hardware thread-selection algorithm progresses across the cores that are defined within the process before selecting additional threads within a given core.

3.11.2 Depth-first assignment

This algorithm corresponds to BG_THREADLAYOUT = 2. With depth-first assignment, the hardware thread-selection algorithm progresses within each core before moving to another core defined within the process.

3.11.3 Thread affinity control

Controlling the placement of pthreads on the existing hardware threads is supported by the kernel through the sched_setaffinity() system call. The target of the affinity operation must be one and only one hardware thread. The interface to specify the target hardware thread is defined by the glibc structure, cpu_set_t. The CPU numbers to be specified by the caller correspond to the processor IDs 0 - 63. The caller must be aware of the range of valid processor IDs for the current process. For a configuration where there is one process on the node, all processor IDs are owned by the process. However, on a system that has four

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

processes in the node, the first process owns processor IDs 0 - 15. The second process owns processor IDs 16 - 31. Determining what processor IDs are controlled by a given process can be accomplished by using the Kernel_ThreadMask(T) and the Kernel_MyTcoord() SPIs. After the T coordinate is obtained using the Kernel_MyTcoord system programming interface (SPI), supply it to the Kernel_ThreadMask(T) SPI. The SPI returns a 64-bit mask representing the processor IDs owned by the currently running process.

There are two methods to set the affinity of a pthread. The first method is at pthread creation time through the pthread attributes structure. The second method is explicitly through the set_affinity system call.

3.11.4 Setting affinity with the pthread attribute

Example 3-1 shows how to set affinity with the pthread attributes at pthread creation.

Example 3-1 Setting affinity through the pthread attributes

```
pthread_attr_t attr; // create an attribute object
cpu_set_t cpumask; // create a cpu mask object
pthread_attr_init(&attr); // initialize an attribute object
CPU_ZERO(&cpumask); // initialize the cpu mask
CPU_SET(processorID, &cpumask);
pthread_attr_setaffinity(&attr, CPU_SETSIZE, &cpumask);
rc = pthread_create(&thread[t], &attr, myThreadFunction, NULL);
```

3.11.5 Setting affinity with the system call

The following code example shows how to set explicit affinity with the system call.

Example 3-2 Setting affinity using the system call

```
cpu_set_t mask;
CPU_ZERO( &mask ); /* CPU_SET sets only the bit corresponding to cpu. */
CPU_SET( processorID, &mask ); /* pthread_setaffinity returns 0 in success */
if( pthread_setaffinity_np( tid, sizeof(mask), &mask ) == -1 )
{
printf("WARNING: Could not set CPU Affinity, continuing...\n");
}
```

3.11.6 Extended thread affinity control

Extended thread affinity control is a facility that allows a process to place, using set affinity, software threads on hardware threads that were not originally allocated to that process. This feature is useful in application environments where an application might enter different phases of execution that require a larger number of threads to be used by a subset of the processes in a node while other processes in the node are not actively using their threads.

Enablement

An environment variable is used to enable the extended thread affinity control facility. If the BG_THREADMODEL environment variable is set to the value 2, set affinity APIs can be used to place a pthread onto a hardware thread that is not configured as a hardware thread owned by the current process. See Table D-5 on page 142 for more information about the BG_THREADMODEL variable.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

For example, if the application must transition between 16 active processes, each using four hardware threads, to four active processes, each using 16 hardware threads, the application is started with 16 processes configured. Each process creates its pthreads using the pthread_create() function. Each of the four processes can use the pthread_create() function to create up to a total of 16 threads and use the setaffinity API interfaces to place these pthreads on hardware threads outside its configured set of hardware threads. When the application reaches the end of its first phase, 12 of the 16 processes block using a standard POSIX synchronization mechanism, such as a shared mutex, condition, or barrier. Then the four remaining processes begin running their additional pthreads using the additional hardware threads that were previously used by the now blocked 12 processes. When this phase of the application completes, the processes are unblocked and the application returns to its original behavior of having 16 active processes each using four hardware threads.

Usage restrictions

The following restrictions apply to the extended thread affinity control facility:

- The job must be configured with 2, 4, 8, or 16 ranks per node.
- A core can host the originally configured process plus 1, 2, 3, or 4 additional threads of any one additional process within the node.
- MPI operations are not supported for pthreads that are executing on a hardware thread that was not originally configured to a pthread's process.
- Memory allocation across the processes in the node is based exclusively on the initial memory configuration at job start time.
- Transactional memory and thread level speculation operations are not supported on pthreads that are executing on a hardware thread not originally configured to a pthread's process.
- Setting and handling of the itimer is not supported for pthreads that are executing on a hardware thread not originally configured to a pthread's process.
- Performance monitoring (BGPM) is not supported for pthreads that are executing on a hardware thread not originally configured to a pthread's process.
- The user's code cannot perform a setaffinity to the process leader's thread ID (that is, main thread).

Controlling Application Phases

The application can use any of the following CNK supported interprocess synchronization mechanisms to transition into and out of actively running pthreads that are executing on a hardware thread that is not configured to a pthread's process:

- Barriers using pthread_barrier with shared attribute set
- Conditions using pthread_cond with shared attribute set
- Mutexes using pthread_mutex with shared attribute set

The application can also use its own synchronization mechanisms as long as the blocked threads are not waiting on the completion of a function shipping system call.

3.12 Thread priority

The software thread priority can be set within a pthread through either the pthread attribute structure or through an explicit system call. The priority values that are supported depend on the scheduling policy that is set for the pthread. Thread priorities are evaluated in the scheduler when a condition occurs within a hardware thread that causes the scheduler to

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

select a potentially different pthread for dispatching. Because the Blue Gene/Q system has absolute hardware thread affinity, the relative pthread priorities of pthreads on different hardware threads has little consequence. The relative thread priorities are important for pthreads assigned to the same hardware thread.

There are conditions in which a communication thread might require control only when no other application threads are running. Because of this requirement, communication threads can specify a priority that is lower than any application thread. Conversely, there are situations when a communication thread might need to be the highest priority software thread on the hardware thread. Therefore, communication threads are allowed to set a priority value that is more favored than any application thread priority. This widened range of priorities is supported by the use of a special scheduling policy, SCHED_COMM. See 3.12.1, “Setting priority through the pthread attribute” on page 22.

Thread priority can be modified dynamically, for example, a pthread might want to raise or lower its priority before relinquishing control. A priority change results in a call to the kernel within the target thread. At that time, the relative priorities of the software threads on the hardware thread are re-evaluated. The most eligible software thread is dispatched.

3.12.1 Setting priority through the pthread attribute

Priority can be set through the pthread attribute structure supplied to pthread_create. The following API must first be issued to have the priority information in the attribute used. The following code sets the inherit attribute:

```
pthread_attr_setinheritsched(&attr, PTHREAD_EXPLICIT_SCHED);
```

Setting the priority within the pthread attribute can be done by either specifying just the priority or by specifying both the policy and priority. To specify both the policy and priority, see 3.12, “Thread priority” on page 21. To specify just the priority, use the following API:

```
pthread_attr_setschedparam(pthread_attr_t *attr, struc sched_param *param)
```

If this API is used, the policy is inherited from the caller if the pthread_attr_setschedpolicy() was not called, even though the PTHREAD_EXPLICIT_SCHED value is specified on the pthread_attr_setinheritsched() API is specified.

To determine the priority range for a given scheduler policy, use the following APIs:

- sched_get_priority_max(int Policy)
- sched_get_priority_min(int Policy)

Table 3-3 outlines the priorities for the Blue Gene/Q system. However, these ranges are subject to change. The application code must avoid making assumptions regarding the valid priority range for a given policy and use the previously mentioned APIs.

Table 3-3 Blue Gene/Q priorities

Policy	Minimum priority	Maximum priority
SCHED_OTHER	2	98
SCHED_FIFO	2	98
SCHED_COMM	1	99

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

3.12.2 Explicit setting of priority

Priority can be set explicitly through the use of the pthread_setschedparam() API. Example 3-2 on page 20 shows an example of explicitly setting affinity.

Example 3-3 Setting priorities explicitly

```
#include <sched.h>
int pthread_setschedparam(pthread_t thread, int policy, const struct sched_param
*param);
```

3.12.3 Hardware thread priority

The hardware thread priority represents the relative proportion of available core cycles that are to be given to hardware threads that share the same core. There are seven PowerPC architected hardware thread priorities. However, the Blue Gene/Q system internally implements two priority levels. Based on internal configuration settings, the mapping between the architected and available priority levels are shown in Table 3-4.

Table 3-4 Priority level mapping

PowerPC architected priority level	PowerPC instruction	PPR32::PRI	Implemented Blue Gene/Q priority level
low	or 1, 1, 1	0b010	low
medium-low	or 6, 6, 6	0b011	medium
medium	or 2, 2, 2	0b100	medium

Setting hardware thread priority from an application

Applications can use the low and medium Blue Gene/Q hardware thread priorities. The following statement brings in the required header file that contains the inline interfaces:

```
#include <hwi/include/bqc/A2inlines.h>
```

The interfaces control hardware thread priorities.

Important: These priority terms refer to the architected PowerPC priority level terminology, not the Blue Gene/Q priority terminology.

The following inlines can be used to set hardware thread priorities:

- void ThreadPriorityMedium();
- void ThreadPriorityMediumLow();
- void ThreadPriorityLow();

The ThreadPriority_Medium() and ThreadPriority_MediumLow() interfaces both map to the medium Blue Gene/Q priority level. The ThreadPriority_Low() sets the low Blue Gene/Q priority level.

The current priority of the hardware thread can be obtained by reading the PPR32 register. Table 3-4 describes the mapping of the priority levels. The thread priority can also be set by writing to this register. This approach can be useful when restoring a previously saved priority after priority modification. Example 3-4 on page 24 demonstrates a sequence of saving, modifying, and restoring the hardware priority of a thread.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Example 3-4 Sequence of saving, modifying, and restoring the hardware priority of a thread

```
// Save current hardware priority
uint64_t ppc32 = mfspr(SPRN_PPR32);

// Force hardware priority to low
ThreadPriority_Low();

// perform some function ...

// restore priority
mtspr(SPRN_PPR32, ppc32);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

4

Memory

This chapter provides an overview of the memory subsystem and explains how it relates to the Compute Node Kernel (CNK). This chapter includes the following topics:

- Memory system overview
- Memory management
- Memory protection
- Shared memory
- Persistent memory
- Compute node ramdisk
- Support for the /proc file system
- L1P prefetcher
- L2 atomic operations
- Speculative execution
- Support for dynamic linking
- Transactional memory

© Copyright IBM Corp. 2013. All rights reserved.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

4.1 Memory system overview

The Blue Gene/Q system contains a distributed memory system, which includes an on-chip cache hierarchy and an off-chip main store. It contains optimized on-chip symmetric multiprocessing (SMP) support for locking and communication between the 17 ASIC processors. Each processor can have four threads.

The aggregate memory of the total system is distributed in the style of a multicomputer, with no hardware sharing between nodes. Each node contains 16 GB of physical main memory. This memory is non stacked synchronous dynamic random access memory (SDRAM).

The first-level (L1) caches are contained in the A2 core macro. The L1P cache is used as a prefetch cache and write-back buffers for L1 data. The second-level (L2) cache is 32 MB.

Table 4-1 on page 27 lists the memory specifications for the Blue Gene/Q system.

4.1.1 L1 prefetch cache overview

The level 1 prefetch (L1p) cache is a module that provides the interface between an A2 core and the rest of the Blue Gene/Q system. It interfaces to the Blue Gene/Q switch, the device control register (DCR) device ring, a memory-mapped I/O space that is local to the core, and a static random-access memory (SRAM) module that might be local to the core. The L1p cache manages a 32 $\times$ 128 byte cache structure to identify and prefetch memory access patterns. This functionality is critical for performance. The L1p cache also performs write combining. It presents multiple small writes to the switch as a single write, while maintaining data coherency.

The L1P cache has the following functions:

- Provides A2 interfaces to the Blue Gene/Q system:
  - Request
  - Store
  - Reload
  - Synchronize
  - Reservation
  - Invalidation
- DCR bridge visible as memory mapped I/O
- Prefetching:
  - Stream prefetch engine with automatically detected and software-hinted streams
  - List prefetch engine
  - Optional symmetrical treatment of information and data prefetch
- Write combining support
- Synchronization support using generation protocol
- Pipelined switch interface:
  - Out-of-order interface to distinct destinations
  - In-order interface to a single destination

L1P instruction support for Blue Gene/Q compute chips

Table 4-1 on page 27 lists the memory specifications for the Blue Gene/Q system.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Table 4-1 Blue Gene/Q memory specifications

Cache	Quantity[a]	Size	Latency[b]	Replacement policy	Other information	Clock domain
L1 instruction cache	18 (1 per processor)	16 KB	3 processor clocks (pclk)	Pseudo least recently used (LRU)	4-way set-associative 64-byte line size	Pclk
L1 data cache	18 (1 per processor)	16 KB	6 pclk (integer)	Pseudo LRU	8-way set-associative 64-byte line size	Pclk
L1 prefetch cache	18 (1 per processor)	32 × 128 bytes	24 pclk	Depth stealing and round robin	128-byte line	Pclk / 2
L2 cache	16	2 MB/slice 32 MB total L2 cache on-chip	82 pclk	LRU	16-way set-associative 16-way sliced 4 banks per slice 8 sub-banks per slice 128-byte line	Pclk / 2
Double-data rate (DDR) memory	2	16 GB total	≥ 350 pclk		128-byte line	Pclk × (5 / 6)
Embedded dynamic random-access memory (eDRAM)	1	256 KB	≥ 80 pclk	Software control	16 bytes wide 8 eDRAM macro-internal bank	Pclk / 2

a. This value is the quantity on each Blue Gene/Q compute chip.
b. The latency value is determined relative to instruction dispatch.

Prefetch algorithms

Two prefetch algorithms are supported, linear streams and list streams.

Linear Streams

Up to 16 concurrent linear streams of consecutive addresses can be simultaneously prefetched. Linear streams can be automatically identified, or hinted, using data cache block touch (**`dcbt`**) instructions, or established optimistically for any miss. Stream underflow (a hit on a line that is being fetched from the switch) triggers a depth increase when adaptation is enabled. Stream replacement and depth stealing lines are selected with a least recently hit algorithm.

List Streams

With software cooperation, access patterns can be recorded and reused by a list fetch engine. This implementation allows iterative application software to make efficient use of completely general, but repetitive, access patterns. The recording of patterns of physical memory access by hardware enables virtual memory issues to be ignored.

A2 interface

The L1p cache accepts commands and data from the A2 core at the pclk period. Received commands are queued in a 32-deep lookup queue. This depth of 32 supports eight

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

outstanding data load requests, four instruction load requests, and 20 store requests. The A2 core supports a maximum of eight outstanding data load requests and four instruction load requests. At reset, the A2 core is programmed to issue no more than 20 outstanding store commands. The number 20 corresponds to the maximum of 16 requests that can be accepted by the switch and an additional four active store commands, not committed to the switch, which the L1p cache can maintain for write combining. The elements of this queue include pointers to a request array that contains the address associated with that command. For store operations, this queue also includes a pointer to the location in the 20-entry store buffer that contains the store data.

4.1.2 L2 cache functional overview

The L2 cache units provide most of the memory system caching on the Blue Gene/Q compute chip. There are 16 individual caches, or *slices*. Each cache is assigned to store a unique subset of the physical memory lines. The physical memory addresses that are assigned to each cache slice are static and configurable. The L2 line size is 128 bytes, which is twice the width of an L1 line. L2 slices are set-associative and organized as 1024 sets. Each set has 16-way association. The L2 data store comprises embedded DRAM, and the tag store comprises SRAM. The main memory is accessed through two on-chip DDR controllers. Each controller manages eight L2 slices. The primary logic of the L2 caches operates at half the processor clock frequency. Some interface logic operates at lower frequencies. Each L2 slice has a single read data port that is 256 bits wide, a single write data port that is 256 bits wide, and a single request port. This port is shared by all processors through the crossbar switch.

The L2 caches primarily operate as normal, set-associative caches. They also support speculative threads and atomic memory transactions.

The L2 caches serve as the point of coherence for all processors. Therefore, they generate L1 invalidations when required. Because the L2 caches are inclusive of the L1 caches, they can remember which processors might have a valid copy of every line. They can multicast invalidations to only those processors. The L2 caches are also a synchronization point, so they coordinate synchronization (**`msync`**), load and reserve (**`lwarx`**), and store conditional (**`stwcx`**) instructions.

4.1.3 Boot eDRAM overview

The Blue Gene/Q system uses a boot eDRAM macro. The eDRAM macro has the following properties:

- 256 KB capacity
- 16-byte-wide access
- 1.25 ns cycle time, 5 ns latency
- 4-way banked, fully pipelined for high throughput

The module is directly operational after reset and provides the boot code. It is also used as a background communication path to the host system. The boot eDRAM macro is connected to the A2 cores with the device bus, which is directly connected to the cores with the crossbar switch. Joint Test Action Group (JTAG) access is managed by the JTAG controller, which is also connected to the device bus.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

4.2 Memory management

For optimal performance, manage memory carefully on the Blue Gene/Q system. The memory subsystem of Blue Gene/Q nodes has specific characteristics and limitations. Although a Blue Gene/Q node has 16 GB of memory, physical memory size constraints must still be considered when writing, running, and debugging applications.

The CNK does not dynamically grow its memory usage over time. The CNK consumes a fixed size of 16 MB out of the 16 GB of memory. Therefore, additional threads, mmaps, system calls, buffers, and so on, do not change internal kernel memory usage. The remainder of the memory (16,368 MB on a 16 GB node) is partitioned for the application.

When the application is started, the CNK examines the following information:

- Virtual addresses, sizes, and permissions for all application sections
- Size of memory to parcel
- Requested size of the shared memory segment
- Persistent memory size and its present physical address
- The number of processes to create
- Whether an interpreter is required (an interpreter is commonly required by dynamically linked executable programs)

The CNK partitions memory to form a static memory map. This static memory map is a translation between virtual addresses (as seen by the application) into physical addresses in the DDR3 memory. This partitioning process is designed to generate a valid mapping that maximizes memory use.

4.3 Memory protection

The CNK has several mechanisms that provide protection against incorrect memory accesses:

- All storage used by the kernel is inaccessible to user applications.
- The text segment of a statically linked application is write protected.
- The text segment of a dynamically linked application is write protected.
- The nonshared address space of processes on the node is not directly accessible by other processes on the node.
- Guard pages can be activated if the compiler does not insert speculative **`dcbt`** instructions.

4.4 Shared memory

The CNK supports shared memory between all the processes on a given node. The size of shared memory must be specified to the **`runjob`** command with an environment variable. Shared memory is supported in all process counts. However, shared memory with one process per node is not necessary because each processor already has access to all of the node memory.

Shared memory is allocated with the standard Linux shm_open() and mmap() methods. The CNK does not have dynamic virtual pages. Therefore, the physical memory that backs the

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

shared memory must come out of a memory region that is dedicated for shared memory. The size of this memory region is set when a job is started.

The BG_SHAREDMEMSIZE environment variable specifies the amount of memory to be allocated in MB. Use the **runjob** --envs flag. For example, `BG_SHAREDMEMSIZE = 32` allocates 32 MB of shared memory storage. For more information about environment variables, see “Compute Node Kernel environment variables” on page 142.

The amount of memory to be set aside for this memory region can be changed at job launch.

Example 4-1 illustrates shared-memory *allocation*.

Example 4-1 Shared memory allocation

```
fd = shm_open( SHM_FILE, O_RDWR, 0600 );
ftruncate( fd, MAX_SHARED_SIZE );
shmptr1 = mmap( NULL, MAX_SHARED_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
```

Example 4-2 illustrates shared-memory *deallocation*.

Example 4-2 Shared memory deallocation

```
munmap(shmptr1, MAX_SHARED_SIZE);
close(fd)
shm_unlink(SHM_FILE);
```

The shm_open() and shm_unlink() routines access a pseudo-device, `/dev/shm/filename`, which the kernel interprets. Because multiple processes can access or close the shared-memory file, allocation and deallocation are tracked by a simple reference count. Therefore, the processes are not required to coordinate deallocation of the shared memory region.

The value of BG_SHAREDMEMSIZE can be queried by the application using the Kernel_GetMemorySize(KERNEL_MEMSIZE_SHARED, &shared_size) SPI call.

4.5 Persistent memory

Persistent memory is process memory that retains its contents from job to job. To allocate persistent memory, the environment variable BG_PERSISTMEMSIZE = X must be specified. “X” represents the number of megabytes to be allocated for use as persistent memory. For the persistent memory to be maintained across jobs, all job submissions must specify the same value for the BG_PERSISTMEMSIZE variable. The contents of persistent memory can be reinitialized during job startup either by changing the value of BG_PERSISTMEMSIZE or by specifying the environment variable BG_PERSISTMEMRESET = 1. The persist_open() kernel function supports persistent memory.

4.6 Compute node ramdisk

The CNK provides a random-access file system that is resident in compute node memory. This file system is local to the compute node. File system operations to the ramdisk do not result in I/O activity to the I/O node or other compute nodes.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

There are three mount points for the compute ramdisk. Table 4-2 on page 31 shows the mount points.

Table 4-2 Mount points for the compute node ramdisk

Name	Mount point	Size	Scope
Shared memory	`/dev/shm/`	Size is determined by the BG_SHAREDMEMSIZE environment variable. MPICH and PAMI use some of this shared memory.	Node-wide, cleared when job exits.
Persistent memory	`/dev/persist/`	Size is determined by the BG_PERSISTMEMSIZE environment variable.	Node-wide, cleared only when BG_PERSISTMEMSIZE is specified differently or if BG_PERSISTMEMRESET is set.
Local memory	`/dev/local`	Size is determined by the Kernel_SetLocalFSWindow() SPI call.	Process-wide, cleared when job exits.

The CNK supports a range of system calls for the ramdisk, including the following calls: read, write, open, close, lseek, utime, rename, dup, dup2, mmap, munmap, truncate, ftruncate, stat, lstat, fstat, fsync, llseek, readv, writev, truncate64, ftruncate64, stat64, lstat64, and fstat64.

This support has the following limitations:

- File access permissions and ownership are not tracked or honored. The **`chmod`** and **`chown`** functions do not have an effect. However, the access() system call can be used to determine file existence.
- File directories are not modeled. So, the mkdir() and rmdir() system calls have no effect. Instead, the open() system call honors separate file namespaces that are specified with directory prefixes. (That is, the slash '/' character is treated as part of the file name, not as a separator of the directory hierarchy.)
- The flock() system call handles only the LOCK_EX and LOCK_UN opcodes.
- Advisory file access hints using the posix_fadvisory() calls are ignored.
- The unlink() system call cannot reclaim space for unlinked, memory-mapped files in the CN RAM disk.

4.7 Support for the /proc file system

The CNK creates several files that can be used to obtain process-specific data. These files are in the `/proc/<pid>` directory. Table 4-3 lists the files.

Table 4-3 Files in the /proc/<pid> directory

File	Description
/proc/<pid>/exe	A symbolic link to the executable.
/proc/<pid>/cwd	A symbolic link to the current working directory.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

File	Description
/proc/<pid>/maps	A regular file that represents the memory map for the process. The memory map includes text, data, heap, stack, and dynamic library address ranges for the process.
/proc/<pid>/cmdline	A regular file that contains the command line passed into the process.
/proc/<pid>/environ	A regular file that contains the environment variables for the process at job start.

The <pid> is the value that is returned by the getpid() function. For example, if the getpid() function returns 17, the file name is "/proc/17/maps". Alternatively, the `/proc/self/<filename>` syntax can be used.

Regular files can be accessed through the Blue Gene/Q Code Development and Tools Interface (CDTI) Get File Names, Get File Stat Data, and Get File Contents commands. For more information, see the *IBM System Blue Gene Solution: Blue Gene/Q Code Development and Tools Interface*, REDP-4659 Redpapers publication.

4.8 L1P prefetcher

The Blue Gene/Q hardware has two prefetch engines that can be manipulated by the user to control cache prefetch behaviors.

The L1 prefetcher (L1p) is a module in the Blue Gene/Q compute chip that is interposed between the A2 bus and the rest of the Blue Gene/Q chip. There are 17 copies of the L1p. Each copy is attached to one of the 17 Blue Gene/Q A2 cores. Figure 4-1 shows the L1p module.

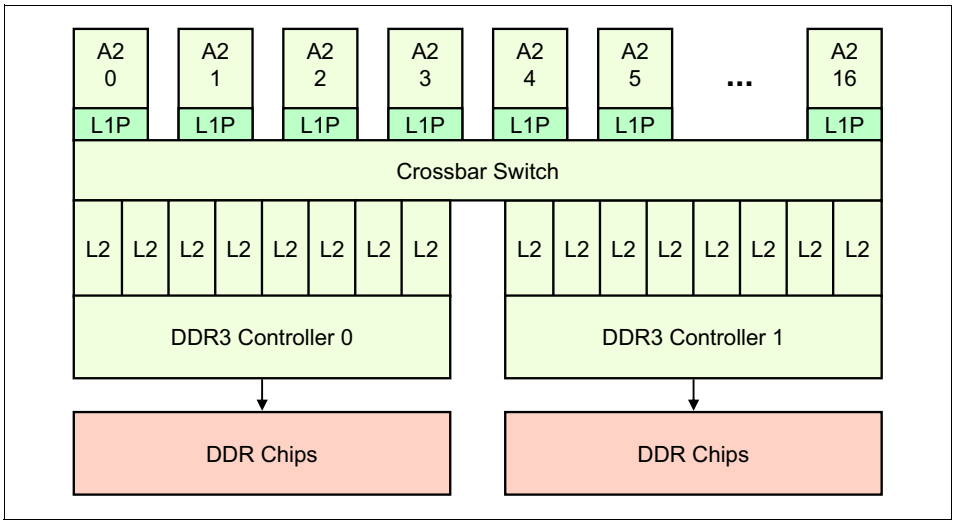


Figure 4-1 L1p module

The L1p performs the performance-critical tasks of identifying and prefetching memory access patterns and managing a 32 x 128-byte cache structure for prefetch data. The prefetcher is designed to predict which pieces of data will be required by the processor. The L1 prefetcher in the Blue Gene/Q system provides two prefetch algorithms: a linear stream prefetcher and a perfect prefetcher.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

4.8.1 Linear stream prefetcher overview

The linear stream prefetcher that is used in the Blue Gene/Q system detects positive sequential memory access strides and prefetches ahead of the stream when possible. It can track up to 16 streams of memory accesses. Each stream is sequential and a stride of one L1 cache line (64 bytes).

Figure 4-2 on page 33 describes two streams of data that are required by the processor. The streams are at different addresses, but the linear stream prefetcher can track them independently. The goal is that when the processor needs address 0x100080 or 0x175080, the L1p has prefetched that data into its prefetch cache, reducing memory access latency.

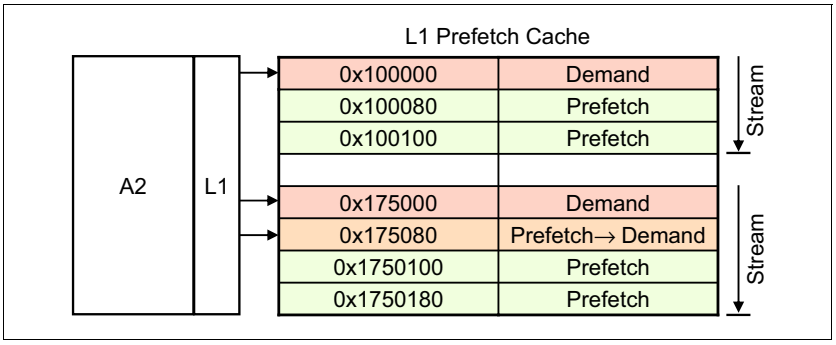


Figure 4-2 Linear stream prefetcher

The linear stream prefetcher is the default prefetch algorithm and is always active.

Establishing a stream

The L1p only prefetches from established streams. The stream detection logic has several different modes that are settable by the application to control how streams are established.

Optimistic mode Assumes that all L1 misses will become streams. In this mode, the L1p immediately starts prefetching from the next cache line in the stream.

Confirmed mode The linear stream prefetcher will wait for at least one additional L1 miss that corresponds to the stream to be detected. After confirmation, the L1p starts prefetching from the stream.

Confirmed or cache touch mode The prefetcher behaves like confirmed mode. Additionally, a stream will be established if an explicit **`dcbt`** (data cache block touch) instruction that results in an L1 cache miss is executed.

Adaptive prefetching

When a stream is established, it is internally assigned a stream number. This stream number is used to track the depth of the prefetch for that stream. It is possible that some streams advance quicker than other streams. Therefore, certain streams should prefetch further ahead than other streams.

With adaptive prefetching, when the L1p detects that an established stream encounters an L1p miss that is not already in the prefetch cache, it automatically increases that stream number's prefetch depth by one L1p 128-byte cache line (up to the adaptive prefetch depth limit). The maximum and initial prefetch depths for adaptive streams are configurable between

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

1 and 8. The rate of adaptation might be configurably throttled to performance tune the adaptive prefetch.

Thread synchronization

Each core in a Blue Gene/Q system has a single dedicated linear stream prefetcher. However, all four hardware threads share the same linear stream prefetcher. This sharing can cause an atomicity and ordering problem if multiple threads modify the linear stream prefetcher's configuration registers. The L1p and the SPI do not restrict or block access to the configuration registers. If the application developer is potentially modifying the configuration registers from multiple hardware threads, locking must be added or there might be some non-deterministic choices in the L1p configuration.

The following strategies can be used for locking:

pthread_mutexes Standard POSIX locking primitives. These primitives are simple and work well in threaded processes.

larx/stcx PowerPC load reservation locking mechanism. This mechanism can be used to create a lighter-weight lock than pthread_mutex. The **`larx/stcx`** instructions can also be used for multiple processes that have a shared memory region.

L2 atomics Using the Blue Gene/Q L2 atomic operations to "take a ticket" with load and increment. The thread blocks until the "now serving" counter matches the ticket. The thread then updates the configuration register and performs a store and add operation to add 1 to the "now serving" counter.

L2 transactional memory regions cannot be used for atomically setting L1p configuration registers because the L2 does not version the L1p MMIO memory region.

4.8.2 Perfect prefetcher overview

The perfect prefetcher in the Blue Gene/Q system uses a recorded pattern of memory accesses to effectively prefetch data into the caches. Unlike the linear stream prefetcher, the perfect prefetch algorithm requires that the application train the perfect prefetcher with specific patterns of memory accesses. When the application executes the same section of code again, the application must inform the L1p hardware that the previously recorded pattern will be reoccurring. As the hardware thread is executing this section of code, the L1p hardware is tracking the progress of the pattern and attempting to prefetch ahead of the anticipated data. Since the recorded pattern and the next pattern through a section of code might not be precisely the same, the L1p has some tolerance for pattern deviations.

There are four perfect prefetchers per L1p. Each perfect prefetcher is assigned to a separate hardware thread in the associated A2 core. This allows each hardware thread to be creating and executing a separate list of prefetches with no requirement for software coordination.

The headers and documentation for the L1p system programming interface (SPI) are provided in the `spi/include/l1p` directory.

Training the perfect prefetcher

To train the perfect prefetcher, the application configures the L1p with the L1P_PatternConfigure(size) L1p SPI call. The size parameter contains the maximum number of L1 misses that are expected through the code sequence. This is used for calculating the memory buffer space that is needed to hold the patterns. There is no enforced limit (outside of the size of available memory) for the maximum size of the pattern. If the buffer space is

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

exceeded by the pattern, pattern recording is halted and the "Maximum" bit in the L1P_Status_t structure indicates this overflow condition.

Next, the application calls L1P_PatternStart() with the record flag set and then executes a section of code. Meanwhile, the L1p hardware is recording each L1 cache miss in the application-provided storage. When the application has completed the code section, it tells the L1p to stop recording by means of another L1p SPI call, L1P_PatternStop().

This prefetch algorithm works when there is a consistent L1 cache miss pattern. For applications that momentarily deviate from consistency, the L1p can disable or pause the prefetcher (both training and prefetching). This process prevents the prefetcher from recording L1 cache misses that are not likely to repeat during the next execution of the recorded code.

Using a trained pattern

When a list of L1 cache misses has been trained, the application calls the L1P_PatternStop() function. This stops training and sets up the trained pattern to be used for prefetching on the next iteration.

To start prefetching with this new pattern, the application then issues a L1P_PatternStart() call. This call tells the hardware thread's L1 perfect prefetcher to start loading the list into its cache. It then monitors the loaded section of the list, and tracks the L1 cache misses with the list.

In this call to the L1P_PatternStart() function, the record flag can optionally be set. When set (in self-healing mode), the L1p creates a new revision of the list in a separate physical memory location for later use. The L1P_PatternStart() function then implicitly toggles between the two lists (current and next list) by swapping the physical addresses for the L1p perfect prefetcher's read/write base addresses.

To synchronize the prefetching of the data pattern with the A2 execution, the perfect prefetcher must track where the application is executing with respect to the prefetch list. Since the application is not required to be perfectly reproducible with regards to L1 cache misses, some tolerance is provided for the appearance of L1 miss addresses that are not present in the prerecorded pattern and for addresses that are recorded in the pattern which are missing from the actual stream of L1 cache misses. Figure 4-3 shows an L1p cache miss.

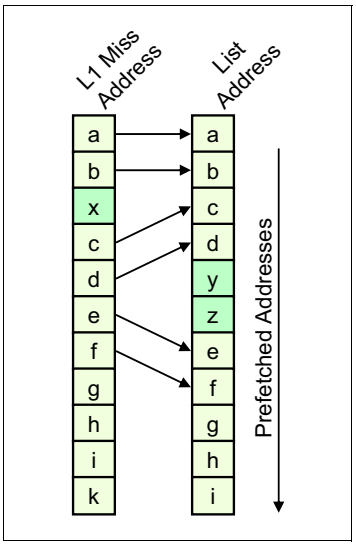


Figure 4-3 L1p cache miss

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

In Figure 4-3, the address at location 'x' was not forecast in the list. However, the next L1 miss at location 'c' was expected. The perfect prefetcher ignores the rogue address 'x' and continues matching at location 'c'. Similarly, locations 'y' and 'z' were in the list, but were not presented to the L1p as L1 cache misses. Again, the L1p was able to look ahead in the pattern and adjust its list address offset to correspond to location 'e'.

Since the L1p cannot have a multi-megabyte metadata cache containing the full list, it fetches as many as 24 entries of the list to identify data to be prefetched and to synchronize with the stream of L1 misses. The pattern prefetching hardware is able to compare the current L1 miss address with the next, not-yet-matched list entry and up to 7 subsequent addresses in the prerecorded pattern. If an L1 miss is not present in this group of up to 8 pattern addresses available for comparison, the perfect prefetcher drops that L1 miss address but tracks the number of such consecutive non-list misses. If the number of consecutive misses exceeds a predefined miss threshold, the perfect prefetcher abandons the list and halts prefetching. This behavior is achieved by providing a 24-location buffer inside each perfect prefetcher into which the prerecorded pattern of addresses is automatically loaded from memory. These 24 locations are divided into two groups. The first group contains the first 8 addresses in the pattern presently being prefetched and is instrumented to permit the 8 comparisons. The remaining 16 locations are in standard SRAM and provide a buffering function, increasing the likelihood that the prerecorded addresses are available in the perfect prefetcher when needed.

Thus, there are two scenarios that can lead to list abandonment:

- The thread started executing code with memory access patterns that contain a sequence of addresses that is unrelated to the recorded memory patterns of length greater than the preset threshold. The preset threshold can be set with the L1P_PatternSetAbandonThreshold() function.
- The thread's memory access pattern jumped ahead in the list by more than 8 entries (and therefore the L1p lost synchronization).

The perfect prefetcher status bits, returned by the L1P_PatternStatus() function, can be used to determine if the prefetch abandoned or completed the list.

A recorded pattern can be used at any time, but only 1 prefetch list can be active per hardware thread at any time.

Saving and restoring patterns

The L1p perfect prefetcher SPI only manages one pattern at a time. However, for greater flexibility, the L1P_GetPattern() function can be used to retrieve the active pattern from the SPI. This allows pattern storage allocated by the L1P_PatternConfigure() function to be detached and set aside for later usage without destroying the pattern or requiring regeneration of the pattern.

Later on, the application developer can restore the old pattern using the L1P_SetPattern() function. This is more efficient because it does not require a reallocation and regeneration of the pattern. After restoring the pattern, the application calls the L1P_PatternStart() function to begin executing the pattern.

The number of patterns that can be retained by the application is limited only by the amount of memory available to hold patterns.

When a pattern has been retrieved using the L1P_GetPattern() function, the application must manage the storage for the pattern. When the pattern is no longer needed, the application should call the L1P_DeallocatePattern() function to release the storage.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Interaction with the linear stream prefetcher

Each L1p comprises a single copy of the linear stream prefetcher and four copies of the perfect prefetcher. Although the algorithms are separate, they share a significant amount of the internal L1p arrays.

The prefetch data array contains all of the demand-loaded and prefetched data, regardless of which prefetch algorithm fetched the data. Therefore, this array is shared between all five prefetchers (one linear stream + four perfect prefetchers). The L1P_SetStreamTotalDepth() routine can be used to limit the linear stream prefetcher's total depth. This reserves a section of the prefetch data array specifically for the perfect prefetcher.

Nested patterns

If a segment of code that is using the perfect prefetcher starts executing a subroutine or library that is not well understood, the application programmer must consider the following scenarios:

1. If the target subroutine has a fairly predictable access pattern and contains no perfect prefetcher use, the subroutine call can also contribute to the pattern that is being recorded by the calling routine. No surrounding perfect prefetcher directives are required.
2. If the subroutine has some random access or data-dependent elements, it might be helpful for the application programmer to issue an L1p_PatternPause() command before the subroutine call and an L1p_PatternResume() command when that subroutine returns. Pausing the prefetcher might reduce the chance of pattern abandonment. When the subroutine is understood, reevaluate the usage of pause.
3. If the subroutine or library also uses the perfect prefetcher and makes its own L1P_Configure() calls, there are controls over the behavior of the L1p SPI when there are nested L1P_Configure() calls.

   If a L1P_Configure() call occurs while a perfect prefetcher is active, the default action is to perform a context switch of the calling hardware thread's L1p perfect prefetcher hardware state. The addresses from which the pattern is being read and to which the new pattern is being written are stored to memory with the content of the prefetcher's configuration registers. When the matched L1P_Unconfigure() function is executed, the perfect prefetcher context is restored and its normal prefetching continues. The pattern that is produced during such a sequence can be successfully used to prefetch data during a later execution of the same code. Any number of such context switches can be nested, supporting a normal, modular programming environment. From a performance perspective, these context switches might require a system call, which might reduce performance.

   If this default L1P_Configure() function behavior is not preferred, the application programmer can call the L1P_PatternSetNestingMode() function. This routine allows the application to manage the L1P SPI behavior when a nested routine is being called. There are four modes:

   – Save/Restore context is always performed when encountering L1P_Configure/Unconfigure() function calls. This is the default mode.
   – Nested L1P_Configure() routines disable the ability to change the active pattern (that is, the current pattern stays active).
   – Nested L1P_Configure() and L1P_SetPattern() routines are disabled. However, other L1P SPI routines that do not switch the active pattern continue to function.
   – Any nested L1P_Configure() routines can result in a fatal error. This can be used for debugging.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Performance counter support

The perfect prefetcher makes full use of the performance counters in the Blue Gene/Q chip. Thus, software can monitor the number of:

- Pattern write overflows
- Times the pattern was abandoned
- Pattern starts
- Times the core was stalled waiting for the pattern to be read
- Times the core was stalled waiting for the pattern to be written
- Times the core address does not match any pattern address
- Times an address in the pattern is skipped over
- Times the pattern comparison catches up with the pattern prefetching
- Comparisons with the pattern

These performance counters can be obtained through the Blue Gene/Q performance counter library and related tools.

4.8.3 L1P prefetcher API descriptions

This section describes the L1P prefetcher API. It contains the following information:

- Defines and enumerations
- Data types
- L1P perfect prefetcher configuration functions
- L1p perfect prefetcher control functions
- Explicit pattern management functions
- L1P linear stream prefetcher control functions
- L1p error conditions

Defines and enumerations

Table 4-4 describes the L1P_StreamPolicy_t enumeration.

Table 4-4 enum L1P_StreamPolicy_t

Name	Description
L1P_stream_optimistic	Any L1 cache miss memory reference (optimistically) establishes a stream.
L1P_stream_confirmed	The L1p waits for confirmation before establishing the stream.
L1P_stream_confirmed_or_dcbt	Any L1 cache miss memory reference using a **dcbt** (data cache block touch) instruction automatically creates an established stream. Otherwise, the L1p waits for confirmation before establishing the stream.

Table 4-5 describes the L1P_PatternNest_t enumeration.

Table 4-5 enum L1P_PatternNest_t

Name	Description
L1P_NestingSaveContext (default)	Any nested L1P_Configure() routines result in an implicit context save. The matched L1P_Unconfigure() routine restores the L1P context.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Name	Description
L1P_NestingIgnore	The L1P pattern routines are disabled for the thread after a nested L1P_Configure() routine is performed. The pattern routines are re-enabled when the matching L1P_Unconfigure() routine is performed.
L1P_NestingFlat	When the application makes a nested L1P_Configure() call, it is reference counted and ignored. Any L1P_SetPattern() calls are ignored if they occur in a nested context.
L1P_NestingError	Nested L1P_Configure() routines cause an error message and assert a failure. This causes the termination of the application.

Table 4-6 describes the L1P_PatternLimitPolicy_t enumeration.

Table 4-6 enum L1P_PatternLimitPolicy_t

Name	Description
L1P_PatternLimit_Disable	The limit on the number of allocated patterns is disabled (that is, no limit)
L1P_PatternLimit_Error	Exceeding the limit on the number of allocated patterns causes the pattern allocation to fail with L1P_NOMEMORY.
L1P_PatternLimit_Assert	Exceeding the limit on number of allocated patterns causes an assertion failure and the process abnormally terminate.
L1P_PatternLimit_Prune	Exceeding the limit on number of allocated patterns causes pattern allocations to be treated as though the nesting mode is L1P_NestingIgnore.

The value for L1P_CACHELINESIZE is 128.

Data types

Table 4-7 describes the L1P_Pattern_t struct.

Table 4-7 Struct L1P_Pattern_t

Name	Type	Width	Description
Size	Size_t	8 bytes	Size (in bytes) of the memory region that is allocated by the L1P_Allocate() routine.
ReadPattern	Void*	8 bytes	Virtual memory address for the read pattern
WritePattern	void*	8 bytes	Virtual memory address for the write/generated pattern

Table 4-8 on page 40 describes the L1P_Status_t struct.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Table 4-8 struct L1P_Status_t

Name	Type	Width	Description
Finished	uint64_	1 bit	Boolean that indicates that the perfect prefetcher has completed the list. This bit is cleared when a list has started executing, and set when the list has completed.
Abandoned		1 bit	Set if a failure to match causes list comparison to be abandoned.
Maximum		1 bit	Set if the length of the update reaches the maximum.

L1P perfect prefetcher configuration functions

Table 4-9 describes the L1P_PatternConfigure function.

Table 4-9 int L1P_PatternConfigure(uint64_t n)

Name	Description		
Parameter	uint64_t n	Input	The maximum number of L1 misses that can be tracked by the list.
Return Codes	L1P_NOMEMORY	The application was unable to allocate enough memory.	
	L1P_ ALREADYCONFIGURED	The L1p perfect prefetcher was already configured.	
Latency	The implementation might require system calls. On the CNK, this routine might use the glibc malloc() function internally. The malloc() function can then perform brk() or mmap() system calls to allocate storage.		
Description: Allocates enough storage so that the perfect prefetcher can track up to <n> L1 misses. Storage is retained until the following actions occur: ► L1P_Unconfigure() is performed. ► L1P_SetPattern() is performed. If the L1P_Configure() command is nested: ► If nesting mode has been set to L1P_NestingSaveContext, the L1P SPI pushes a L1P context structure onto a stack of L1P context structures. When an L1P_Unconfigure() function is called, this L1P context structure is restored. This is the default mode. ► If nesting mode has been set to L1P_NestingIgnore, the L1P SPI will reference count the L1P_Configures. When nested, the SPI does not write new pattern addresses into the L1p hardware. When the same number of L1P_Unconfigure() routines have been called, the L1P SPI returns to normal function. ► If the nesting node has been set to L1P_NestingFlat, then the L1P SPI will reference count and ignore nested L1P_Configures calls. All L1P_SetPattern() calls are ignored if they occur in a nested context. ► If nesting mode has been set to L1P_NestingError, the L1P SPI will display an error message and assert. This terminates the active process with a core file. This mode is to be used for debug purposes.			
Example			

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Name	Description	
Nested L1P_Configure: L1P_Configure(1000); // …code… L1P_Configure(1500); // …code… L1P_Unconfigure(); // …code… L1P_Unconfigure();		Unnested L1P_Configure L1P_Configure(1000); // …code… L1P_Unconfigure(); L1P_Configure(1500); // …code… L1P_Unconfigure();

Table 4-10 describes the int L1P_PatternUnconfigure() function.

Table 4-10 int L1P_PatternUnconfigure()

Name	Description	
Parameters	None	
Return codes	L1P_NOTCONFIGURED	The L1p has not been configured.
Latency	Implementation might require system calls. On CNK, this routine might use the glibc free() routine internally. The free() call can then perform brk() or munmap() system calls to free storage.	
Description: Deallocates storage used by the L1p SPI. If one is available, the L1P SPI will pop a L1P context structure from the stack of L1P context structures. The context will then be used to restore the previous L1P pattern status and pointers.		

L1p perfect prefetcher control functions

This section describes the L1p perfect prefetcher control functions.

Table 4-11 describes the L1P_PatternStart(int record) function.

Table 4-11 int L1P_PatternStart(int record)

Name	Description		
Parameters	int record	Input	Boolean that indicates whether L1P_PatternStart generates a new pattern. If set to TRUE, a new pattern is generated. Generation of a new pattern might occur simultaneously with the execution of an old pattern.
Return Codes:	L1P_PATTERNACTIVE	L1P_PatternStart() called while a pattern was active.	
	L1P_NOTCONFIGURED	The L1p has not been configured.	
Latency	Inlineable function call that accesses user-space memory mapped registers.		

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Name	Description
Description: The perfect prefetcher will start monitoring L1 misses and performing prefetch requests based on those misses. The 'record' parameter instructs the PatternStart to record the pattern of L1 misses for the next iteration. This L1P_PatternStart() should be called at the beginning of every entrance into the section of code that has been recorded.	

Table 4-12 describes the L1P_PatternPause() function.

Table 4-12 int L1P_PatternPause()

Name	Description	
Parameters	None	
Return codes	L1P_NOTCONFIGURED	The L1p prefetcher has not been configured.
Latency	Inlineable function call that accesses user-space memory mapped registers.	
Description: Suspends the active perfect prefetcher. The Linear Stream Prefetcher and the other three perfect prefetchers on the core continue to execute. This routine can be used in conjunction with L1P_PatternResume() function to avoid recording out-of-bound memory fetches, such as instructions performing a periodic printf. It can also be used to avoid sections of code that perform memory accesses that are inconsistent between iterations.		

Table 4-13 describes the L1P_PatternResume() function.

Table 4-13 int L1P_PatternResume()

Name	Description	
Parameters	None	
Return codes	L1P_NOTCONFIGURED	The L1p has not been configured.
Latency	Inlineable function call that accesses user-space memory mapped registers	
Description: Resumes the perfect prefetcher from the last pattern offset location. This routine can be used in conjunction with L1P_PatternPause() to avoid recording memory fetches that are not likely to repeat, such as instructions performing a periodic printf. It can also be used to avoid sections of code that perform memory accesses that are inconsistent between iterations.		

Table 4-14 describes the L1P_PatternStop() function.

Table 4-14 int L1P_PatternStop()

Name	Description	
Parameters	None	
Return codes	L1P_NOTCONFIGURED	The L1p has not been configured.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Name	Description
Latency	Inlineable function call that accesses user-space memory mapped registers
Description: Stops the perfect prefetcher and resets the list offsets to zero.	

Table 4-15 describes the L1P_PatternStatus function.

Table 4-15 int L1P_PatternStatus(L1P_State_t status)*

Name	Description		
Parameters	L1P_Status_t status	Output	Perfect prefetcher status bits
Return codes	None defined		
Latency	Inlineable function call that accesses user-space memory mapped registers		
Description: Stops the perfect prefetcher and resets the list offsets to zero.			

Table 4-16 describes the L1P_PatternStatus function.

Table 4-16 int L1P_PatternStatus(L1P_State_t status)*

Name	Description		
Parameters	L1P_Status_t status	Output	Perfect prefetcher status bits
Return codes	None defined		
Latency	Inlineable function call that accesses user-space memory mapped registers		
Description: Returns the current status for the L1 perfect prefetcher.			

Table 4-17 describes the L1P_PatternGetCurrentDepth function.

Table 4-17 int L1P_PatternGetCurrentDepth(uint64_t fetch _depth, uint64_t* generate _depth)*

Name	Description		
Parameters	uint64_t* fetch_depth	Output	Current depth of L1 misses in the prefetching pattern.
	uint64_t* generate_depth	Output	Current depth of L1 misses in the generated pattern.
Return codes	None defined		
Latency	Inlineable function call that accesses a read-only user-space memory mapped registers		

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Name	Description
Description: Returns the current pattern depths for the L1 perfect prefetcher. The pattern depth is the current index into the pattern that the L1p is executing. The fetch_depth parameter is used to determine how far in the current pattern/sequence the L1p has progressed. The generate depth parameter can be used to optimize the pattern length parameter to L1P_PatternConfigure() to reduce the memory footprint of the L1p pattern.	

Table 4-18 describes the L1P_PatternGetNestingMode function.

Table 4-18 int L1P_PatternGetNestingMode (L1P_PatternNest_t mode)*

Name	Description		
Parameters	L1P_PatternNest_t mode	Output	Old Nesting
Return Codes	None defined		
Latency	Inlineable function call that accesses user-space memory mapped registers		
Description: Returns the current nesting mode for the L1 perfect prefetcher. The supported nesting modes are L1P_NestingSaveContext, L1P_NestingIgnore, L1P_NestingFlat, L1P_NestingError. A description of each of these modes is in “Defines and enumerations” on page 38.			

Table 4-19 describes the L1P_PatternSetNestingMode function.

Table 4-19 int L1P_PatternSetNestingMode(L1P_PatternNest_t mode)

Name	Description		
Parameters	L1P_PatternNest_t mode	Input	New nesting mode
Return codes	None defined		
Latency	Inlineable function call that accesses user-space memory mapped registers		
Description: Returns the current status for the L1 perfect prefetcher. The default mode is L1P_NestingSaveContext. Other nesting modes are L1P_NestingIgnore, L1P_NestingFlat, L1P_NestingError. A description of each of these modes is in “Defines and enumerations” on page 38.			

Table 4-20 on page 45 describes the L1P_PatternSetAbandonThreshold function.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Table 4-20 int L1P_PatternSetAbandonThreshold(uint64_t numL1misses)

Name	Description		
Parameters	Uint64_t numL1misses	Input	The number of consecutive, non-matching L1 misses that will result in a pattern being abandoned. The valid range is 1 to 63. Default = 63
Return codes	None defined		
Latency	Inlineable function call that accesses user-space memory mapped registers		
Description: Sets the number of consecutive L1 misses that did not match the current location in the pattern. After this number has been exceeded, the prefetching activity will cease and the pattern will be marked as "Abandoned" in the L1P_Status_t structure returned by the L1P_PatternStatus() function.			

Table 4-21 describes the L1P_PatternSetAbandonThreshold function.

Table 4-21 int L1P_PatternSetAbandonThreshold(uint64_t numL1misses)

Name	Description		
Parameters	Uint64_t numL1misses	Input	The number of consecutive, non-matching L1 misses that will result in a pattern being abandoned. The valid range is 1 to 63. Default = 63
Return codes	None defined		
Latency	Inlineable function call that accesses user-space memory mapped registers		
Description: Sets the number of consecutive L1 misses that did not match the current location in the pattern. After this number has been exceeded, the prefetching activity will cease and the pattern will be marked as "Abandoned" in the L1P_Status_t structure returned by the L1P_PatternStatus() function.			

Table 4-22 describes the L1P_PatternGetAbandonThreshold function.

Table 4-22 int L1P_PatternGetAbandonThreshold(uint64_t numL1misses)*

Name	Description		
Parameters	Uint64_t* numL1misses	Output	The number of consecutive, non-matching L1 misses that will result in a pattern being abandoned.
Return codes	None defined		

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Name	Description		
Latency	Inlineable function call that accesses user-space memory mapped registers		
Returns the number of consecutive L1 misses that did not match the current location in the pattern. After this number has been exceeded, the prefetching activity will cease and the pattern will be marked as "Abandoned" in the L1P_Status_t structure returned by the L1P_PatternStatus() function.			

Table 4-23 describes the L1P_PatternSetEnable function.

Table 4-23 int L1P_PatternSetEnable(int enable)

Name	Description		
Parameters	int enable	Input	L1p pattern prefetcher enable flag
Return codes	None defined		
Latency	Inlineable function call that accesses user-space memory mapped registers		
Description: Sets a software enable/disable for L1p perfect prefetcher. This can be used to ascertain whether the usage of the prefetcher is improving performance.			

Table 4-24 describes the L1P_PatternGetEnable function.

Table 4-24 int L1P_PatternGetEnable(int enable)*

Name	Description		
Parameters	int enable	Output	L1p pattern prefetcher enable flag
Return codes	None defined		
Latency	Inlineable function call that accesses user-space memory mapped registers		
Description: Returns the software enable/disable for L1p perfect prefetcher.			

Explicit pattern management functions

This section describes explicit pattern management functions.

Table 4-25 describes the L1P_AllocatePattern function.

*Table 4-25 int L1P_AllocatePattern(uint64_t n, L1P_Pattern_t** ptr)*

Name	Description		
Parameters	uint64_t n	Input	The maximum number of L1 misses that can be tracked by the list.
	L1P_Pattern_t** ptr	Output	Pointer to an existing memory access pattern.
Return codes	L1P_NOMEMORY	Application was unable to allocate enough memory	

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Name	Description
Latency	Implementation might require system calls. On the CNK, this routine can use the glibc malloc() function internally. The malloc() function call can then perform brk() or mmap() system calls to allocate storage.
Description: Allocates storage to hold an L1p pattern of L1 miss addresses. This allows for the application to allocate storage for uninitialized patterns. This pattern storage can be passed to L1P_SetPattern(). Storage must be deallocated with L1P_DeallocatePattern().	

Table 4-26 describes the L1P_SetPattern function.

Table 4-26 int L1P_SetPattern(L1P_Pattern_t pattern)*

Name	Description		
Parameters	L1P_Pattern_t* pattern	Input	Pointer to a valid pattern
Return codes	L1P_NOTAPATTERN	The specified pointer is not a valid pointer.	
Latency	Implementation might require system calls. On CNK, since memory protection is a requirement, this routine will result in a system call to validate the pattern and setup physical addresses needed by the hardware.		
Description: Sets the perfect prefetcher's hardware registers with a given pattern. This allows for retaining several patterns of memory accesses and finer control of the L1p. It is not required for the default usage model. The L1p SPI will not deallocate the structure.			

Table 4-27 describes the L1P_GetPattern function.

*Table 4-27 int L1P_GetPattern(L1P_Pattern_t** pattern)*

Name	Description		
Parameters	L1P_Pattern_t** pattern	Output	Location to store the pointer to the pattern structure.
Return codes	L1P_NOTCONFIGURE	L1p has not been configured.	
Latency	Implementation might require system calls.		
Description: ► Returns pointers to the current L1p pattern. Later, the pattern pointer can then be passed back into L1P_SetPattern(). ► After L1P_GetPattern is called, the application will own the pattern and must call L1P_DeallocatePattern() to reclaim that storage. This allows pattern storage that is allocated through L1P_PatternConfigure() to be detached and retained for later usage. This allows for retaining several patterns of memory accesses and finer control of the L1p. It is not required for the default usage model.			

Table 4-28 on page 48 describes the L1P_DeallocatePattern function.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Table 4-28 int L1P_DeallocatePattern(L1P_Pattern_t ptr)*

Name	Description		
Parameters	L1P_Pattern_t* ptr	Input	Pointer to an existing memory access pattern.
Return codes	L1P_NOTAPATTERN	The specified pointer is not a valid pointer.	
Latency	Implementation might require system calls. On CNK, this routine can use the glibc free() routine internally. The free() call can then perform brk() or munmap() system calls to deallocate storage.		
Description: Deallocates storage previously assigned to the list of addresses. This allows for the application to deallocate storage for patterns that have been detached from normal L1p SPI control. Do not use L1P_DeallocatePattern() on non-detached patterns.			

Table 4-29 describes the L1P_PatternSetPatternLimit function.

Table 4-29 int L1P_PatternSetPatternLimit(L1P_PatternLimitPolicy_t policy, int numallocatedpatterns,)

Item	Description		
Parameters	L1P_PatternLimitPolicy_t policy	Input	Specifies the behavior when the number of allocated patterns has been exceeded.
	int numallocatedpatterns	Input	Number of allocated patterns that are allowed in the application
Return codes	None defined		
Latency	Inlineable function call that accesses user-space memory mapped registers		
Description: Sets behavior when the number of allocated patterns that the application can have active exceeds an artificial limit. This can be used to determine if there is a memory leak in the pattern allocations. The default policy is L1P_PatternLimit_Disable.			

Table 4-30 describes the L1P_PatternGetPatternLimit function.

Table 4-30 int L1P_PatternGetPatternLimit(L1P_PatternLimitPolicy_t policy, int* numactivelists)*

Item	Description		
Parameters	L1P_PatternLimitPolicy_t policy	Output	Behavior when the number of allocated patterns has been exceeded.
	int numactivelists	Output	Number of active/allocated patterns that are allowed in the application
Return codes	None defined		

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Item	Description
Latency	Inlineable function call that accesses user-space memory mapped registers
Description: Returns the current behavior when the number of allocated patterns that the application can have active exceeds that limit. The current limit is also returned.	

L1P linear stream prefetcher control functions

This section describes the L1p linear stream prefetcher control functions.

Table 4-31 describes the L1P_GetStreamAdaptiveMode function.

Table 4-31 int L1P_GetStreamAdaptiveMode(int adaptiveState)*

Item	Description		
Parameters	int adaptiveState	Output	Boolean that indicates whether adaptive mode is enabled or disabled. TRUE = enabled. FALSE = disabled.
Return codes	None defined		
Latency	Inlineable function call that accesses user-space memory mapped registers		
Description: Returns enable/disable status of the linear stream prefetcher's adaptation mode.			

Table 4-32 describes the L1P_SetStreamAdaptiveMode function.

Table 4-32 int L1P_SetStreamAdaptiveMode(int Enable)

Item	Description		
Parameters	int Enable	Input	Boolean that enables/disables adaptive mode. TRUE = enabled. FALSE = disabled.
Return codes	None defined		
Latency	Inlineable function call that accesses user-space memory mapped registers		
Enables or disables the linear stream prefetcher's depth adaptation mode			

Table 4-33 describes the L1P_GetStreamPolicy function.

Table 4-33 int L1P_GetStreamPolicy(L1P_StreamPolicy_t policy)*

Item	Description		
Parameters	L1P_StreamPolicy_t policy	Output	Current L1P stream policy
Return codes	None defined		

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Item	Description
Latency	Inlineable function call that accesses user-space memory mapped registers
Description: Returns the linear stream prefetch policy in the specified pointer. The policy controls when a stream is established.	

Table 4-34 describes the L1P_SetStreamPolicy function.

Table 4-34 int L1P_SetStreamPolicy(L1_StreamPolicy_t policy)

Item	Description		
Parameters	L1P_StreamPolicy_t policy	Input	New Policy
Return codes	L1P_PARMRANGE	An invalid stream policy was specified.	
Latency	Inlineable function call that accesses user-space memory mapped registers		
Description: Changes the linear stream prefetch policy. The policy controls when a stream is established.			

Table 4-35 describes the L1P_GetStreamDepth function.

Table 4-35 int L1P_GetStreamDepth(uint32_t depth)*

Item	Name		
Parameters	depth	Output	Integer 1 to 8 for the number of 128-byte lines ahead to fetch for all future established stream
Return codes	L1P_PARMRANGE	The specified address would have resulted in a segmentation violation.	
Latency	Inlineable function call that accesses user-space memory mapped registers		
Description: Returns the default stream depth when a new stream has been created. This default depth can be modified on a per stream basis using the adaptive mode (if enabled).			

Table 4-36 describes the L1P_SetStreamDepth function.

Table 4-36 int L1P_SetStreamDepth(uint32_t depth)

Item	Description		
Parameters	uint32_t depth	Input	Number of 128 byte lines ahead to fetch for all future established stream. The valid range is 1 to 8.
Return codes	L1P_PARMRANGE	Specified stream depth is not within the valid range.	
Latency	Inlineable function call that accesses user-space memory mapped registers		

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Item	Description
Description: When a new stream is established, the stream is set to the initial target prefetch depth specified by L1P_SetStreamDepth(). A streams prefetch depth can subsequently vary if the adaptive prefetch mode is enabled.	

Table 4-37 describes the L1P_GetStreamTotalDepth function.

Table 4-37 int L1P_GetStreamTotalDepth(uint32_t depth)*

Item	Description	
Parameters	depth	Integer 1 to 32 for total footprint of 128-byte lines that the stream engine will endeavor to use.
Return codes	L1P_PARMRANGE	The specified address will cause a segmentation violation.
Latency	Inlineable function call that accesses user-space memory mapped registers	
Description: Gets the number of 128-byte cache lines that can be used by the linear stream prefetcher. Unallocated lines will be used by the perfect prefetcher. This can help prevent thrashing between the prefetch algorithms.		

Table 4-38 describes the L1P_SetStreamTotalDepth function.

Table 4-38 int L1P_SetStreamTotalDepth(uint32_t depth)

Item	Description		
Parameters	Uint32_t depth	Input	Total footprint of 128-byte lines stream engine will endeavor to use. The valid range is 1 to 32.
Return codes	L1P_PARMRANGE	The specified total stream depth is not within the valid range.	
Latency	Inlineable function call that accesses user-space memory mapped registers		
Description: Sets the number of 128-byte cache lines that can be used by the linear stream prefetcher. The unallocated lines will continue to be used by the perfect prefetcher to help prevent thrashing between the prefetch algorithms.			

L1p error conditions

Table 4-39 on page 52 describes the L1p error conditions.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Table 4-39 L1p error conditions

Error code	Description
0	No error
L1P_NOMEMORY	There was not enough memory available to set up the L1P for the given pattern size.
L1P_PARMRANGE	The parameters that are passed to the L1P exceeded the valid range supported by the L1p hardware.
L1P_PATTERNACTIVE	Attempted to use a function when a pattern was already active. The application must issue an explicit L1P_PatternStop() before calling the function.
L1P_NOTAPATTERN	The application specified a pointer that either does not represent a generated pattern or the pointer is not valid.
L1P_ALREADYCONFIGURED	The L1p has already been configured without being previously unconfigured.
L1P_NOTCONFIGURED	The L1p has not been configured.

4.8.4 Performance considerations

This section describes the performance considerations for the L1p prefetcher.

Pattern loading

The L1p hardware can simultaneously load an existing list and create a new list. This can be used to do continuous refinement of the L1 cache miss list of addresses. However, in some cases, the first iteration through a routine creates a good enough list, such that additional gains would be overshadowed by the cost of periodically writing the refined list to DDR memory. This behavior can be controlled using the record parameter to the L1P_PatternStart() function.

Pattern creation overhead

There is a memory overhead versus performance overhead optimization with regards to list creation. When a list has been created, the application can remember the list for future reference. There is not an architectural limit to the number of lists that can be maintained. However, each list consumes memory and there is a bookkeeping overhead associated with tracking the list and keeping it resident in memory. There is also an opportunity cost associated with using that memory for other optimizations (for example, bigger lookup tables).

When switching between different patterns, the SPI performs one system call to install the new pattern's address in the L1p registers. Since a system call is a relatively heavy-weight operation, avoid switching patterns for small sections of code. It preferable to pause the pattern during these periods. Pausing or resuming a pattern is only a user-space MMIO write and can be accomplished with only a few instructions.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Prefetcher contention

Each A2 core's L1p is a shared resource: there are four hardware threads on each A2 core that shares the L1p. Each hardware thread can be running a list using its perfect prefetcher. Some hardware threads can be performing lots of linear stream prefetches while another hardware thread is executing a prefetch list pattern.

All of these activities compete for prefetch buffer space in the L1p. The L1P_SetStreamDepth(), L1P_SetStreamAdaptiveMode(), and L1P_SetStreamTotalDepth() functions are designed to be used by application developers to balance applications for optimal performance.

4.9 L2 atomic operations

The Blue Gene/Q nodes have support for atomic memory operations in the L2 cache. In some circumstances, atomic memory operations can be more efficient than standard PowerPC **larx/stcx** atomic instructions. The **larx/stcx** instructions require at least two operations for atomicity:

1. A load with reservation, which brings the data back to a processor general-purpose register (GPR)
2. A store operation, which pushes out the data

Typically, there is also a simple arithmetic operation interposed between those steps. Blue Gene/Q L2 atomics allow for a single load or store operation to perform a simple arithmetic operation in the L2 cache. This method saves the latency of the load. If the L2 atomic operation is a **store** operation code, the store operation is placed on the queue and the A2 core does not stall.

The CNK has support for Blue Gene/Q L2 atomic operations. However, the memory regions that contain L2 atomic memory must be predesignated. This predesignation is required because the CNK must create special memory translation entries for L2 atomic memory. Use the following SPI routine to predesignate memory:

```
uint64_t Kernel_L2AtomicsAllocate(void* atomic_vaddress, size_t length);
```

There are a limited number of memory translation entries. The CNK tries various combinations of mappings for atomic operations. However, the call can fail. If a failure occurs, try the call with a different virtual address.

4.10 Speculative execution

The Blue Gene/Q nodes contain a multiversion L2 memory cache that can be configured for speculative execution (also known as thread level speculation). This support enables the system software to simultaneously execute portions of the program on up to 128 hardware threads. The compiler generates multiple possible execution paths. The software runtime environment uses real-time performance data to determine which path is selected. If the system detects a conflict, it automatically reruns the code without speculation to ensure correct execution.

The Symmetric Multi-Processing Runtime (SMPRT) for the compiler and the CNK work together to configure the hardware for speculative execution support. For more information about using speculative execution, see the SMPRT #pragmas in the IBM XL compiler

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

documentation. Section 7.2.1, “IBM XL compilers” on page 80 describes the IBM XL compilers.

4.11 Support for dynamic linking

The CNK uses the Linux user callable facility for dynamic linking, which loads a library image into the virtual address space of an application process. It loads only the executable and linking format (ELF) sections that are required by the application into the physical memory space. To release the library from both virtual and physical memory, call the dlclose() function. This function is similar to the function that is used on the Linux operating system.

The CNK supports Python-based applications with minimal or no modifications. In these applications, it is necessary to communicate and load appropriate sections of the scientific codes into the application.

The CNK does not support fork() or exec() functions for shell commands that might be used in existing Python-based applications. If an application uses the fork() function, the exec() function commands, or runtime use of shell commands, it might require modification to execute correctly on the Blue Gene/Q system. Some of the required modifications might include:

- Replacing use of the Linux **cp**() (copy) command with inline code to copy files.
- Replacing shell commands with system calls to delete files, for example, use unlink (“path”) instead of system (**“rm** `-f path`”).
- Moving application setup to the front end node.

Each compute node independently requests dynamic libraries to be loaded. This solution relies on the file system caches on the Linux I/O node to avoid huge spikes in demand to the file system. It is possible that the file system caching might be insufficient for certain classes of dynamic applications.

4.12 Transactional memory

Transactional memory can be used to simplify simultaneous use of large numbers of threads. The Blue Gene/Q nodes contain a multiversion L2 memory cache that can be configured for transactional memory.

When transactional memory mode is used, the user defines the parallel work to be done. The user also defines which code is atomic. The hardware automatically detects memory read or write conflicts in the atomic region and the runtime retries the region. When many sections of code are marked atomic, performance can be reduced if these sections are frequently rerun.

The XLSMP runtime and the CNK work together to configure the hardware to support transactional memory. For more information about using transactional memory, see the SMPRT #pragmas in the IBM XL compiler documentation. Section 7.2.1, “IBM XL compilers” on page 80 describes the IBM XL compilers.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5

Compute Node Kernel interfaces

This chapter describes the kernel interfaces that the CNK provides for applications that run on compute nodes. It includes the following information:

- Lightweight principles
- Kernel access
- System calls

© Copyright IBM Corp. 2013. All rights reserved.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.1 Lightweight principles

The CNK is designed as a simple and lightweight kernel to maximize performance and reliability for *high-performance computing (HPC)* applications. It provides an environment for running user processes that is similar to Linux. It is not a full Linux kernel implementation. Instead, it implements a subset of the Linux functionality and a subset of the Portable Operating System Interface (POSIX) functionality.

5.2 Kernel access

The following interfaces can be used to access the CNK services:

- Application programming interface (API) provided by the library
- System programming interface (SPI) provided by the kernel
- System call (syscall) interface provided by the kernel

5.2.1 Application programming interfaces

The library provides various APIs. Each API can be used to send system calls to the CNK. Some of these APIs are lightweight wrappers to the kernel system calls. Some APIs provide more library functionality and might call more than one system call per invocation. This section describes three groups of supported APIs:

- File I/O and directory operations
- Sockets
- Process and threads

File I/O and directory operations

Instead of executing the system call on the compute node, the CNK might send the system call to the I/O node for execution. This is described as *function-shipping* the system call. Depending on the targeted file system, the CNK might function-ship system calls that are invoked by these APIs to the Common Input Output Services (CIOS) service on the I/O node. The CIOS is a user-level process that services applications in the compute node. The decision to function-ship a specific request depends on the file system that is targeted by the API and the specific system call that is used by the API. Table 5-1 shows the File I/O and directory operations.

Table 5-1 File I/O and directory operations

Function prototype	Header required	Description and type
int access(const char *path, int mode);	<unistd.h>	Determines the accessibility of a file. Mode: R_OK, X_OK, F_OK; returns 0 on success or -1 on error.
int chmod(const char *path, mode_t mode);	<sys/types.h> <sys/stat.h>	Changes the access permissions on an already open file. Mode: S_ISUID, S_ISGID, S_ISVTX, S_IRWXU, S_IRUSR, S_IWUSR, S_IXUSR, S_IRWXG, S_IRGRP, S_IWGRP, S_IXGRP, S_IRWXO, S_IROTH, S_IWOTH, and S_IXOTH. Returns 0 if the permissions are correct or -1 on error.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Function prototype	Header required	Description and type
int chown(const char *path, uid_t owner, gid_t group);	<sys/types.h> <sys/stat.h>	Changes the owner and group of a file.
int close(int fd);	<unistd.h>	Closes a file descriptor. Returns 0 on success or -1 on error.
int dup(int fd);	<unistd.h>	Duplicates an open descriptor. Returns a new file descriptor on success or -1 on error.
int dup2(int fd, int fd2);	<unistd.h>	Duplicates an open descriptor. Returns a new file descriptor on success or -1 on error.
int fchmod(int fd, mode_t mode);	<sys/types.h> <sys/stat.h>	Changes the mode of a file. Returns 0 on success or -1 on error.
int fchown(int fd, uid_t owner, gid_t group);	<sys/types.h> <unistd.h>	Changes the owner and group of a file. Returns 0 on success or -1 on error.
int fcntl(int fd, int cmd, int arg);	<sys/types.h> <unistd.h> <fcntl.h>	Manipulates a file descriptor. Supported commands are F_GETFL, F_DUPFD, F_GETLK, F_SETLK, F_SETLKW, F_GETLK64, F_SETLK64, F_SETLKW64.
int fstat(int fd, struct stat *buf);	<sys/types.h> <sys>/<stat.h>	Gets the file status. Returns 0 if correct or -1 on error.
int stat64(const char *path, struct stat64 *buf);	<sys/types.h> <sys>/<stat.h>	Gets the file status.
int statfs(const char *path, struct statfs *buf);	<sys/vfs.h>	Gets the file system statistics.
long fstatfs64 (unsigned int fd, size_t sz, struct statfs64 buf);	<sys/vfs.h>	Gets the file system statistics.
int fsync(int fd);	<unistd.h>	Synchronizes changes to a file. Returns 0 on success or -1 on error.
int ftruncate(int fd, off_t length);	<sys/types.h> <unistd.h>	Truncates a file to a specified length. Returns 0 on success or -1 on error.
int ftruncate64(int fildes, off64_t length);	<unistd.h>	Truncates a file to a specified length for files larger than 2 GB. Returns 0 on success or -1 on error.
int lchown(const char *path, uid_t owner, gid_t group);	<sys/types.h> <unistd.h>	Changes the owner and group of a symbolic link. Returns 0 on success or -1 on error.
int link(const char *existingpath, const char *newpath);	<unistd.h>	Links to a file. Returns 0 on success or -1 on error.
off_t lseek(int fd, off_t offset, int whence);	<sys/types.h> <unistd.h>	Moves the read/write file offset. Returns 0 on success or -1 on error.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Function prototype	Header required	Description and type
int _llseek(unsigned int fd, unsigned long offset_high, unsigned long offset_low, loff_t *result, unsigned int whence);	<unistd.h> <sys/types.h> <linux/unistd.h> <errno.h>	Moves the read/write file offset.
int lstat(const char *path, struct stat *buf);	<sys/types.h> <sys>/<stat.h>	Gets the symbolic link status. Returns 0 on success or -1 on error.
int lstat64(const char *path, struct stat64 *buf);	<sys/types.h> <sys/stat.h>	Gets the symbolic link status. Determines the size of a file larger than 2 GB.
int open(const char *path, int oflag, mode_t mode);	<sys/types.h> <sys>/<stat.h> <fcntl.h>	Opens a file. oflag: O_RDONLY, O_WRONLY, O_RDWR, O_APPEND, O_CREAT, O_EXCL, O_TRUNC, O_NOCTTY, O_SYNC, mode: S_IRWXU, S_IRUSR, S_IWUSR, S_IXUSR, S_IRWXG, S_IRGRP, S_IWGRP, S_IXGRP, S_IRWXO, S_IROTH, S_IWOTH, and S_IXOTH. Returns the file descriptor on success or -1 on error.
ssize_t pread(int fd, void *buf, size_t nbytes, off64_t offset);	<unistd.h>	Reads from a file at offset. Returns the number of bytes read on success, 0 if end of file, or -1 on error.
ssize_t pwrite(int fd, const void *buf, size_t nbytes, off64_t offset);	<unistd.h>	Writes to a file at offset; returns the number of bytes written on success or -1 on error.
ssize_t read(int fd, void *buf, size_t nbytes);	<unistd.h>	Reads from a file. Returns the number of bytes read on success, 0 if end of file, or -1 on error.
int readlink(const char *path, char *buf, int bufsize);	<unistd.h>	Reads the contents of a symbolic link. Returns the number of bytes read on success or -1 on error.
ssize_t readv(int fd, const struct iovec iov[], int iovcnt)	<sys/types.h> <sys/uio.h>	Reads a vector. Returns the number of bytes read on success or -1 on error.
int rename(const char *oldname, const char *newname);	<stdio.h>	Renames a file. Returns 0 on success or -1 on error.
int stat(const char *path, struct stat *buf);	<sys/types.h> <sys/stat.h>	Gets the file status. Returns 0 on success or -1 on error.
int stat64(const char *path, struct stat64 *buf);	<sys/types.h> <sys/stat.h>	Gets the file status.
int statfs (char *path, struct statfs *buf);	<sys/types.h> <sys/stat.h>	Gets the file system statistics.
long statfs64 (const char *path, size_t sz, struct statfs64 *buf);	<sys/statfs.h>	Gets the file system statistics.
int symlink(const char *actualpath, const char *sympath);	<unistd.h>	Makes a symbolic link to a file. Returns 0 on success or -1 on error.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Function prototype	Header required	Description and type
int truncate(const char *path, off_t length);	<sys/types.h> <unistd.h>	Truncates a file to a specified length. Returns 0 on success or -1 on error.
truncate64(const char *path, off_t length);	<unistd.h> <sys/types.h>	Truncates a file to a specified length.
mode_t umask(mode_t cmask);	<sys/types.h> <sys/stat.h>	Sets and gets the file mode creation mask. Returns the previous file mode creation mask.
int unlink(const char *path);	<unistd.h>	Removes a directory entry. Returns 0 on success or -1 on error.
int utime(const char *path, const struct utimbuf *times);	<sys/types.h> <utime.h>	Sets the file access and modification times. Returns 0 on success or -1 on error.
ssize_t write(int fd, const void *buff, size_t nbytes);	<unistd.h>	Writes to a file. Returns the number of bytes written on success or -1 on error.
ssize_t writev(int fd, const struct iovec iov[], int iovcntl);	<sys/types.h> <sys/uio.h>	Writes a vector. Returns the number of bytes written on success or -1 on error.
int chdir(const char *path);	<unistd.h>	Changes the working directory. Returns 0 on success or -1 on error.
char *getcwd(char *buf, size_t size);	<unistd.h>	Gets the path name of the current working directory. Returns the buf value on success or NULL on error.
int getdents(int fildes, char **buf, unsigned nbyte);	<sys/types.h>	Gets the directory entries in a file system. Returns 0 on success or -1 on error.
int getdents64(unsigned int fd, struct dirent *dirp, unsigned int count);	<sys/dirent.h>	Gets the directory entries in a file system.
int mkdir(const char *path, mode_t mode);	<sys/types.h> <sys/stat.h>	Makes a directory; mode: S_IRUSR, S_IWUSR, S_IXUSR, S_IRGRP, S_IWGRP, S_IXGRP, S_IROTH, S_IWOTH, and S_IXOTH. Returns 0 on success or -1 on error.
int rmdir(const char *path);	<unistd.h>	Removes a directory. Returns 0 on success or -1 on error.

Sockets

The socket support allows the creation of both outbound and inbound socket connections with standard Linux APIs. For example, an outbound socket can be created by calling the socket() function, followed by the connect() function. An inbound socket can be created by calling the socket() function followed by the bind(), listen(), and accept() functions.

Communication through the socket is provided by the glibc send(), recv(), and select() function calls. These function calls run the socketcall() system call with different parameters.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The CNK provides socket support through the standard Linux socketcall() system call. The CNK function-ships the socketcall() system call to the CIOS, which performs the requested operation.

Table 5-2 summarizes the supported socket APIs.

Table 5-2 Supported socket APIs

Function prototype	Header required	Description and type
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);	<sys/types.h> <sys/socket.h>	Extracts the connection request on the queue of pending connections. Creates a new connected socket. Returns a file descriptor on success or -1 on error.
int bind(int sockfd, const struct sockaddr *my_addr, socklen_t addrlen);	<sys/types.h> <sys/socket.h>	Assigns a local address. Returns 0 on success or -1 on error.
int connect(int socket, const struct sockaddr *address, socklen_t address_len);	<sys/types.h> <sys/socket.h>	Connects a socket. Returns 0 on success or -1 on error.
int getpeername(int socket, struct sockaddr *restrict address, socklen_t *restrict address_len);	<sys/socket.h>	Gets the name of the peer socket. Returns 0 on success or -1 on error.
int getsockname(int socket, struct sockaddr *restrict address, socklen_t *restrict address_len);	<sys/types.h> <sys/socket.h>	Gets the name of the peer socket. Returns 0 on success or -1 on error.
int getsockopt(int s, int level, int optname, void *optval, socklen_t *optlen);	<sys/socket.h>	Manipulates options that are associated with a socket. Returns 0 on success or -1 on error.
int listen(int sockfd, int backlog);	<sys/types.h> <sys/socket.h>	Accepts connections. Returns 0 on success or -1 on error.
int poll(struct pollfd *fds*[], nfds_t *nfds*, int *timeout*);	#include <poll.h>	The poll() function provides applications with a mechanism for multiplexing input/output over a set of file descriptors.
ssize_t recv(int s, void *buf, size_t len, int flags);	<sys/types.h> <sys/socket.h>	Receives a message only from a connected socket. Returns 0 on success or -1 on error.
ssize_t recvfrom(int s, void *buf, size_t len, int flags, struct sockaddr *from, socklen_t *fromlen);	<sys/types.h> <sys/socket.h>	Receives a message from a socket regardless of whether it is connected. Returns 0 on success or -1 on error.
ssize_t recvmsg(int s, struct msghdr *msg, int flags);	<sys/types.h> <sys/socket.h>	Receives a message from a socket regardless of whether it is connected. Returns 0 on success or -1 on error.
ssize_t send(int socket, const void *buffer, size_t length, int flags);	<sys/types.h> <sys/sockets.h>	Sends a message only to a connected socket. Returns 0 on success or -1 on error.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Function prototype	Header required	Description and type
ssize_t sendto(int socket, const void *message, size_t length, int flags, const struct sockaddr *dest_addr, socklen_t dest_len);	<sys/types.h> <sys/socket.h>	Sends a message on a socket. Returns 0 on success or -1 on error.
ssize_t sendmsg(int s, const struct msghdr *msg, int flags);	<sys/types.h> <sys/socket.h>	Sends a message on a socket. Returns 0 on success or -1 on error.
int setsockopt(int s, int level, int optname, const void *optval, socklen_t optlen);	<sys/types.h> <sys/socket.h>	Manipulates options that are associated with a socket. Returns 0 on success or -1 on error.
int shutdown(int s, int how);	<sys/socket.h>	Causes all or part of a connection on the socket to shut down. Returns 0 on success or -1 on error.
int socket(int domain, int type, int protocol);	<sys/types.h> <sys/socket.h>	Opens a socket. Returns a file descriptor on success or -1 on error.
int socketpair(int d, int type, int protocol, int sv[2]);	<sys/types.h> <sys/socket.h>	Creates an unnamed pair of connected sockets. Returns 0 on success or -1 on error.

Processes and threads

This section shows the supported APIs that are associated with the control and access of processes and threads executing on the compute nodes. Additional APIs might operate correctly if they use the system calls that are supported by the CNK.

Table 5-3 lists the supported process and thread management APIs.

Table 5-3 Supported APIs for managing threads that run on compute nodes

Function prototype	Header required	Description and type
gid_t getgid(void);	<unistd.h>	Gets the group ID.
pid_t getpid(void);	<unistd.h>	Gets the process ID. This ID is a nonzero value that uniquely identifies a process with a node. This ID is not unique across all of the processes in a job.
int getrlimit(int resource, struct rlimit *rlp)	<sys/resource.h>	Gets information about resource limits.
int getrusage(int who, struct rusage *r_usage);	<sys/resource.h>	Gets information about resource use. All time reported is attributed to the user application, so the reported system time is always zero.
uid_t getuid(void);	<unistd.h>	Gets the user ID.
int setrlimit(int resource, const struct rlimit *rlp);	<sys/resource.h>	Sets resource limits. Only RLIMIT_CORE can be set.
clock_t times(struct tms *buf);	<sys/times.h>	Gets the process times. All time reported is attributed to the user application, so the reported system time is always zero.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Function prototype	Header required	Description and type
int brk(void *end_data_segment);	<unistd.h>	Changes the allocated size in the heap segment.
void exit(int status)	<stdlib.h>	Terminates a process.
int uname(struct utsname *buf);	<sys/utsname.h>	Gets the name of the current system and other information, for example, the version and release.
void *mmap(void *addr, size_t len, int prot, int flags, int fildes, off_t off);	#include <sys/mman.h>	Establishes a mapping between a process address space and a file, shared memory object, or typed memory object.
int shm_open(const char **name*, int *oflag*, mode_t *mode*);	#include <sys/mman.h> #include <sys/stat.h> #include <fcntl.h>	Creates and opens a new, or opens an existing, POSIX shared memory object.
int pthread_create(pthread_t **thread*, const pthread_attr_t **attr*, void *(**start_routine*) (void *), void **arg*);	#include <pthread.h>	Starts a new thread in the calling process.
void pthread_exit(void *retval);	#include <pthread.h>	Terminates the calling thread.
int pthread_yield(void);	<sched.h>	Forces the running thread to relinquish the processor.
pthread_setschedprio(pthread_t thread, int prio);	#include <pthread.h>	Sets the scheduling priority of the thread.
int nanosleep(const struct timespec *req, struct timespec *rem);	#include <time.h>	Suspends the execution of the calling thread until either at least the time specified.
int kill(pid_t pid, int sig);	<signal.h>	Sends a signal. A signal can be sent only to the same process.
int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);	<signal.h>	Allows the calling process to examine and specify the action to be associated with a specific signal.
typedef void (*sighandler_t)(int) sighandler_t signal(int signum, sighandler_t handler);	<signal.h>	This interface is supported for existing applications. Use the sigaction interface for new applications.
int shm_open(const char **name*, int *oflag*, mode_t *mode*);	#include <sys/mman.h> #include <sys/stat.h> #include <fcntl.h>	Creates and opens a new, or opens an existing, POSIX shared memory object.

5.2.2 System programming interface

The SPI that is provided by the CNK allows low-level access to Blue Gene/Q-specific interfaces. Many of the SPIs are implemented using special internal Blue Gene/Q system calls. Some of the SPIs are implemented in the user state and do not require entry into the kernel.

For information about kernel SPIs, see the installed documentation in the `/bgsys/drivers/ppcfloor/spi/doc/html` directory. This information is also available on the Knowledge Center tab in Navigator.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The following tables list the header files that contain the SPIs and describe the SPIs in the files.

Table 5-4 lists the supported SPI header files.

Table 5-4 SPI header files and the interfaces they provide

Interface file	Description
`/spi/include/kernel/collective.h`	Allocates the collective class route IDs, and sets the configuration of collective class routes.
`/spi/include/kernel/gi.h`	Allocates the global interrupt class route IDs, and sets the configuration of global interrupt class routes.
`/spi/include/kernel/location.h`	Provides location information including the node location in the block, process information with the node, the core in the node, and the hardware thread in the core.
`/spi/include/kernel/memory.h`	Manages regions of memory within the compute node. Opens the persistent memory handle with the persist_open kernel function.
`/spi/include/kernel/process.h`	Retrieves information about the process. This information includes how many processes are configured per node, how many processors are assigned to a process, which hardware threads are assigned to the process.
`/spi/include/kernel/spec.h`	Controls the speculative execution of threads.
`/spi/include/kernel/thread.h`	Retrieves scheduler information about active and runnable pthreads on a hardware thread.
`/spi/include/kernel/MU.h`	Controls and retrieves information from the messaging unit hardware.
`/spi/include/kernel/rdma.h`	Interfaces for an abbreviated version of OFED RDMA CM from the compute node to its I/O node.
`/spi/include/kernel/sendx.h`	Provides extensions to the light-weight kernel for user-defined function-shipping exchanges with a dynamically loaded library attached to the sysiod daemon on the I/O node. The extensions are in a derived plug-in class that is coded by the user on the I/O node where the base class is defined in `/ramdisk/include/services/Plugin.h`. The function-ship operations include simple message passing to the more complex operations of RDMA and using file descriptors.
`/spi/include/l1p/pprefetch.h`	Controls the perfect prefetcher hardware.
`/spi/include/l1p/sprefetch.h`	Controls the stream prefetcher hardware.
`/spi/include/l1p/flush.h`	Causes an L1P flush of all pending **`load`** and **`store`** operations to the L2 cache.
`/spi/include/l2/atomic.h`	L2 atomic operations.
`/spi/include/l2/barrier.h`	L2 atomic-based barrier operations.
`/spi/include/l2/lock.h`	L2 atomic-based lock operations.

5.3 System calls

The system call is the lowest-level interface that an application can use to access kernel functions. It is typically best to use the library APIs as the primary interface to the kernel. See 5.2.1, “Application programming interfaces” on page 56. However, direct execution of system

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

calls is allowed and is sometimes required. Example 5-1 shows an example of a direct invocation of a system call.

Example 5-1 Direct invocation of a system call

```
#include <unistd.h>
#include <sys/syscall.h>
#include <sys/types.h>
main(int argc, char *argv[])
{
pid_t tid;
tid = syscall(SYS_gettid);
}
```

Supported Linux APIs

The following system calls are supported by the CNK.

ftruncate64	kill	pwrite64	sched_yield	time
futex	lseek	read	setitimer	times
getcwd	lstat	readlink	setrlimit	tmwrite
getdents	lstat64	readv	sigaction	truncate
getdents64	mkdir	rename	signals	truncate64
getgroups	mmap	rmdir	sigprocmask	uid
getitimer	mremap	rt_sigaction	socketcall	umask
getpid	munmap	rt_sigprocmask	stat	uname
getrlimit	nanosleep	sched_get_priority_max	stat64	unlink
getrusage	open	sched_get_priority_min	statfs	utime
gettid	poll	sched_getaffinity	statfs64	write
gettimeofday	prctl	sched_getparam	symlink	writev
ioctl	pread64	sched_setscheduler		

All other system calls return the errno value ENOSYS.

Additional information about system calls

For more information about Linux system calls, see the syscalls(2) manual page.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

6

Parallel paradigms

This chapter contains information about the parallel paradigms that are offered on the Blue Gene/Q system. These paradigms include MPI for distributed-memory architecture and the OpenMP API for shared-memory architectures. These paradigms are referred to as *high-performance computing* (*HPC*). The Blue Gene/Q system also offers a paradigm where applications do not require communication between tasks and each node is running a different instance of the application. This paradigm is known as *high-throughput computing* (*HTC*).

This chapter addresses the following topics:

- Programming model
- Blue Gene/Q MPI implementation
- Blue Gene/Q MPI extensions
- MPI functions
- Compiling MPI programs on the Blue Gene/Q system
- OpenMP
- Multiple Program, Multiple Data

© Copyright IBM Corp. 2013. All rights reserved.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

6.1 Programming model

The Blue Gene/Q system has a distributed memory system and uses explicit message passing to communicate between tasks that are running on different nodes. Each node has shared memory. The OpenMP API and thread parallelism are supported.

The MPI standard is also supported. For more information, see the Message Passing Interface Forum site on the web at the following address:

http://www.mpi-forum.org/

The Blue Gene/Q MPI implementation uses the IBM Parallel Active Messaging Interface (PAMI) as a low-level messaging interface. The Blue Gene/Q PAMI implementation directly accesses the Blue Gene/Q hardware through the message unit system programming interface (MUSPI). The MPI, PAMI, and MUSPI interfaces are public, supported interfaces on the Blue Gene/Q system. These interfaces can be used by applications to perform communication operations. For more information about PAMI, see the installed documentation in the `/bgsys/drivers/ppcfloor/comm/doc/html` directory. This information is also available on the Knowledge Center tab in Navigator.

Other programming paradigms for the Blue Gene/Q system use one or more of the supported software interfaces, as illustrated in Figure 6-1. Support for these alternative paradigms is provided by the open source communities that develop them.

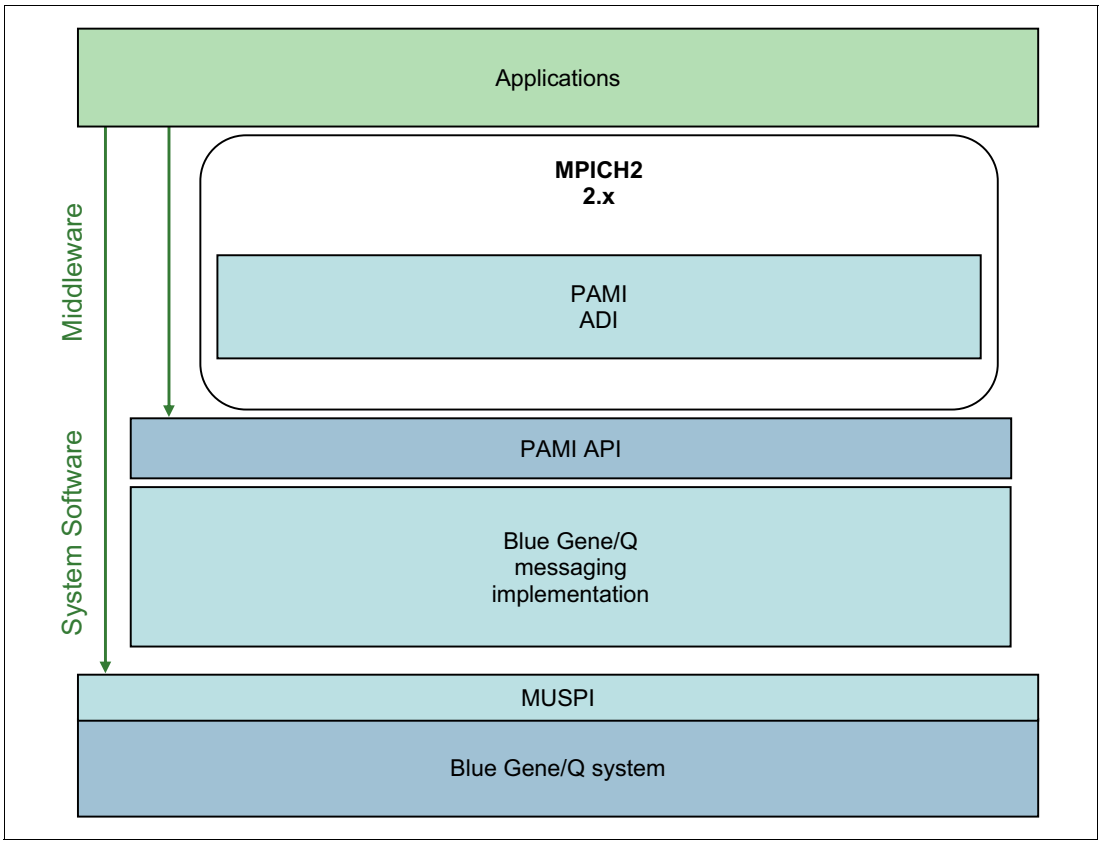


Figure 6-1 Messaging software stack

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

6.2 Blue Gene/Q MPI implementation

The MPI implementation on the Blue Gene/Q system supports the MPI-2.1 standard, except for the process creation and management functions. The MPI-2.1 standard is available from the following website:

http://www.mpi-forum.org/docs/docs.html

The MPI implementation on the Blue Gene/Q system is derived from the MPICH2 implementation of the Mathematics and Computer Science Division (MCS) at Argonne National Laboratory. For more information, see the MPICH2 website:

http://www.mcs.anl.gov/research/projects/mpich2/

To support the Blue Gene/Q hardware, the following additions and modifications are made to the MPICH2 software architecture:

- An additional Blue Gene/Q driver implements the MPICH2 abstract device interface (ADI).
- MPIX functions create hardware-specific MPI extensions.

The most significant change is that collective operations can use different networks in different circumstances.

Section 6.2.1, “High-performance network for efficient parallel execution” on page 67 summarizes the different networks on the Blue Gene/Q system and network routing.

Sections 6.2.2, “Forcing MPI to allocate too much memory” on page 69 through 6.2.7, “Buffer alignment sensitivity” on page 71 contain information about several sample MPI codes. These sections explain some of the implementation-dependent behaviors of the MPI library. Section 6.3.2, “Determining hardware properties” on page 73 contains an automatic optimization technique that is available on the Blue Gene/Q MPI implementation.

6.2.1 High-performance network for efficient parallel execution

The Blue Gene/Q system provides two different communication networks for hardware acceleration for certain collective operations.

Five-dimensional torus network

The five-dimensional (5D) torus network provides point-to-point and collective communication facilities. The network has an embedded arithmetic logic unit (ALU) system for doing math operations (for example, during a reduction operation). The ALU supports floating-point data and supports sum and min or max operations. Hardware acceleration with the ALUs is available on subcommunicators and MPI_COMM_WORLD. However, a limited number of class routes are available. If a single job uses multiple subcommunicators, the hardware acceleration is not available on some subcommunicators. Section 6.3.1, “Changing class-route usage at run time” on page 72 describes how to control which communicators are hardware accelerated.

For point-to-point messaging, the route from a sender to a receiver on a torus network has the following two possible paths:

Deterministic routing Packets from a sender to a receiver go along the same path. One advantage of this path is that the packet order is always maintained without additional logic. However, this technique also creates network hot spots if several point-to-point communications occur simultaneously and their deterministic routes cross on some node.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Adaptive routing Different packets from the same sender to the same receiver can travel along different paths. The exact route is determined at run time, depending on the current load. This technique generates a more balanced network load but introduces a latency penalty.

The appropriate deterministic or adaptive routing depends on the protocol that is used for communication. The Blue Gene/Q MPI implementation supports four different protocols:

Immediate protocol The immediate protocol is used for messages smaller than or equal to 112 bytes. These messages are always deterministically routed.

Short protocol The short protocol is used for messages that comprise a single packet. Internal protocol metadata uses 16 bytes of the torus packet payload, leaving a maximum of 496 bytes of application data that can be transferred with the short protocol. These messages are always deterministically routed.

Eager protocol The eager protocol is used for medium-sized messages. This protocol sends messages to the receiver without negotiating that the receiver is ready to receive the message. This process can cause an unexpected message to be received, which requires an unexpected memory buffer to be allocated and maintained until the receiver posts a matching MPI_Recv value. For more information, see Section 6.2.2, “Forcing MPI to allocate too much memory” on page 69. The eager protocol uses deterministic routes for its packets.

Rendezvous protocol Large messages are sent using the rendezvous protocol. In this case, an initial connection between the two partners is established. The receiver uses remote direct memory access to obtain data from the sender only after that connection is established. This protocol uses adaptive routing and is optimized for maximum bandwidth. By default, the MPI send operations use the rendezvous protocol, instead of the eager protocol, for messages larger than 2048 bytes. The initial rendezvous handshake increases the latency.

The Blue Gene/Q MPI library supports a PAMID_RZV environment variable, which can be set by the **`runjob`** command. Use this variable to set the message size, in bytes, as described in the preceding list, for the rendezvous protocol. Consider the following guidelines:

- Decrease the rendezvous threshold if any of the following situations are true:
  - High overlap of communication and computation is required.
  - Eager messages are creating artificial hot spots, resulting in network congestion.
  - Low latency is not required for medium-length messages.
  - Unexpected messages are causing the application to run out of memory.
- Increase the rendezvous threshold if any of the following situations are true:
  - Most communication is to the nearest neighbor.
  - Low latency is required for medium-length messages.
- Set the rendezvous threshold to 0 if the following situation is true:
  - High memory use due to eager connection data structures in a large-scale job is causing the application to run out of memory.

Several other environment variables can be used to customize MPI communications. See Appendix D, “MPI and CNK environment variables” on page 129 for descriptions of these environment variables.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Use the following guidelines to maximize efficiency for MPI applications on the Blue Gene/Q system:

- Overlap communication and computation using the MPI_Irecv() and MPI_Isend() functions, which allow the messaging unit (MU) hardware to complete the data transfer in the background.
- Avoid load imbalance.
- Avoid buffered and synchronous sends. Post receives in advance.

  The MPI standard defines several specialized communication modes in addition to the standard send function, MPI_Send(). Avoid the buffered send function, MPI_Bsend(), because it causes the MPI library to perform additional memory copies. Avoid using the synchronous send function, MPI_Ssend(), because it is not a local operation. It incurs an increased latency compared to the standard send without saving memory allocation.
- Avoid vector data and noncontiguous data types.

  While the MPI-derived data types can elegantly describe the layout of complex data structures, using these data types is generally detrimental to performance. Many MPI implementations, including the Blue Gene/Q MPI implementation, pack (that is, memory copy) such data objects before sending them. This packing of data objects is contrary to the original purpose of MPI-derived data types, which is to avoid such memory copies. Memory copies are particularly expensive on Blue Gene/Q because the network hardware is extremely fast relative to the processor clock. To improve application performance, avoid noncontiguous MPI data types and memory copies.

6.2.2 Forcing MPI to allocate too much memory

Avoid forcing MPI to allocate too much memory, which is easy to do with basic code. For example, the legal MPI code shown in Example 6-1 might force too much memory allocation. Memory must be allocated to temporarily store the incoming message data from CPU1, and to add an unexpected request object to the MPI receive queue, until a matching receive is posted by CPU2. If too much memory is allocated, failures occur because of excessive message buffering.

Example 6-1 MPI code that can cause excessive memory allocation

```
MPI_Isend(cpu2, tag1);
MPI_Isend(cpu2, tag2);
...
MPI_Isend(cpu2, tagn);
...
MPI_Recv(cpu1, tagn);
MPI_Recv(cpu1, tagn-1);
...
MPI_Recv(cpu1, tag1);
```

In addition to memory allocation issues, application performance can be degraded due to the presence of many unexpected messages. The message request queue is searched linearly to meet MPI matching requirements. If many messages are on the request queue, perhaps due to a flood of unexpected messages, the search can take longer.

You can accomplish the same goal and avoid memory allocation and request queue performance issues by recoding as shown in Example 6-2 on page 70.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Example 6-2 MPI code that can prevent excessive memory allocation

```
MPI_Isend(cpu2, tag1);
MPI_Isend(cpu2, tag2);
...
MPI_Isend(cpu2, tagn);
...
MPI_Recv(cpu1, tag1);
MPI_Recv(cpu1, tag2);
...
MPI_Recv(cpu1, tagn);
```

The Blue Gene/Q MPI rendezvous protocol does not allocate a temporary buffer to receive unexpected messages. However, a request object must still be allocated for the receive queue until the matching send information is received. The rendezvous protocol reduces memory use from unexpected buffers, but it does not prevent memory allocation issues or receive queue performance issues. Correct buffer allocation prevents most problems by significantly reducing the memory footprint and presence of unexpected messages.

6.2.3 Not waiting for the MPI_Test function

According to the MPI standard, an application must either wait or continue testing until the MPI_Test function returns true. If the application does not wait, small memory leaks might occur. These leaks can accumulate over time and cause a memory overrun. Example 6-3 shows the code and the problem.

Example 6-3 Potential memory overrun caused by not waiting for the MPI_Test function

```
req = MPI_Isend( ... );
MPI_Test (req);
... do something else; forget about req ...
```

Use the MPI_Wait function or loop until the MPI_Test function returns *true*.

6.2.4 Flooding the network with messages

The code shown in Example 6-4 is legal, but it floods the network with messages. It can cause CPU 0 to run out of memory. This code might work in some cases, but it is not scalable.

Example 6-4 Flood of messages that can cause a memory overrun

```
if (rank != 0) MPI_Send( ... to rank 0 ...);
else if (rank == 0) {
   for (i=1; i<n; i++) MPI_Recv(... from rank i ...);
}
```

6.2.5 Deadlocking the system

The code shown in Example 6-5 on page 71 does not conform to the MPI standard. Each side does a blocking send to its communication partner before posting a receive acknowledgment for the message coming from the other partner.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Example 6-5 MPI code that can deadlock the system

```
if (task == task1) {
  MPI_Send(... to task2 ...);
  MPI_Recv(... from task2 ...);
}
if (task == task2) {
  MPI_Send(... to task1 ...);
  MPI_Recv(... from task1 ...);
}
```

This code has a high probability of deadlocking the system. Ensure that your code conforms to the MPI specification by changing the order of sends and receives or by using nonblocking communication calls.

Do not rely on the runtime system to correctly handle nonconforming MPI code. However, it is easier to debug such situations when you receive a runtime error message than to try and detect a deadlock and trace it back to its root cause.

6.2.6 Violating MPI buffer ownership rules

A number of problems can occur when the send/receive buffers that participate in asynchronous message-passing calls are accessed before it is legal to do so. This section shows examples of incorrect code.

One common mistake is to write to a send buffer before the MPI_Wait() function for that request has completed as shown in Example 6-6.

Example 6-6 Write to a send buffer before the MPI_Wait() function has completed

```
req = MPI_Isend(buffer,&req);
buffer[0] = something;
MPI_Wait(req);
```

The code in Example 6-6 causes a race condition with any MPI implementation. Depending on runtime factors that the application cannot control, sometimes the old buffer[0] is sent and sometimes the new value is sent.

In Example 6-7, a receive buffer is read before MPI_Wait() function because the asynchronous receive request has completed.

Example 6-7 Receive buffer before MPIWait() function completes

```
req = MPI_Irecv(buffer);
z = buffer[0];
MPI_Wait (req);
```

The code shown in Example 6-7 is illegal. The contents of the receive buffer are not guaranteed until after MPI_Wait() function is called.

6.2.7 Buffer alignment sensitivity

The MPI implementation on the Blue Gene/Q system is sensitive to the alignment of the buffers that are being sent or received. Aligning buffers on 32-byte boundaries can improve performance. If the buffers are at least 32-bytes aligned, the messaging software can use

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

internal math routines that are quad-processing extension (QPX) optimized. Additionally, the L1 cache is optimized on 64-byte boundaries.

For buffers that are declared in static (global) storage, use **attribute{_}_((aligned(32)))** on the declaration as shown in Example 6-8.

Example 6-8 Buffers that are declared in static (global) storage

```
struct DataInfo
{
unsigned int iarray[256];
unsigned int count;
} data_info __attribute__ ((aligned ( 32)));
or
unsigned int data __attribute__ ((aligned ( 32)));
or
char data_array[512] __attribute__((aligned( 32)));
```

For buffers that are declared in automatic (stack) storage, only up to a 16-byte alignment is assured. Therefore, use dynamically allocated aligned static (global) storage instead.

6.3 Blue Gene/Q MPI extensions

This section describes extensions to the MPI library that is available on the Blue Gene/Q system. It contains information about the following topics:

- Functions to dynamically configure the algorithms used by the MPI collectives while the application is running, as described in 6.3.1, “Changing class-route usage at run time” on page 72
- Functions to determine specific information about the hardware being used by the job (such as torus coordinates for an MPI rank), as described in 6.3.2, “Determining hardware properties” on page 73

6.3.1 Changing class-route usage at run time

The five-dimensional (5D) torus requires class routes for collective operations. There are only 13 class routes available (MPI_COMM_WORLD consumes one of them), so that a node can only be in 13 communicators before hardware acceleration for collectives becomes unavailable.

The Blue Gene/Q MPI implementation allows developers to enable or disable the use of a class route by a given communicator. Used correctly, this feature can provide better application performance:

- int MPIX_Comm_update(MPI_Comm comm, int optimize);

The value 0 for *optimize* disables class route use on the communicator, *comm*. Any other value enables it. This call is collective. All nodes in the communicator must call this function.

The function returns the following values:

MPI_SUCCESS The property was successfully changed.

MPI_ERR_COMM The communicator is not valid.

Any other error code The optimization or deoptimization failed.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

6.3.2 Determining hardware properties

Several MPIX functions can be used to determine the hardware properties of the current node and job:

MPIX_Init_hw(MPIX_Hardware_t *hw)
: This function takes an MPIX_Hardware_t structure, as defined in mpix.h, and completes the fields. The hardware structure provides:
The physical rank irrespective of mapping
The size of the block irrespective of mapping
The number of processes per node
The core-thread ID of this process
The frequency of the processor clock
The size of the memory on the compute node
The number of torus dimensions
The size of each torus dimension
The torus coordinates of this process
A wrap-around link attribute for each torus dimension

int MPIX_Torus_ndims(int *numdimensions)
: This function returns the dimensionality of the torus (typically five on the Blue Gene/Q system).

int MPIX_Rank2torus(int rank, int *coords)
: This function returns the torus physical coordinates in the coords array for the MPI_COMM_WORLD rank passed in. The coords array needs to be predeclared and preallocated. It has the size numdimensions+1 (typically six on the Blue Gene/Q system).

int MPIX_Torus2rank(int *coords, int *rank)
: This function returns the MPI_COMM_WORLD rank for the passed in torus coordinates. The coords array needs to be of size numdimensions+1 (typically six on the Blue Gene/Q system).

6.4 MPI functions

This section lists several references that provide a comprehensive description of the MPI functions.

Appendix A in *Parallel Programming in C with MPI and OpenMP,* by Michael J. Quinn, describes all of the MPI functions, as defined in the MPI-1 standard. This reference also provides additional information and describes when to use each function.

In addition, you can find information about the MPI standard on the Message Passing Interface (MPI) standard website at:

`http://www.mcs.anl.gov/research/projects/mpi/`

A comprehensive list of the MPI functions is available on the MPI Routines web page at:

`http://www.mcs.anl.gov/research/projects/mpi/www/www3/`

The MPI Routines page includes MPI calls for C and Fortran. For more information, see the following books about MPI and MPI-2:

- *MPI: The Complete Reference, 2nd Edition, Volume 1*, by Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker, and Jack Dongarra

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

- *MPI: The Complete Reference, Volume 2: The MPI-2 Extensions*, by William Gropp, Steven Huss-Lederman, Andrew Lumsdaine, Ewing Lusk, Bill Nitzberg, William Saphir, and Marc Snir

Teaching MPI is beyond the scope of this book. See the following web page for tutorials and extensive information about MPI:

http://www.mcs.anl.gov/research/projects/mpi/learning.html

6.5 Compiling MPI programs on the Blue Gene/Q system

The Blue Gene/Q software provides scripts to compile and link MPI programs. These scripts simplify building MPI programs by setting the include paths for the compiler and linking in the libraries that implement MPICH2, the common Blue Gene/Q message layer interface (PAMI), and the low-level hardware interfaces (MUSPI) that are required by Blue Gene/Q MPI programs.

There are six versions of the libraries and the scripts:

gcc A version of the libraries that was compiled with the GNU Compiler Collection (GCC) and uses fine-grained locking in MPICH. These libraries also have error checking and assertions enabled.

gcc.legacy A version of the libraries that was compiled with the GNU Compiler Collection and uses a coarse-grain lock in MPICH. These libraries also have error checking and assertions enabled and can provide slightly better latency in single-thread codes, such as those that do not call MPI_Init_thread(... MPI_THREAD_MULTIPLE ...). Use one of the gcc libraries for initial application porting work.

xl A version of the libraries with MPICH compiled with the XL compilers and PAMI compiled with the GNU compilers. This version has the fine-grained MPICH locking and all error checking and asserts enabled. These libraries can provide a performance improvement over the gcc libraries.

xl.legacy A version of the libraries with MPICH compiled with the XL compilers and PAMI compiled with the GNU compilers. This version has the coarse-grained MPICH lock and all error checking and assertions are enabled. These libraries can provide a performance improvement over the gcc.legacy libraries for single-threaded applications.

xl.ndebug A version of the libraries with MPICH compiled with the XL compilers and PAMI compiled with the GNU compilers. This version has the fine-grained MPICH locking. Error checking and assertions are disabled. This setting can provide a substantial performance improvement when an application functions satisfactorily. Do not use this library version for initial porting and application development.

xl.legacy.ndebug A version of the libraries with MPICH compiled with the XL compilers and PAMI compiled with the GNU compilers. This version has the coarse-grained MPICH lock. Error checking and asserts are disabled. This can provide a substantial performance improvement when an application functions satisfactorily. Do not use this library version for porting and application development. This library version can provide a performance improvement over the xl.ndebug library version for single-threaded applications.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The various library versions are installed in `/bgsys/drivers/ppcfloor/comm`. There is a bin directory in each path (for example, `/bgsys/drivers/ppcfloor/comm/xl/bin`).

The bin directory contains scripts that either use the GNU compilers to build the application (mpicc, mpif77, and so on), or the XL compilers to build the application (mpixlf77, mpixlc) and then link with the MPICH and support libraries.

The following scripts are provided to compile and link MPI programs:

mpicc	C compiler
mpicxx	C++ compiler
mpif77	Fortran 77 compiler
mpif90	Fortran 90 compiler
mpixlc	IBM XL C compiler
mpixlc_r	Thread-safe version of mpixlc
mpixlcxx	IBM XL C++ compiler
mpixlcxx_r	Thread-safe version of mpixlcxx
mpixlf2003	IBM XL Fortran 2003 compiler
mpixlf2003_r	Thread-safe version of mpixlf2003
mpixlf77	IBM XL Fortran 77 compiler
mpixlf77_r	Thread-safe version of mpixlf77
mpixlf90	IBM XL Fortran 90 compiler
mpixlf90_r	Thread-safe version of mpixlf90
mpixlf95	IBM XL Fortran 95 compiler
mpixlf95_r	Thread-safe version of mpixlf95
mpich2version	FEN executable that prints MPICH2 version information

The following environment variables can be set to override the compilers used by the scripts:

MPICH_CC	C compiler
MPICH_CXX	C++ compiler
MPICH_FC	Fortran 77 compiler

The IBM XL Fortran 90 compiler is incompatible with the Fortran 90 MPI bindings in the MPICH library built with GCC. Therefore, the GNU versions of the mpixlf90 scripts cannot be used with the Fortran 90 MPI bindings.

Example 6-9 shows MPI wrapper script examples.

Example 6-9 Compiling with the MPI wrapper scripts provided in the Blue Gene/Q driver

```
$ /bgsys/drivers/ppcfloor/comm/gcc/bin/mpicc -o hello hello.c
$ /bgsys/drivers/ppcfloor/comm/xl/bin/mpixlc -o hello hello.c
$ /bgsys/drivers/ppcfloor/comm/gcc/bin/mpif77 -o hello hello.f
$ /bgsys/drivers/ppcfloor/comm/xl/bin/mpixlC -o hello hello.C
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Example 6-10 shows how to use the mpixlf77 script in a makefile.

Example 6-10 Using the mpixlf77 MPI script

```
XL         = /bgsys/drivers/ppcfloor/comm/xl/bin/mpixlf77

EXE         =  fhello
OBJ         =  hello.o
SRC         =  hello.f
FLAGS       =  -O3

$(EXE): $(OBJ)
        ${XL} $(FLAGS) -o $@ $^

$(OBJ): $(SRC)
        ${XL} $(FLAGS) -c $<

clean:
        $(RM) $(OBJ) $(EXE)
```

To build MPI programs for the Blue Gene/Q system, the compilers can be run directly instead of using the provided MPI compiler scripts. When running the compilers directly, you must explicitly include the required MPI libraries.

Example 6-11 shows a makefile that does not use the scripts. Replace (`library name`) with one of the six library types (gcc, gcc.legacy, xl, xl.legacy, xl.ndebug, xl.legacy.ndebug). This script assumes that the IBM XL compilers are installed in the default location `opt/ibmcmp`. If the compilers are installed in another location, the path in the examples must be also changed to match the alternative location.

Example 6-11 Makefile with explicit reference to libraries and include files

```
BGQ_FLOOR   = /bgsys/drivers/ppcfloor
BGQ_IDIRS   = -I$(BGQ_FLOOR)/comm/(library name)/include \
BGQ_LIBS    = -L$(BGQ_FLOOR)/comm/(library name)/lib -lmpich -lmpl -lopa \
              -L/bgsys/drivers/ppcfloor/comm/sys/lib -lpami \
              -L/bgsys/drivers/ppcfloor/spi/lib -lSPI_cnk -lrt -lstdc++ -lpthread

XL          = /opt/ibmcmp/xlf/bg/14.1/bin/bgxlf

EXE         =  fhello
OBJ         =  hello.o
SRC         =  hello.f
FLAGS       =  -O3 $(BGQ_IDIRS)

$(EXE): $(OBJ)
        ${XL} $(FLAGS) -o $@ $^ $(BGQ_LIBS)

$(OBJ): $(SRC)
        ${XL} $(FLAGS) -c $<

clean:
        $(RM) $(OBJ) $(EXE)
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

6.6 OpenMP

The OpenMP API is supported on the Blue Gene/Q system for shared-memory parallel programming in C/C++ and Fortran. This API is jointly defined by a group of hardware and software vendors and evolved as a standard for shared-memory parallel programming.

OpenMP comprises a collection of compiler directives and a library of functions that can be used in OpenMP programs. This combination provides a simple interface for developing parallel programs on shared-memory architectures. Multithreading is enabled on the Blue Gene/Q system. The OpenMP API provides access to data parallelism and functional parallelism.

For additional information, see the official OpenMP website at:

http://www.openmp.org/

6.6.1 OpenMP implementation for Blue Gene/Q

The Blue Gene/Q system supports shared-memory parallelism on single nodes. OpenMP is supported in the IBM extensible language (XL) compilers and the GNU GCC compilers. When using either the XL compilers or the GNU compilers, OpenMP can be used with MPI.

The IBM XL compilers provide support for OpenMP v3.1. The GNU compilers provide support for OpenMP v3.0.

See the corresponding compiler documentation for information about how to use OpenMP.

6.7 Multiple Program, Multiple Data

Multiple program, multiple data (MPMD) jobs are jobs for which a different executable and arguments can be supplied for a single job. All tasks of the job share the same MPICOMMWORLD communicator and can share data between different executables using the torus.

To enable MPMD support, specify a mapping file with the **runjob** --mapping option. Within the mapping file, there are keywords that control MPMD behavior on the nodes.

```
#mpmdbegin {ranks}
#mpmdcmd <executable> <arg0> <arg1> ... <argn>
#mpmdend
```

{ranks} specifies the MPI rank numbers. Multiple MPI ranks can be specified with a comma, for example:

```
#mpmdbegin 3,6,9
```

It is also possible to specify ranges of MPI ranks using a dash, for example:

```
#mpmdbegin 0-15
```

Additionally, ranges can be specified with a stride 'x' option, for example:

```
#mpmdbegin 0-15x2
```

Ranks 0, 2, 4, 6, 8, 10, 12, and 14 are included.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Sets and ranges can also be mixed:

#mpmdbegin 0,2,5-15

Avoid oversubscribing a rank to multiple programs.

There is also a shortcut option for specifying a calculated mapping without specifying each rank in the map file. To use that option:

#mapping ABCDET

All permutations of ABCDET are permitted.

There is also a restriction on MPMD ranks. All ranks in the same node must have the same program.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

7

Developing applications with Blue Gene/Q compilers

Applications to be run on the Blue Gene/Q system must be compiled and linked with a compiler that targets the Blue Gene/Q environment. Because compilation occurs on the front end node and not on the Blue Gene/Q system, these compilers are cross compilers.

This chapter describes the considerations for developing, compiling, and optimizing C/C++ and Fortran applications for the IBM Blue Gene/Q PowerPC A2 processor and the quad-processing extension (QPX) in the PowerPC AS v2 floating-point unit. This chapter contains information about the following topics:

- Programming environment overview
- Compilers for the Blue Gene/Q system
- Compiling and linking applications on the Blue Gene/Q system
- Compiler options specific to the Blue Gene/Q system
- Support for pthreads and OpenMP
- Creating libraries on the Blue Gene/Q system
- Running dynamically linked applications on the Blue Gene/Q system
- Mathematical Acceleration Subsystem Libraries
- Engineering and Scientific Subroutine Libraries
- Cross-compilation on the Blue Gene/Q system
- Python support
- Using the QPX floating-point unit

© Copyright IBM Corp. 2013. All rights reserved.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

7.1 Programming environment overview

Figure 2-1 on page 10 shows the system calls that the CNK manages, including forwarding I/O to the I/O node kernel. Figure 6-1 on page 66 shows a summary of the messaging software stack that supports the execution of Blue Gene/Q applications.

7.2 Compilers for the Blue Gene/Q system

The Blue Gene/Q system includes support for the IBM extensible library (XL) family of optimizing compilers. It also supports the GNU Compiler Collection, a Python interpreter, and the GNU toolchain tools.

7.2.1 IBM XL compilers

The Blue Gene/Q system includes the IBM XL compilers. These compilers can be used to develop C, C++, and Fortran applications for the IBM Blue Gene/Q system. This family comprises the following products, which are referred to in this chapter as the *IBM XL compilers for Blue Gene/Q*:

- XL C/C++ Advanced Edition V12.1 for Blue Gene/Q
- XL Fortran Advanced Edition V14.1 for Blue Gene/Q

The information presented in this chapter is an overview of the features that are available for use with the Blue Gene/Q system. For complete documentation about these compilers, see the information at the following websites:

- XL C/C++

  http://www.ibm.com/software/awdtools/xlcpp/library/

- XL Fortran

  http://www.ibm.com/software/awdtools/fortran/xlfortran/library/

Documentation is also typically included as PDF files in the installation directories under `/opt/ibmcmp`.

The default installation directory for the IBM XL compilers is `/opt/ibmcmp`. The system administrator can specify another installation directory. See the compiler documentation for more information about changing the default installation directories for the XL compilers.

If an alternative installation location is used for the XL compiler, create a link that refers to the alternative installation location in the `/opt/ibmcmp` directory, for example:

```
ln -s /bgsys/xlcompilers/latest /opt/ibmcmp
```

When this link is created, you can run the compiler with /opt/ibmcmp, but use the compilers that are installed in the alternative locations.

The examples in this chapter are based on the default installation location of `/opt/ibmcmp`. If another installation location is used, the path in the examples must be also changed to match the alternative location.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

7.2.2 GNU Compiler Collection

The standard GNU Compiler Collection V4.4.6 for C, C++, and Fortran is supported on the Blue Gene/Q system. The versions of the toolchain components are:

- gcc 4.4.6
- binutils 2.21.1
- glibc 2.12.2
- gdb 7.2

For more information about the toolchain compilers, see the man pages or the GNU website:

http://gcc.gnu.org

7.2.3 Python interpreter

You can install patches to build a version of Python that runs on the Blue Gene/Q system. See Section 7.11, “Python support” on page 95 for more information.

7.2.4 Toolchain tools

The GNU toolchain provides a variety of tools. These tools are in the `/bgsys/drivers/ppcfloor/gnu-linux/bin` directory and have the prefix powerpc64-bgq-linux-. Some tools have function added for the Blue Gene/Q system:

gdb	Contains support for remote debugging, the display of the Blue Gene/Q instruction set, and the display of QPX register contents.
objdump	Disassembles instructions on the Blue Gene/Q system.
readelf	Recognizes and displays the note section for the Blue Gene/Q system.
nm	Recognizes the vector4double data type for the XL compiler

7.3 Compiling and linking applications on the Blue Gene/Q system

The following Blue Gene/Q GNU compilers are stored at `/bgsys/drivers/ppcfloor/gnu-linux/bin`:

- **`powerpc64-bgq-linux-gcc`**
- **`powerpc64-bgq-linux-gfortran`**
- **`powerpc64-bgq-linux-g++`**

The names of the XL compilers for Blue Gene/Q are listed in Table 7-1. There are multiple variations for each language (C, C++, Fortran), depending on the language standard to be used. Use the thread-safe version of the compiler (the name ends in _r) to compile programs that run threads.

Table 7-1 Scripts available in the bin directory for compiling and linking

Language	Script name or names
C	bgc89, bgc99, bgcc, bgxlc bgc89_r, bgc99_r bgcc_r, bgxlc_r
C++	bgxlc++, bgxlc++_r, bgxlC, bgxlC_r

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Language	Script name or names
Fortran	bgf2003, bgf95, bgxlf2003, bgxlf90_r, bgxlf_r, bgf77, bgfort77, bgxlf2003_r, bgxlf95, bgf90, bgxlf, bgxlf90, bgxlf95_r

Example 7-1 shows how to compile and link a simple program.

Example 7-1 Linking and compiling a simple program

```
Compile and link a program with the toolchain:
$ /bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc64-bgq-linux-gcc -o hello hello.c
Compile and link a program with the XL C compiler:
$ /opt/ibmcmp/vacpp/bg/12.1/bin/bgxlc -o hello hello.c
Compile and link a program with the XL Fortran compiler:
/opt/ibmcmp/xlf/bg/14.1/bin/bgxlf90_r -o hello hello.f
Compile and link a program with the toolchain Fortran compiler:
/bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc64-bgq-linux-gfortran -o hello hello.f
```

7.4 Compiler options specific to the Blue Gene/Q system

Both the GNU compilers and the XL compilers have many options to configure compilation and linking. See the compiler documentation links for a complete list of options and descriptions. The options in this section are provided with the Blue Gene/Q versions of the compilers and are specific to programs that are compiled for the Blue Gene/Q system.

7.4.1 Options for the Blue Gene/Q system

This section presents the options for the Blue Gene/Q system.

GNU compilers

Specify the following option to use dynamic linking for the program:

-dynamic For performance reasons, the compilers for the Blue Gene/Q system default to static linking. If this option is not specified, static linking is used.

XL compilers

This section provides the XL compilers.

Default options

The following options are set when the bgxl compiler invocation scripts in the bin directory of the installation folder are used. These options are the default settings:

-qarch=qp -qtune=qp These options identify that the code is targeted for Blue Gene/Q.

-q64 The Blue Gene/Q compilers generate only 64-bit code.

-qsimd=auto Use this option to indicate whether the compiler transforms code into a form that can use the QPX floating-point instruction set. This process is sometimes referred to as simdizing the code. The -qsimd option defaults to auto. To disable the auto simdization of code, use the -qsimd=noauto option.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Additional Blue Gene/Q options

These options are specific to the Blue Gene/Q system. These options are not set by default:

-qnostaticlink Use this option to specify that the executable program is generated with dynamic linking. For all Blue Gene/Q XL compilers, the default linking mode is static.

-qmkshrobj Use this option to generate shared libraries when linking with the Blue Gene/Q XL compilers.

-qflttrap=qpxstore Use this option to generate code that permits floating-point exceptions to occur when the QPX floating-point unit is used. The QPX floating-point unit generates a limited set of floating-point exceptions. Only the QPX store instruction can generate a floating-point exception for not a number (NAN) or infinity (INF). This option is not enabled by default.

-qtm Use this option to process the transactional memory #pragmas in the program. When source code contains transactional memory #pragmas and is compiled with the -qtm option, the compiler generates code that uses the transactional memory on the Blue Gene/Q system. The #pragmas to identify the transactional code must be used in addition to the option -qtm for this feature to be enabled. Transactional memory is only useful when threads are present. A thread-safe compiler (that is, a compiler with _r in its name) must be used with this option. For more information about the syntax and use of transactional memory, see the links to the compiler documentation.

-qsmp=speculative Use this option to process the speculative thread #pragmas in the code. Both speculative thread constructs and this option are required to generate speculative threads. Use this option with a thread-safe compiler (that is a compiler with _r in its name). For more information about the syntax and use of speculative threads, see the links to the compiler documentation.

7.4.2 Unsupported compiler options

The following compiler options are not supported by the Blue Gene/Q compilers.

GNU compilers:

-m32 The Blue Gene/Q system supports only 64-bit architecture. The -m32 option is not supported.

XL compilers:

-q32 The Blue Gene/Q system uses a 64-bit architecture. The 32-bit mode is not supported.

-qaltivec The A2 processor does not support vector single instruction, multiple data (VMX) instructions.

7.5 Support for pthreads and OpenMP

Programs that use threads can be built with the Blue Gene/Q compilers and run on the Blue Gene/Q system. Threads on Blue Gene/Q system are implemented as pthreads that are defined by glibc in the Blue Gene/Q toolchain.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

OpenMP can also be used to create threads, which are implemented by the OpenMP run-time environment as pthreads. For more information about the threading model for the Blue Gene/Q system, see 3.9, “Threading overview” on page 17. For more information about the GNU OpenMP environment, which is also referred to as GOMP, see the GNU website at: http://gcc.gnu.org/projects/gomp/.

The IBM XL compilers for the Blue Gene/Q system support the OpenMP 3.1 standard. See the XL compiler documentation for information about the options that are required to enable OpenMP. This section also contains information about the source code changes for the OpenMP run time environment.

The GNU toolchain for Blue Gene/Q contains support for OpenMP 3.0.

7.5.1 Thread stack size for the Blue Gene/Q system

The thread stack size depends on the system configuration:

Minimum thread stack size

The minimum thread stack size is determined by glibc. In glibc 2.10, the value for PTHREAD_STACK_MIN is defined as 128 KB for PowerPC64. The smallest allowable stack in glibc 2.12.2 is also 128 KB. If the stack size is smaller than the minimum value, the pthread_create() function returns an error.

Default thread stack size

The default thread stack size on the Blue Gene/Q system is 4 MB.

Maximum thread stack size

The maximum size for a thread stack in a program depends on the amount of space that is available to allocate for stack space. There are many factors that can affect stack space. These factors include how much heap space is available to the process, how much heap space is already used, how many threads are being created, how much thread local storage is being used, and how many processes are running on the node. If memory errors occur when the pthread_create() function is called, there is probably not enough space to create the stack for the new thread. To set the thread stack size when creating a thread with pthread_create, use the pthread_setstacksize function as shown in Example 7-2.

Example 7-2 Using the pthread_setstacksize option to set the thread stack size

```
pthread_attr_t attr;
  pthread_t thd;
  int rc;
  pthread_attr_init(&attr);
  pthread_setstacksize(&attr, 8000000);
  rc = pthread_create(&thd, &attr, p, 0);
```

To set the thread stack size when using OpenMP threads, use the OMP_STACKSIZE environment variable. For example, you can enter the following command to set the OpenMP stack to 8 MB:

```
export OMP_STACKSIZE=8M
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

7.6 Creating libraries on the Blue Gene/Q system

On the Blue Gene/Q system, two types of libraries can be created:

- Static libraries
- Shared (dynamically loaded) libraries

When a program is statically linked, the required code from the static libraries is linked into the program. Example 7-3 illustrates how to create a static library on the Blue Gene/Q system with the XL family of compilers.

Example 7-3 Static library creation using the XL compilers

```
# Compile with the XL compiler
/opt/ibmcmp/vac/bg/12.1/bin/bgxlc -c pi.c
/opt/ibmcmp/vac/bg/12.1/bin/bgxlc -c main.c
#
# Create the library
/bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc64-bgq-linux-ar rcs libpi.a pi.o
#
# Create the executable program
/opt/ibmcmp/vac/bg/12.1/bin/bgxlc -o pi main.o -L. -lpi
```

Example 7-4 shows the same procedure with the GNU collection of compilers.

Example 7-4 Static library creation using the GNU compilers

```
# Compile with the GNU compiler
/bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc64-bgq-linux-gcc -c pi.c
/bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc64-bgq-linux-gcc -c main.c
#
# Create the library
/bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc64-bgq-linux-ar rcs libpi.a pi.o
#
# Create the executable program
/bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc64-bgq-linux-gcc -o pi main.o -L. -lpi
```

Shared libraries are loaded at execution time.

Use the -qnostaticlink option with the XL C and C++ compilers to build a dynamic binary. The static libgcc.a is linked in by default. To use the shared version of the libgcc library, also specify -qnostaticlink=libgcc. For example, use /opt/ibmcmp/vacpp/bg/12.1/bin/bgxlc -o hello hello.c -qnostaticlink -qnostaticlink=libgcc.

Example 7-5 shows shared library creation with the XL compiler.

Example 7-5 Shared library creation using the XL compiler

```
# Use XL to create shared library
/opt/ibmcmp/vac/bg/12,1/bin/bgxlc -qpic -c libpi.c
/opt/ibmcmp/vac/bg/12.1/bin/bgxlc -qpic -c main.c
#
# Create the shared library
/opt/ibmcmp/vac/bg/12.1/bin/bgxlc -qmkshrobj -Wl,-soname, libpi.so.0 -o libpi.so.0.0 libpi.o
#
# Set up the soname
ln -sf libpi.so.0.0 libpi.so.0
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
#
# Create a linker name
ln -sf libpi.so.0 libpi.so
#
# Create the executable program
/opt/ibmcmp/vac/bg/12.1/bin/bgxlc -o pi main.o -L. -lpi -qnostaticlink -qnostaticlink=libgcc
```

Example 7-6 illustrates the same procedure with the GNU collection of compilers.

Example 7-6 Shared library creation using the GNU compiler

```
# Compile with the GNU compiler
/bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc64-bgq-linux-gcc -fPIC -c libpi.c
/bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc64-bgq-linux-gcc -fPIC -c main.c
#
# Create shared library
/bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc64-bgq-linux-gcc -shared \
 -Wl,-soname,libpi.so.0 -o libpi.so.0.0 libpi.o -lc
#
# Set up the soname
ln -sf libpi.so.0.0 libpi.so.0
#
# Create a linker name
ln -sf libpi.so.0 libpi.so
#
# Create the executable program
/bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc64-bgq-linux-gcc -o pi main.o -L. -lpi -dynamic
```

The -qnostaticlink and -qmkshrobj options can also be used in a similar manner with the XL Fortran compilers.

7.7 Running dynamically linked applications on the Blue Gene/Q system

Unlike most other platforms, the compilers that generate code to run on the Blue Gene/Q system use static linking instead of dynamic linking by default. The use of static linking improves performance. If dynamic linking is used, there are some differences in how to build a program to run on the Blue Gene/Q system.

7.7.1 Creating a program

If no linking options are specified when linking a program, a statically linked program is generated. To use dynamic linking with GNU compilers, use the -dynamic option. Example 7-7 shows how to link a program that is to run with dynamic linking.

Example 7-7 Linking a program to be run with dynamic linking

```
/bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc64-bgq-linux-gcc -o hello hello.c -dynamic
/opt/ibmcmp/vacpp/bg/12.1/bin/bgxlc -o hello hello.c -qnostaticlink
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

A program that is created with a compiler that targets the Blue Gene/Q system identifies the path to the dynamic linker as `/bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc64-bgq-linux/lib/ld64.so.1`. This directory is available on the front end nodes and the I/O node. The readelf tool displays this information. For more information, see 7.7.5, “Tools for dynamic linking” on page 88.

7.7.2 Creating a shared library

When the GNU compilers are used, a shared library for the Blue Gene/Q system is created. This library is the same library as for Linux on IBM Power®. Compile the code that is included in a shared library with the pic option to identify that it contains position independent code. Example 7-8 provides syntax for creating a shared library with GNU compilers.

Example 7-8 Creating a shared library with GNU compilers

```
/bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc64-bgq-linux-gcc -c util.c -fpic
/bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc64-bgq-linux/gcc -o libtest.so util.o -shared
```

When the XL compilers are used, a shared library for the Blue Gene/Q system is created with the -qmkshrobj option. The -qmkshrobj option is supported for C, C++, and Fortran. Example 7-9 provides syntax for creating a shared library with XL compilers.

Example 7-9 Creating a shared library with XL compilers

```
/opt/ibmcmp/vacpp/bg/12.1/bin/bgxlc -c util.o -qpic
/opt/ibmcmp/vacpp/bg/12.1/bin/bgxlc -o libtest.so util.o -qmkshrobj
```

Important: Do not use the **ld** command to explicitly link a program. Use the Blue Gene/Q compilers (GNU and XL) to run the link command. This method ensures that the compiler links in dependent libraries that might be missed if the **ld** command is used alone.

7.7.3 Running a Blue Gene/Q dynamically linked program on a front end node

Some dynamically linked programs that are built for the Blue Gene/Q system run on the front end node because it is a PowerPC64 processor. This configuration is not supported.

To run programs in this configuration, explicitly invoke the dynamic linker, and use the Blue Gene/Q program as an argument.

Example 7-10 Invoking the dynamic linker

```
/bgsys/drivers/ppcfloor/gnu-linux/powerpc64-bgq-linux/lib/ld64.so.1 ./hello
```

7.7.4 Running a dynamically linked program on the Blue Gene/Q system

Running a dynamically linked program on the Blue Gene/Q system is similar to the way a statically linked program is run. When dynamically linked programs are run on the Blue Gene/Q system, the paths to those libraries must be known to the Blue Gene/Q dynamic linker, or the program fails to run. As described in 7.7.1, “Creating a program” on page 86, the information about the path to the Blue Gene/Q dynamic linker is embedded in the program when it is linked. By default, the dynamic linker searches the directories that are expected to contain shared libraries for use with Blue Gene/Q system. These locations include the directories for the Blue Gene/Q toolchain shared libraries and the Python shared library if the Python library is installed. The linker also searches for Message Passing Interface (MPI) or

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Parallel Active Messaging Interface (PAMI) shared libraries for the Blue Gene/Q driver. The driver is in the `/usr/lib64/bgq/` directory on the I/O node.

The Blue Gene/Q dynamic linker follows the same search conventions as the native GNU dynamic linker. At program load time, the dynamic linker attempts to load all of the dependent libraries in the program that are identified as NEEDED in the dynamic section of the program Executable and Linkable Format (ELF) file, as displayed by the **`readelf`** tool. The search path order for a dynamically linked program on the Blue Gene/Q compute node contains the following locations:

- The path in the DT_RPATH dynamic section of the program, if it exists
- The path identified by the LD_LIBRARY_PATH environment variable that is specified on the **`runjob`** command
- The path in the DT_RUNPATH dynamic section of the program, if it exists
- The information provided in /etc/ld.so.bgq.cache file on the I/O node
- The default paths searched by the Blue Gene/Q dynamic linker, which include `/lib64/bgq` and `/usr/lib64/bgq` on the I/O node
- The default native paths `/lib64` and `/usr/lib64`

The NEEDED, DT_RPATH, and DT_RUNPATH information for a program can be viewed using the **`readelf`** utility. For more information, see Section 7.7.5, “Tools for dynamic linking” on page 88.

7.7.5 Tools for dynamic linking

The tools that are described in this section provide information that can be used to determine how programs or shared libraries are built. It also provides information about the search paths that are used to find shared libraries.

The readelf tool

The readelf tool is in the toolchain. This tool provides information about the content of the dynamically linked program. It displays the path to the Blue Gene/Q dynamic linker, the set of shared library dependencies in the program (that are identified as NEEDED in the dynamic section of the program ELF file), and the DT_RPATH and DT_RUNPATH information. Figure 7-1 on page 89 displays this information.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  PHDR           0x0000000000000040 0x0000000001000040 0x0000000001000040
                 0x0000000000000188 0x0000000000000188  R E    8
  INTERP         0x00000000000001c8 0x00000000010001c8 0x00000000010001c8
                 0x0000000000000015 0x0000000000000015  R      1
      [Requesting program interpreter:
/bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc64-bgq-linux/lib/ld64.so.1]
  LOAD           0x0000000000000000 0x0000000001000000 0x0000000001000000
                 0x0000000000000824 0x0000000000000824  R E    10000
  LOAD           0x0000000000010000 0x0000000001100000 0x0000000001100000
                 0x0000000000000300 0x0000000000000370  RW     10000
  DYNAMIC        0x0000000000010028 0x0000000001100028 0x0000000001100028
                 0x0000000000000170 0x0000000000000170  RW     8
  NOTE           0x00000000000001e0 0x00000000010001e0 0x00000000010001e0
                 0x0000000000000040 0x0000000000000040  R      4
  GNU_EH_FRAME   0x00000000000007d8 0x00000000010007d8 0x00000000010007d8
                 0x0000000000000014 0x0000000000000014  R      4
...

Dynamic section at offset 0x10028 contains 19 entries:
  Tag        Type                         Name/Value
 0x0000000000000001 (NEEDED)             Shared library: [libc.so.6]
 0x000000000000000f (RPATH)              Library rpath: [/bgusr/boger]
```

Figure 7-1 Output from the readelf tool

The native readelf tool and the readelf tool in the Blue Gene/Q toolchain are similar. In some cases, the Blue Gene/Q tool provides additional information.

The ldd tool

In most cases, the native ldd tool in the /usr/bin/ldd directory does not provide the correct information for dynamically linked programs that are created to run on the Blue Gene/Q system. Run the ldd tool in the Blue Gene/Q toolchain from a front end node to find the shared library dependencies. Figure 7-2 shows output from the ldd tool when the Blue Gene/Q toolchain is run from a front end node.

```
/bgsys/drivers/ppcfloor/gnu-linux/powerpc64-bgq-linux/bin/ldd ./hello
linux-vdso64.so.1 => (0x00000fff9bb80000)
libc.so.6 => /bgsys/drivers/ppcfloor/gnu-linux/powerpc64-bgq-linux/lib/libc.so.6
(0x00000fff9b940000)
/bgsys/drivers/ppcfloor/gnu-linux/powerpc64-bgq-linux/lib/ld64.so.1 (0x0000000040450000)
```

Figure 7-2 Output from the ldd tool on a front end node

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Figure 7-3 shows output from the Blue Gene/Q ldd tool when it is run from a front end node.

```
/bgsys/drivers/ppcfloor/gnu-linux/powerpc64-bgq-linux/bin/ldd ./hello
linux-vdso64.so.1 => (0x00000fffa2720000)
libc.so.6 => /lib64/bgq/libc.so.6 (0x00000fffa24e0000)
/bgsys/drivers/ppcfloor/gnu-linux/powerpc64-bgq-linux/lib/ld64.so.1 (0x0000000040450000)
```

Figure 7-3 Output from the Blue Gene/Q ldd tool

The Blue Gene/Q toolchain shared libraries for a front end node are in the directories for the toolchain in /bgsys.

To run the equivalent of the ldd tool on the compute node, use the **runjob** command as shown in Figure 7-4. Long lines are separated with the backslash (\) character.

```
runjob --block R00-M0-N01 --corner R00-M0-N01-J00 --shape 1x1x1x1x1 \
  --cwd /bgusr/boger/bgq/c --envs LD_TRACE_LOADED_OBJECTS=1 --exe hello
        libc.so.6 => /lib64/bgq/libc.so.6 (0x0000001e01503000)
        /bgsys/drivers/ppcfloor/gnu-linux/powerpc64-bgq-linux/lib/ld64.so.1 => ld
(0x0000003001000000)
```

Figure 7-4 Running ldd on the compute node

LD_DEBUG tracing

To trace the search paths that are used by the dynamic linker, use the LD_DEBUG environment variable. The Blue Gene/Q variable has the same values as the Linux on Power variable. To see all of the supported values for the LD_DEBUG variable, run tracing with the LD_DEBUG=help: option. See Example 7-11.

Example 7-11 LD_DEBUG=help tracing

```
runjob --block R00-M0-N01 --corner R00-M0-N01-J00 \
  --shape 1x1x1x1x1 --raise --cwd /bgusr/boger/bgq/c --envs LD_DEBUG=help  --exe
./hello
Valid options for the LD_DEBUG environment variable are:

libs        display library search paths
reloc       display relocation processing
files       display progress for input file
symbols     display symbol table processing
bindings    display information about symbol binding
versions    display version dependencies
dbgevents   display debug events
all         all previous options combined
statistics  display relocation statistics
unused      determined unused DSOs
help        display this help message and exit

To direct the debugging output into a file instead of standard output
a filename can be specified using the LD_DEBUG_OUTPUT environment variable.
```

To see information about the directories that were searched to find a particular shared library, use the LD_DEBUG=libs: option. See Example 7-12 on page 91.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Example 7-12 LD_DEBUG=libs debugging

```
runjob --block R00-M0-N01 --corner R00-M0-N01-J00 --shape 1x1x1x1x1 --raise \
 --cwd /bgusr/boger/bgq/c --envs LD_DEBUG=libs  --exe hello
         1:     find library=libc.so.6 [0]; searching
         1:      search path=/bgusr/boger/tls:/bgusr/boger (RPATH from file hello)
         1:       trying file=/bgusr/boger/tls/libc.so.6
         1:       trying file=/bgusr/boger/libc.so.6
         1:      search cache=/etc/ld.so.bgq.cache
         1:       trying file=/lib64/bgq/libc.so.6
         1:
         1:
         1:     calling init: /lib64/bgq/libc.so.6
         1:
         1:
         1:     initialize program: hello
         1:
         1:
         1:     transferring control: hello
         1:
Hello from pid: 1 start: 0x1
         1:
         1:     calling fini: hello [0]
         1:
         1:
         1:     calling fini: /lib64/bgq/libc.so.6 [0]
         1:
```

To display detailed information about the libraries that were loaded, including the start address and size, use the LD_DEBUG=files: option. See Example 7-13.

Example 7-13 LD_DEBUG=files debugging

```
runjob --block R00-M0-N01 --corner R00-M0-N01-J00 --shape 1x1x1x1x1 --raise \
 --cwd /bgusr/boger/bgq/c --envs LD_DEBUG=files  --exe hello
         1:     file=hello [0];  generating link map
         1:       dynamic: 0x0000000001100028  base: 0x0000000000000000   size:
0x0000000000100380
         1:         entry: 0x00000000011001b8  phdr: 0x0000000001000040  phnum: 7
         1:
         1:
         1:     file=libc.so.6 [0];  needed by hello [0]
         1:     file=libc.so.6 [0];  generating link map
         1:       dynamic: 0x0000001e01719e10  base: 0x0000001e01503000   size:
0x00000000002320f8
         1:         entry: 0x0000001e0171ad78  phdr: 0x0000001e01503040  phnum: 10
         1:
         1:
         1:     calling init: /lib64/bgq/libc.so.6
         1:
         1:
         1:     initialize program: hello
         1:
         1:
         1:     transferring control: hello
         1:
Hello from pid: 1 start: 0x1
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
1:
1:     calling fini: hello [0]
1:
1:
1:     calling fini: /lib64/bgq/libc.so.6 [0]
1:
```

When using LD_DEBUG on a multi-node block, use the --label option on the **runjob** command to display which output corresponds to which node.

7.8 Mathematical Acceleration Subsystem Libraries

The Mathematical Acceleration Subsystem (MASS) libraries are tuned mathematical intrinsic functions that are available in versions for the IBM AIX® and Linux operating systems, including the Blue Gene/Q system. The MASS libraries provide improved performance over the standard mathematical library routines, are thread-safe, and support compilations in C, C++, and Fortran applications. For more information about MASS, see the Mathematical Acceleration Subsystem webpage at:

http://www.ibm.com/software/awdtools/mass/index.html

The MASS libraries are included with the XL compiler collections for Blue Gene/Q, which are installed in the `/opt/ibmcmp` path.

7.9 Engineering and Scientific Subroutine Libraries

The Engineering and Scientific Subroutine (ESSL) libraries for Linux on Power support the Blue Gene/Q system. ESSL provides over 150 math subroutines that are tuned for performance on the Blue Gene/Q system and use ESSL version 5.1.1. For more information about ESSL, see the Engineering Scientific Subroutine Library and Parallel ESSL website at:

http://www.ibm.com/systems/software/essl/index.html

Important: When using IBM XL Fortran V14.1 for the Blue Gene/Q system, use ESSL V5.1.1. If incompatible versions of ESSL and Fortran are selected, the RPM installation fails with a dependency error message.

7.10 Cross-compilation on the Blue Gene/Q system

An I/O node that is used as a front end node is a cross-compilation environment. When building an application in a cross-compilation environment, build tools such as **configure** and **make** might not provide the same results as when building natively. The results depend on whether the tools and application are designed to build correctly in the variety of cross-compilation environments that are available. Problems can occur because the configure and make steps for the build of a tool or application compile and execute small code snippets. These snippets identify characteristics of the target platform as part of the build process. If these code snippets are compiled with a cross-compiler and executed on the build machine instead of the target machine, the program might fail to execute or produce results that do not reflect the target machine.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

See the following sections for information about minimizing unexpected results:

- 7.10.1, “Configuring and building on an I/O node used as a front end node” on page 93
- 7.10.2, “Using implicit program launching from a front end node” on page 93

7.10.1 Configuring and building on an I/O node used as a front end node

An I/O node used as a front end node uses the same hardware as a compute node but runs the Linux operating system instead of the Compute Node Kernel (CNK). If a program is coded to use instructions that are specific to the A2 processor, it runs on an I/O node that is used as a front end node. However, it does not work on a standard front end node. If an application is configured on an I/O node that is used as a front end node, the program can be compiled and run natively. This method prevents some problems that are related to cross-compiling. An I/O node that is used as a front end node can be used to compile and run a set of small snippets. This method often provides correct results. However, it does not work for the following types of programs because CNK support is required:

- Transactional memory
- Speculative execution
- Blue Gene/Q MPI
- System calls that are supported on CNK but not on the Linux operating system
- System calls that provide different results on the CNK than on the Linux operating system

Example 7-14 shows how to compile and run a simple program on an I/O node that is used as a front end node. In this case, the program is running directly on the I/O node.

Example 7-14 Compiling and running a simple program on an I/O node used as a front end node

```
$ /bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc64-bgq-linux-gcc -o hello hello.c
$ ./hello
Hello
```

7.10.2 Using implicit program launching from a front end node

The Blue Gene/Q toolchain includes implicit toolchain launching. Implicit launching means invoking a Blue Gene/Q program as if it were being run natively on a front end node or an I/O node used as a front end node. However, the **runjob** command for that program is run implicitly based on the appropriate settings for some environment variables. This capability is sometimes referred to as magic. To enable this function, the variables in Example 7-15 must be set.

Example 7-15 Using implicit program launching

```
$ export BG_PGM_LAUNCHER=yes
$ export RUNJOB_BLOCK=R00-M0-N00

To run on a single node the following are also needed:

$ export RUNJOB_SHAPE=1x1x1x1x1
$ export RUNJOB_CORNER=R00-M0-N00-J00
```

Use the corresponding **runjob** environment variables to configure additional **runjob** arguments for use on the implicit **runjob** execution. See the man pages for the **runjob** command for a complete list and description of the **runjob** arguments. To disable implicit program launching, unset the BG_PGM_LAUNCHER environment variable.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The **runjob** program does not include scheduling. If the program runs on a single node, you must specify the node on the block.

Ensure that the small programs that are built during the configuration of the package are built using the Blue Gene/Q compilers. Each configuration script is unique, so general instructions for how to force the compiler cannot be provided. However, Example 7-16 works for many packages.

Example 7-16 Example configuration script for Blue Gene/Q compilers

```
$ ./configure CC=/bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc64-bgq-linux-gcc
```

You can verify that the program is being run remotely in this situation by compiling a program with the Blue Gene/Q cross-compiler and executing the program on the front end node.

Example 7-17 shows a sample program.

Example 7-17 Program to verify the Blue Gene/Q implicit runjob invocation

```
#include <stdio.h>
#include <sys/utsname.h>

int main(int argc, char** argv)
{
  struct utsname uts;

  uname(&uts);
  printf("machine: %s\n", uts.machine);
  if (strcmp(uts.sysname, "Linux") == 0) {
    printf("We are on Linux!\n");
  }
  else {
    printf("We are NOT on Linux!\n");
  }
   return 0;
}
```

Example 7-18 shows how to compile and run this program.

Example 7-18 Compiling and running a program

```
Set up the environment variables:
$ export RUNJOB_BLOCK=R00-M0-N03
$ export RUNJOB_SHAPE=1x1x1x1x1
$ export RUNJOB_CORNER=R00-M0-N03-J12
$ export BG_PGM_LAUNCHER=yes

Compile the program:
$/bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc64-bgq-linux-gcc -o test-onbgq
test-onbgq.c

Run the program:
$ ./test-onbgq
machine: BGQ
We are NOT on Linux!
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

To verify that the program is launched with the **runjob** command on the compute node, set the RUNJOB_VERBOSE environment variable to a value that ensures verbose output. For more information, see the manual page for the **runjob** command.

The implicit launch support is compiled into Blue Gene/Q programs when the toolchain compilers or the XL compilers are used. A program can be created without this support by adding the Wl,-e_start_no_magic option when linking the program, as shown in Example 7-19. When this option is used, the program does not use an implicit **runjob** invocation. Instead, it runs the program natively on the front end node.

Example 7-19 Using implicit launch support

```
/bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc64-bgq-linux-gcc -o hello hello.c \
-Wl,-e_start_no_magic
```

7.11 Python support

The Python interpreter is often used for scientific applications. The Blue Gene/Q system includes RPMs for a patched version of the Python interpreter. Python patches are available for Python 2.6.7, 2.6.8, 2.7.3, 3.2.2, 3.2.3 and can be built with either the GNU or XL compilers.

The default Python installation path is `/bgsys/tools`, but the interpreter can be installed in another location. For example, you can build and install the Blue Gene/Q Python library into a more efficient file system such as the IBM General Parallel File System (GPFS).

The Python interpreter is a dynamically linked application. See Example 7-6 on page 86 for information about the Blue Gene/Q environment. For information about how to build the Python interpreter, see the README file in the `/bgsys/drivers/ppcfloor/tools/python` directory. To see the options for the build scripts, use the -h flag.

7.11.1 Using the Python interpreter in a cross-compiled environment

The Blue Gene/Q system is a cross-compiled environment. The Python interpreter for the Blue Gene/Q system is built to run on Blue Gene/Q hardware. When building an application that must use the Python interpreter as part of its build process on the front end node, use the hostpython executable program in the installation directory. The executable programs in `/bgsys/tools/Python-2.6/bin/python` and `/bgsys/tools/Python-2.6/bin/hostpython` are identical, except for the default search paths and the location of the default dynamic linker. Both programs generate the same Python compiled modules, but are used on different hosts when compiling modules.

Table 7-2 summarizes the Python executable programs that exist on a front end node and lists when they can be used.

Table 7-2 Python executable programs on a front end node

Path	Host when used for build	When used to run Python	Targets
/usr/bin/**python**	Front end node	Front end node	RHEL6.x on PPC64
/bgsys/tools/Python-2.6/bin/**python**	I/O node used as a front end node	Compute node, I/O node used as a front end node	Blue Gene/Q hardware

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Path	Host when used for build	When used to run Python	Targets
/bgsys/tools/Python-2.6/bin/**hostpython**	Front end node	Compute node, I/O node used as a front end node	Blue Gene/Q hardware

Here are some examples of how to run the Python interpreter:

- To run the native Python interpreter on the front end node, but not run it on the Blue Gene/Q system, use the native Python interpreter `/usr/bin/`**`python`**.
- To run the Python interpreter on the Blue Gene/Q compute node, use the **`runjob`** command to run the Python interpreter `/bgsys/tools/Python-2.6/bin/`**`python`**.
- To compile a Python module as part of an application, where that module is later used to run on the Blue Gene/Q system, run the **`hostpython`** executable program on the front end node to compile the module.
- To run the Python interpreter on an I/O node used as front end node, use the Python interpreter `/bgsys/tools/Python-2.6/bin/`**`python`**.

7.11.2 Running the Python interpreter on the Blue Gene/Q system

The Python interpreter can be built with either the XL or GNU compilers. When the Python interpreter is built and installed on the front end node as part of the build, a tar file called either bgqpython2.6.7gnu.tar.gz or bgqpython2.6.7XL.tar.gz is created and installed into `/bgsys/linux/bgfs` on the service node. This tar file is used at I/O node boot time to extract the files that are required at run time by the Python interpreter on the I/O node. At run time, the Python shared library is installed from the I/O node into `/usr/lib64/bgq` and the Python modules are installed into `/bgfs/usr/lib64`. The Python interpreter starts faster when the shared libraries and Python modules are accessed from these locations.

Example 7-20 shows how to verify the locations of these files.

Example 7-20 Verifying the locations of Python files

```
On the service node or front end node:
ls /bgsys/linux/bgfs
bgqpython2.6.7gnu.tar.gz bgqpython3.2.tar.gz
On an I/O node used as a front end node
ls /bgfs/usr/lib64
libpython2.6.so.1.0 libpython3.2.so.1.0 python2.6 libpython2.6.so libpython3.2.so python3.2
```

- The Blue Gene/Q Python interpreter is built and installed on the front end node. As part of this installation, the Python shared library and the python modules are packaged into a tar file for use when booting the I/O block where the Python interpreter is run. When the I/O block is booted, the Python shared library libpython2.6.so.x is in `/lib64/bgq/` and the Python modules are found in `/bgfs/usr/lib64/`.
- To compile applications, use the Python library that is on a front end node. In the Blue Gene/Q Python installation, there are two Python binaries: python and hostpython. The hostpython binary is designed to compile Python modules on the front end node. The **`python`** binary is designed to be used when running the Python interpreter on the compute node or an I/O node used as a front end node.

If the Python interpreter is installed in the default location in `/bgsys/tools`, the binaries are stored in the following locations:

- /bgsys/tools/Python-2.6/bin/python hello.py

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

- /bgsys/tools/Python-2.6/bin/hostpython hello.py

Example 7-21 shows how to run the Python interpreter on the Blue Gene/Q system.

Example 7-21 Running Python on the Blue Gene/Q system

```
$ runjob --block R00-M0-N00 : /bgsys/drivers/ppcfloor/gnu-linux/bin/python
testarray.py
```

For more information about the Python interpreter, see the following websites:

- Python Programming Language - Official Web site

  http://www.python.org/
- The Python Tutorial

  http://docs.python.org/tut/tut.html
- The Python Standard Library

  http://docs.python.org/lib/lib.html
- pyMPI: Putting the py in MPI

  http://pympi.sourceforge.net/

7.12 Using the QPX floating-point unit

The Blue Gene/Q hardware contains the quad-processing extension (QPX) to the IBM Power Instruction Set Architecture. The computational model of the QPX architecture is a vector single instruction, multiple data (SIMD) model with four execution slots and a register file that contains 32 registers with 256 bits. Each of the 32 registers contains four elements of 64 bits. Each of the execution slots operates on one vector element. These elements are referred to as vector registers or quad registers.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Figure 7-5 shows the quad floating-point unit.

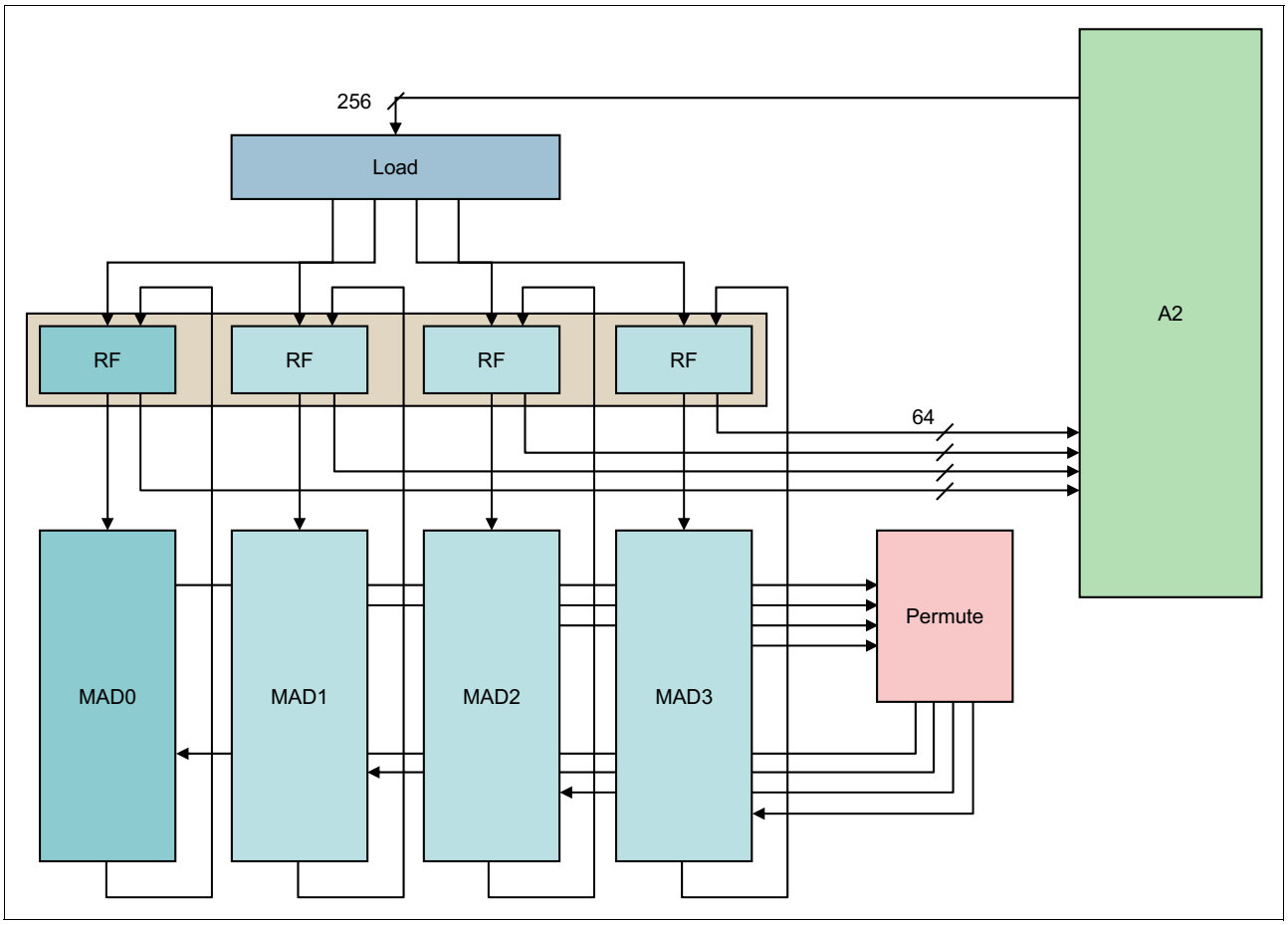


Figure 7-5 Blue Gene/Q quad floating-point unit

7.12.1 Using SIMD instructions in applications

There are two methods to use the QPX floating-point instruction set in your application when using the IBM XL compiler for Blue Gene/Q:

- Using automatic simdization
- Using vector intrinsics functions or assembly code

Using automatic simdization with the IBM XL compiler

If your program is compiled with the XL compiler for Blue Gene/Q, simdization is enabled by default. The compiler attempts to automatically transform code to efficiently use the QPX floating-point instruction set. Simdization is enabled at all optimization levels, but more aggressive and effective simdization occurs at higher optimization levels. Simdization is controlled by the -qsimd option and can be disabled by using the -qsimd=noauto option.

The -qreport option provides information about what code is simdized and why simdization does not occur.

For information about how to enable more simdization to occur, see the compiler documentation links in Section 7.2.1, “IBM XL compilers” on page 80.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Using the QPX vector intrinsics functions

Vector intrinsics functions are provided in the IBM XL compiler for Blue Gene/Q. The vector intrinsics are built-in functions that map directly onto the QPX instruction set. You can use these functions to tune code and enable the compiler to optimize the code as effectively as possible.

For a description of the set of vector intrinsics, see the compiler documentation links in Section 7.2.1, “IBM XL compilers” on page 80.

Example 7-22, Example 7-23, and Example 7-24 on page 100 show some common vector intrinsics functions for Blue Gene/Q. These examples are not optimized for performance, and are not comprehensive.

Example 7-22 shows basic quadword load, store, and arithmetic operations in C.

Example 7-22 C example with basic quadword load, store, and arithmetic operations

```
// vector version of y[i] = a*x[i] + y[i]
// where "x" and "y" are 32-byte aligned

#include <stdio.h>

#define NPTS 8
static double __attribute__((aligned(32))) x[NPTS], y[NPTS];

int main(int argc, char * argv[])
{
   int i;
   double a = 2.0;
   vector4double av, xv, yv;
   for (i=0; i<NPTS; i++) {
      x[i] = (double) i;
      y[i] = (double) i + 1;
   }
   if ((long) x & 0x1F ) printf("x is not 32-byte aligned\n");
   if ((long) y & 0x1F)  printf("y is not 32-byte aligned\n");
   av = vec_splats(a);            // replicate "a" in four vector slots
   for (i=0; i<NPTS; i+=4) {
      xv = vec_ld(0L, &x[i]);     // load four contiguous elements of x[]
      yv = vec_ld(0L, &y[i]);     // load four contiguous elements of y[]
      yv = vec_madd(av, xv, yv);  // yv = av*xv + yv
      vec_st(yv, 0L, &y[i]);      // store four contiguous elements of y[]
   }
   for (i=0; i<NPTS; i++) printf("y[%d] = %.1lf\n", i, y[i]);
   return 0;
}
```

Example 7-23 shows basic quad-word load, store, and arithmetic operations in Fortran.

Example 7-23 Fortran example with basic quadword load, store, and arithmetic operations

```
program fmain
  implicit none
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
  integer i
  integer, parameter :: n = 8
  real(8) a, x(n), y(n)
!IBM* align(32, x, y)
  vector(real(8)) av, xv, yv
  a = 2.0d0
  do i = 1, n
    x(i) = dble(i-1)
    y(i) = dble(i)
  end do
  if (iand(loc(x), z'1F') .ne. 0) print *, 'x is not 32-byte aligned'
  if (iand(loc(y), z'1F') .ne. 0) print *, 'y is not 32-byte aligned'
  av = vec_splats(a)            ! replicate "a" in four vector slots
  do i = 1, n, 4
     xv = vec_ld(0, x(i))       ! load four contiguous elements of x()
     yv = vec_ld(0, y(i))       ! load four contiguous elements of y()
     yv = vec_madd(av, xv, yv)  ! yv = av*xv + yv
     call vec_st(yv, 0, y(i))   ! store four contiguous elements of y()
  end do
  do i = 1, n
    write(*,'(a,i1,a,f4.1)') 'y(', i, ') = ', y(i)
  end do
end
```

The basic quadword load intrinsic, vec_ld(), returns a vector variable with values taken from the address argument rounded down to the nearest 32-byte boundary. Therefore, it is important to know the alignment of the variable that is used. The vec_lda() signaling variant of the vector load function generates an exception if the address is not 32-byte aligned. To handle arbitrary alignment, determine the alignment and use appropriate shift or permute instructions.

Alignment can be determined by a bitwise AND operation between the address and 0x1F = 31 (as a base-10 integer). Variables of type double can be on 8, 16, 24, or 32-byte boundaries. The shift and permute intrinsics are designed to take two vectors as input and return a vector with the required values, shifted or permuted from the input arguments. For example, if you have a target address that has 16-byte alignment, you can get a vector with four contiguous elements starting from that address by two vector loads followed by one shift. Example 7-28 shows this process in C. The third argument to the vector shift function, vec_sldw(), must be an integer constant, not an integer variable.

Similarly, the argument to the general permute control intrinsic, vec_gpci(), must be an integer constant. It is convenient to use an octal constant for that purpose because there is a one-to-one correspondence between the selected slots from the two vector arguments and the digits of the constant in octal format. For example, to select slots 3, 2, 1, and 0, you can code vec_gpci(03210) in C, or vec_gpci(o'3210') in Fortran. To maximize performance, use 32-byte aligned data.

Example 7-24 shows an example of vector and permute instructions in C.

Example 7-24 Vector shift and permute instructions in C

```
// example of vector shift and permute operations
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
#include <stdio.h>

#define NPTS 10
static double __attribute__((aligned(32))) x[NPTS];

int main(int argc, char * argv[])
{
   int i;
   vector4double v1, v2, xv, pctl;

   for (i=0; i<NPTS; i++) x[i] = (double) i;

   if (((long) &x[2] & 0x1F) != 16 ) printf("x[2] is not 16-byte aligned\n");

   v1 = vec_ld(0L, &x[2]);    // v1 has values x[0-3]
   v2 = vec_ld(0L, &x[6]);    // v2 has values x[4-7]

   xv = vec_sldw(v1, v2, 2);  // xv has values x[2-5]
   for (i=0; i<4; i++) printf("xv[%d] = %.1lf\n", i, xv[i]);

   printf("\n");

   pctl = vec_gpci(05432);    // xv has "x" values from slots 5,4,3,2
   xv = vec_perm(v1, v2, pctl);
   for (i=0; i<4; i++) printf("xv[%d] = %.1lf\n", i, xv[i]);

   return 0;
}
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

8

Running and debugging applications

This chapter explains how to run and debug applications on the Blue Gene/Q system.

The following topics are covered:

- Running applications
- Debugging applications
- What to do when a job fails
- Debugging jobs

© Copyright IBM Corp. 2013. All rights reserved.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

8.1 Running applications

Blue Gene/Q applications can be run in several ways. The most common method is to use a job scheduler that supports the Blue Gene/Q system, such as the LoadLeveler scheduler. All the Blue Gene/Q job schedulers use the runjob interface. The runjob interface is described in the *Blue Gene/Q System Administration* (SG24-7869) Redbooks publication.

8.1.1 IBM LoadLeveler

The IBM LoadLeveler product is intended to manage both serial and parallel jobs over a cluster of servers. This distributed environment consists of a pool of machines or servers, often referred to as a *LoadLeveler cluster*. Machines in the pool can be of several types: desktop workstations available for batch jobs (usually when not in use by their owner), dedicated servers, and parallel machines.

The LoadLeveler scheduler allocates machine resources in the cluster to run jobs. The scheduling of jobs depends on the availability of resources in the cluster and various rules, which can be defined by the LoadLeveler administrator. A user submits a job using a job command file. The LoadLeveler scheduler attempts to find resources within the cluster to satisfy the requirements of the job. The LoadLeveler scheduler maximizes the efficiency of the cluster by maximizing the use of resources, while minimizing the job turnaround time that is experienced by users.

The LoadLeveler scheduler provides a rich set of functions for job scheduling and cluster resource management. Some of the tasks that the LoadLeveler scheduler can perform include:

- Choosing the next job to run
- Examining the job requirements
- Collecting available resources in the cluster
- Choosing the best machines for the job
- Dispatching the job to the selected machine
- Controlling running jobs
- Creating reservations and scheduling jobs to run in the reservations
- Job preemption to enable high-priority jobs to run immediately
- Fair-share scheduling to automatically balance resources among users or groups of users
- Co-scheduling to enable several jobs to be scheduled to run at the same time
- Multicluster support to allow several LoadLeveler clusters to work together to run user jobs

See the LoadLeveler documentation for information about setting up and using the LoadLeveler scheduler with the Blue Gene/Q system. The documentation is available online at:

`http://publib.boulder.ibm.com/infocenter/clresctr/vxrx/index.jsp`

8.2 Debugging applications

This section describes the debuggers that are supported by the Blue Gene/Q system.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

8.2.1 General debugging architecture

Four pieces of code are involved when debugging applications on the Blue Gene/Q system:

- The Compute Node Kernel, which provides the low-level primitives that are required to debug applications
- The Tool Control daemon, which runs on the I/O nodes and provides control and communications to compute nodes
- A tool running on the I/O nodes, which is vendor-supplied code that interfaces with the Tool Control daemon
- A tool running on a front end node, which is where the user performs work interactively

A debugger communicates to the compute node through the Code Development and Tools Interface (CDTI) that is provided by the Tool Control daemon that runs on the I/O node. A debugger typically comprises both a tool server program running on the I/O nodes and a tool client program running on the front end node. Both the tool client and the tool server are provided by the debugger vendor. The tool client usually communicates to the tool server with the TCP/IP protocol. The tool server attaches to the Tool Control daemon using a connected local socket. For more information about the CDTI interface, see the *IBM System Blue Gene Solution: Blue Gene/Q Code Development and Tools Interface*, REDP-4659 IBM Redpapers™ publication.

8.2.2 GNU Project Debugger

The GNU Project Debugger (GDB) is the primary debugger of the GNU project. You can learn more about GDB on the web at the following address:

`http://www.gnu.org/software/gdb/gdb.html`

For more information about GDB, see the GDB web site:

`http://www.gnu.org/software/gdb/documentation/`

The Blue Gene/Q system includes support for running GDB with applications that run on compute nodes. IBM provides a simple debug server called *gdbserver*. Each running instance of GDB is associated with one process or rank.

When using GDB to debug a compute node process, the GDB tool must be run using the remote target interface. This section refers to GDB as the GDB client.

Blue Gene/Q gdbserver implementation

The **`gdbserver`** program implements a subset of the GDB remote protocol specification. Therefore, advanced features that might be available in other implementations are not available in this implementation.

Each instance of a GDB client can connect to and debug one process. To debug multiple processes at the same time, run multiple GDB tools at the same time. A maximum of four GDB tools can be run on one job.

The toolchain patches provide a set of patches to use to build against the supported version of the GDB tool. When the Blue Gene/Q toolchain is built, the GDB tool is in the toolchain installation directories.

To debug an application, the debug server must be started and running before you attempt to debug. Use an option on the **`runjob`** command or run the **`start_tool`** command with

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

appropriate arguments. For more information, see the examples in “Running the GNU Project Debugger tool” on page 106.

Prerequisite software

The GDB client software is typically installed during the installation procedure. To verify the installation, check whether the `/bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc64-bgq-linux-gdb` file exists on the front end node. The rest of the software support that is required for GDB is installed as part of the control programs.

When compiling a program for the Blue Gene/Q system, you can use options to simplify debugging.

Use the -g option on the compiler. This option tells the compiler to include symbol and source statement information with the program. This information can be used by the GDB tool to map storage to program data or instructions to source statements. The -g option and its variations are available with both the GNU and XL compilers. For more information about debugging options, see the compiler documentation. If code is not compiled with the -g option, only low-level debugging is available and minimal symbol information is provided.

In addition to using the -g option, it is best to compile with the lowest possible level of optimization when trying to debug an application. Optimization can change the way variables are used and stored, eliminate code sequences, and reorder instructions in a way that can be misleading for the debugger.

Because programs are run remotely on the Blue Gene/Q system, remote debugging must be used to debug programs running on the Blue Gene/Q compute node. To provide GDB support for the compute node, a GDB server, called gdbtool, must first be started for each rank to be debugged. The tool can be started either when the job is started by using an option on the **`runjob`** command, or by explicitly running the **`start_tool`** command to start the gdbtool server. When started with the **`runjob`** command, the gdbtool server is started when the program begins running. When started using the **`start_tool`** command, the gdbtool server is attached to a job that is already running. This method is useful when a job is hanging or not progressing. When started with the **`runjob`** command, a single instance of the gdbtool server is started on each I/O node that is associated with the job. The gdbtool server connects to the node that contains the target process. If multiple gdbtool sessions are to be used to debug a single process or additional processes in the same job, the additional instances must be started with the **`start_tool`** command. See the examples in Running the GNU Project Debugger tool.

Running the GNU Project Debugger tool

Perform these steps to use the GNU Project Debugger tool to debug a Blue Gene/Q application:

1. Compile the program with the -g option and no optimization:

   ```
   /bgsys/drivers/ppcfloor/comm/xl/bin/mpixlc -o helloMPI helloMPI.c -g -O0
   ```

2. Start the GDB tool.

   To start a gdbtool server when running a program, add the -start-tool option to the **`runjob`** command. The path is `/sbin/gdbtool`.

3. Find the IP address for the I/O node that corresponds to the rank to be debugged.

   The default value for the rank is 0, and the default value for the listen_port is 10000. Figure 8-1 on page 107 shows an example of starting the gdbtool server with the **`runjob`** command when this configuration is used.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
runjob --block R00-M0-N01 --cwd `pwd` --start-tool /sbin/gdbtool --exe
helloMPI --tool-args "--rank=4 --listen_port=10001"
tool started on 1 I/O nodes for 32 ranks.

Enter a rank to see its associated I/O node's IP address, or press enter to
start the job:
4
rank 4 uses I/O node R03-ID-J03 at IP address 172.20.215.28
```

Figure 8-1 Finding an IP address for rank 0 and listen_port 10000

A GDB client session is started. The GDB client for Blue Gene/Q is in the same directory as the toolchain. Figure 8-2 shows an example of the output when the session is started.

```
/bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc64-bgq-linux-gdb ./helloMPI
GNU gdb (GDB) 7.1
…
(gdb) target remote 172.20.215.28:10001
Remote debugging using 172.20.215.28:10001
```

Figure 8-2 Example output for starting a GDB client session

4. When this line is displayed, push enter in the **runjob** session. Figure 8-3 shows the output.

```
[Switching to Thread 1]
0x00000000010001d0 in ._start ()
(gdb)
```

Figure 8-3 Output that indicates that GDB commands can be entered

At this point, GDB commands can be entered to debug the program.

5. Optional: To debug another rank in the same process, use the **start_tool** command to start another session to attach to the same process:
   a. Use the **list_jobs** command to find the job ID for the process. Figure 8-4 shows an example command.

```
/bgsys/drivers/ppcfloor/bin/list_jobs --user boger
1 job
    ID Status      Executable  Block       User
 55469 Running     helloMPI    R03-M0-N12  boger
```

Figure 8-4 Using the list_jobs command to find the job ID for a process

   b. Use the **dump_proctable** command to find the IP address that is associated with the rank to be debugged. Figure 8-4 shows an example command.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
/bgsys/drivers/ppcfloor/bin/dump_proctable --id 55469 --rank 21

Rank     I/O node IP address                    pid
21       172.20.215.26                          0x01001001
```

Figure 8-5 Using the dump_proctable command to find an IP address that is associated with the rank to be debugged

c. Run the **start_tool** command to start another gdbtool server. For example, use the command in Figure 8-6 to connect to rank 21 of job 55469, and specify another listen_port.

```
/bgsys/drivers/ppcfloor/bin/start_tool --tool /sbin/gdbtool --args
"--rank=21 --listen_port=10002" --id 55469
tool 2 started on 2 I/O nodes for 32 ranks.
```

Figure 8-6 Running the start_tool command to start another gdbtool server

d. Start another **gdb** client session to connect to the **gdbtool** session. Figure 8-7 shows a command to debug rank 21.

```
/bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc64-bgq-linux-gdb
./helloMPI
…
(gdb) target remote 172.20.215.26:10002
Remote debugging using 172.20.215.26:10002
```

Figure 8-7 Example command to debug rank 21

The GDB tool displays where it is stopped in the program and provides a prompt for entering GDB client commands.

The **tool_status** command can be used to provide additional information when using the **start_tool** command. This command provides status information about tools that have been started with the **start_tool** command. It can be helpful if there are connection problems or other issues during the debugging process. Figure 8-8 shows an example of the **tool_status** command and its output.

```
/bgsys/drivers/ppcfloor/bin/tool_status --id 16047700
2 tools
Id  Path           Status   Message  Start Time
1   /sbin/gdbtool  Running           2012-Feb-28 09:00:24.463421
2   /sbin/gdbtool  Running           2012-Feb-28 09:03:14.977132
```

Figure 8-8 Example tool_status command and output

8.2.3 Coreprocessor debugger

The Coreprocessor debugger can be used to debug problems at all levels (hardware, kernel, and application). See *IBM System Blue Gene Solution: Blue Gene/Q System Administration*, SG24-7869 for more information.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

8.2.4 The addr2line utility

When a Blue Gene/Q program runs unsuccessfully, a core file is generated. You can use the addr2line utility to analyze the core file. This utility uses the debugging information in an executable program to provide information about the file name and line number for the source that was used to create the program. The addr2line program is a standard Linux utility. For more information, see the Linux manual page entry for the **addr2line** command.

Creating a file to use with the addr2line utility

Perform the following steps to create a file that can be used with the addr2line utility:

1. Compile the program with the -g option. The -g switch tells the compiler to include debugging information in the executable program.
2. Run the program. The program generates a core file. The core file is a plain text file that can be viewed with the vi editor. The information in this file must be reformatted before it can be used with the **addr2line** command.

   Example 8-1 shows the default format of the core file.

Example 8-1 Core file output format

```
+++STACK
Frame Address     Saved Link Reg
0000001fffff5ac0  000000000000001c
0000001fffff5bc0  00000000018b2678
0000001fffff5c60  00000000015046d0
0000001fffff5d00  00000000015738a8
0000001fffff5e00  00000000015734ec
0000001fffff5f00  000000000151a4d4
0000001fffff6000  00000000015001c8
0000001fffff6120  00000000014f7cec
0000001fffff6220  0000000001531974
0000001fffff6320  0000000001426d00
0000001fffff63e0  0000000001001730
0000001fffff6d80  00000000010040c8
0000001fffff6e20  000000000103ad0c
0000001fffffba20  000000000117f6e0
0000001fffffbaa0  000000000187ed78
0000001fffffbd80  000000000187f074
0000001fffffbe40  0000000000000000
---STACK
```

3. Delete all of the information except the addresses in the Saved Link Reg column, as shown in Example 8-2.

Example 8-2 Addresses in the Saved Link Reg column

```
000000000000001c
00000000018b2678
00000000015046d0
00000000015738a8
00000000015734ec
000000000151a4d4
00000000015001c8
00000000014f7cec
0000000001531974
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
0000000001426d00
0000000001001730
00000000010040c8
000000000103ad0c
000000000117f6e0
000000000187ed78
000000000187f074
0000000000000000
```

4. Replace the first eight 0s with 0x, as shown in Example 8-3.

Example 8-3 Information for use with the addr2line tool

```
0x0000001c
0x018b2678
0x015046d0
0x015738a8
0x015734ec
0x0151a4d4
0x015001c8
0x014f7cec
0x01531974
0x01426d00
0x01001730
0x010040c8
0x0103ad0c
0x0117f6e0
0x0187ed78
0x0187f074
0x00000000
```

You can also use a script to reformat the output. Example 8-4 shows a Perl script that reads a core file, core.X, and outputs addresses in the correct format for the addr2line program.

Example 8-4 Perl script, bgqtranslate.pl, to reformat the core file for use with the addr2line utility

```
#!/usr/bin/perl -w

#
# open the core.X file for reading
#
open (CF, " < " . $ARGV[0])  || die "usage:  ./bgqtranslate.pl corefile";
#
# Create the necessary extension for each task in the core.X file
#
$extension=".t";
$task_num=0;

$line = <CF>;
$in_stack = 0;

#
# Loop to handle the core.X file, converting the necessary address
# lines into the separate task files that have the correct format
# for use with the addr2line -e executable command
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
#
while (! eof(CF))
{
   chomp ($line);
   if ($line eq "+++STACK")
   {
      $in_stack = 1;
      $line = <CF>;
      open (TF, " > " . $ARGV[0] . $extension . $task_num);
   }
   elsif ($line eq "---STACK")
   {
      $in_stack = 0;
      close (TF);
      $task_num++;
   }
   elsif ($in_stack == 1)
   {
      @addresses = split (/[ \t]+/, $line);
      printf (TF "0x%s\n", substr($addresses[1],8));
   }
   $line = <CF>;
}

close (CF);
```

Running the addr2line utility

Follow the steps in “Creating a file to use with the addr2line utility” on page 109 to reformat the core file before running the addr2line utility.

Use the Linux addr2line command on the front end node and enter the address found in the core file and the -g executable. The utility displays the source line where the problem occurred.

Figure 8-9 on page 112 shows how to use the addr2line utility with a reformatted core file to identify potential problems in the code. In this case, the program is not compiled with the -g flag option because this example is a production run. Compiling without the -g option produces smaller executable code that performs better.

However, notice that the addr2line output points to the malloc() function. This information means that the amount of memory might be insufficient to run this calculation. It might also indicate problems that are related to the use of the malloc() function in the code.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
$ ./bgqtranslate.pl core.0
$ addr2line -e ramses3d.wat < core.0.t0
/bgsys/drivers/DRV2012_0118_0003/ppc64-rhel60/toolchain/gnu/glibc-2.12.2/stdlib/abort.c:77
/bgsys/drivers/DRV2012_0118_0003/ppc64-rhel60/toolchain/gnu/glibc-2.12.2/libio/../sysdeps/unix/sysv/
linux/libc_fatal.c:186
/bgsys/drivers/DRV2012_0118_0003/ppc64-rhel60/toolchain/gnu/glibc-2.12.2/malloc/malloc.c:6323
/bgsys/drivers/DRV2012_0118_0003/ppc64-rhel60/toolchain/gnu/glibc-2.12.2/malloc/malloc.c:3777
/bgusr/smithbr/xmi/new/bgq/comm/lib/dev/mpich2/src/mpi/romio/adio/common/malloc.c:107
/bgusr/smithbr/xmi/new/bgq/comm/lib/dev/mpich2/src/mpi/romio/adio/ad_bg/ad_bg_aggrs.c:266
/bgusr/smithbr/xmi/new/bgq/comm/lib/dev/mpich2/src/mpi/romio/adio/ad_bg/ad_bg_aggrs.c:297
/bgusr/smithbr/xmi/new/bgq/comm/lib/dev/mpich2/src/mpi/romio/adio/ad_bg/ad_bg_hints.c:511
/bgusr/smithbr/xmi/new/bgq/comm/lib/dev/mpich2/src/mpi/romio/adio/common/ad_open.c:86
/bgusr/smithbr/xmi/new/bgq/comm/lib/dev/mpich2/src/mpi/romio/mpi-io/open.c:148
/bgusr/smithbr/xmi/new/bgq/comm/lib/dev/mpich2/src/binding/f77/file_openf.c:201
/bghome/heymanj/RAMSES/TelQuel/Src/bin/../pario/hints.f90:211
/bghome/heymanj/RAMSES/TelQuel/Src/bin/../pario/hints.f90:53
/bghome/heymanj/RAMSES/TelQuel/Src/bin/../amr/read_params.f90:111
/bghome/heymanj/RAMSES/TelQuel/Src/bin/../amr/ramses.f90:18
/bgsys/drivers/DRV2012_0118_0003/ppc64-rhel60/toolchain/gnu/glibc-2.12.2/csu/../csu/libc-start.c:226
/bgsys/drivers/DRV2012_0118_0003/ppc64-rhel60/toolchain/gnu/glibc-2.12.2/csu/../sysdeps/unix/sysv/linux/
powerpc/libc-start.c:194
??:0
```

Figure 8-9 Using the addr2line utility to identify potential problems in the code

8.3 What to do when a job fails

After a job starts, if any rank in the job terminates normally with exit(1), a SIGTERM signal is sent to the remaining ranks. If any rank in the job terminates abnormally because of a signal, the SIGKILL signal is sent to the remaining ranks. The **`locate_rank`** command can be used to correlate a node location (for example, R00-M1-N12-J22) to an MPI rank on a per-job basis.

If a job terminates incorrectly, the error message is presented to the user with the standard error (stderr) stream. Figure 8-10 shows an error that occurs when more memory is allocated than is available.

```
"module_hydro_principal.f90", line 31: 1525-108 Error encountered while
attempting to allocate a data object. The program will stop.
2012-02-20 14:11:41.383 (WARN ) [0xfff84778a40]
R01-M1-N04-64:432467:ibm.runjob.client.Job: normal termination with status 1
from rank 19
```

Figure 8-10 Example output for job termination because of a memory allocation error

Because there is normal termination, no core file (core.X) is generated.

If abnormal termination occurs, the system generates multiple core files. Figure 8-10 shows an example of system output when core files are created. Figure 8-11 on page 113 shows example output when a job fails and core files are created.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
2012-02-01 16:28:34.587 (WARN ) [0xfff8b9289e0]
R01-M1-N00-256:232446:ibm.runjob.client.Job: terminated by signal 11
2012-02-01 16:28:34.588 (WARN ) [0xfff8b9289e0]
R01-M1-N00-256:232446:ibm.runjob.client.Job: abnormal termination by signal 11
from rank 0
```

Figure 8-11 Example output when a job fails and core files are created

The information in the +++STACK/---STACK address stack section of the file can be read with the addr2line utility. See 8.2.4, “The addr2line utility” on page 109.

8.4 Debugging jobs

The snapbug and Coreprocessor tools can be used to debug jobs.

8.4.1 The snapbug tool

The snapug tool can be run only by the system administrator. This tool can be used to collect debugging information from blocks, hung jobs, or jobs that terminate abnormally. It is intended for first-failure data capture for failures on the Blue Gene/Q system.

The snapbug parameters are either a booted block name (compute or I/O block) or a running job ID. The tool uses this information to identify all hardware resources that are associated with that block (that is, if the compute block is specified, the tool finds jobs and the connected I/O block). It extracts information from those resources.

The following examples show the format for running the snapbug tool:

```
/bgsys/drivers/ppcfloor/scripts/snapbug.pl  -block=<blockname>   [-output=<outputdir>]
/bgsys/drivers/ppcfloor/scripts/snapbug.pl  -jobid=<jobid>       [-output=<outputdir>]
```

The snapbug tool provides the following information:

- Reliability, availability, and serviceability (RAS) information from the block since boot
- Information about all jobs that ran on the block since boot (including executables and lightweight core files)
- A Coreprocessor snapshot of I/O blocks and compute blocks
- Flight recorders (CNK and Linux)
- Service node information (driver version, memory usage, bg.properties)
- Control System logs since boot
- I/O node logs
- Block location information

8.4.2 The Coreprocessor tool

The Coreprocessor tool can also be used to debug a job. To specify a job, specify the process ID (PID) of the runjob process. Users can specify either the Blue Gene job ID or the process ID (PID) of the runjob process. To specify the job ID, users can use the -j parameter. To specify the PID, users can use the -pid option. The PID can be specified on the command line with the --pid option or added with the File menu in the Blue Gene Navigator interface. See

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

the *IBM System Blue Gene Solution: Blue Gene/Q System Administration*, SG24-7869 Redbooks publication for more information about the Coreprocessor tool.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

A

Mapping

This appendix provides an overview of the mapping of tasks for the Blue Gene/Q system.

© Copyright IBM Corp. 2013. All rights reserved.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Mapping overview

Mapping is defined as the assignment of processes to Blue Gene processors. The term MPI rank denotes a process, but the concept applies whether MPI or some other communication protocol is used. The network topology for Blue Gene/Q is a five-dimensional (5D) torus or mesh, with direct links between the nearest neighbors in the ±A, ±B, ±C, ±D, and ±E directions. When communication involves the nearest neighbors on the torus network, a large fraction of the theoretical peak bandwidth can be obtained. However, when MPI ranks communicate with many hops between the neighbors, the effective bandwidth is reduced by a factor that is equal to the average number of hops that messages take on the torus network. In a number of cases, it is possible to control the placement of MPI ranks so that communication remains local. This placement can significantly improve scaling for a number of applications, particularly at large processor counts.

The default mapping is to place MPI ranks on the system in ABCDET order where the rightmost letter increments first, and where <A,B,C,D,E> are torus coordinates and T is the processor ID in each node (T = 0 to N -1, where N is the number of processes per node being used). If the job uses the default mapping and specifies one process per node, the following assignment results:

- MPI rank 0 is assigned to coordinates <0,0,0,0,0,0>.
- MPI rank 1 is assigned to coordinates <0,0,0,0,1,0>.
- MPI rank 2 is assigned to coordinates <0,0,0,1,0,0>.

The results continue like this, first incrementing the E coordinate, then the D coordinate, and so on, until all of the processes are mapped. Note that the E dimension is always 2 on the Blue Gene/Q system.

The default mapping is often a good choice. The mapping can also be controlled by passing arguments to the **`runjob`** command, or alternatively by constructing a specially ordered communicator in the application. If a batch scheduling system is used, the **`runjob`** command cannot be used directly, but the batch utility might contain a similar feature.

Mapping might not be important for jobs that use one midplane (512 nodes) or less of the Blue Gene/Q system due to the compact shape, <A,B,C,D,E> = <4,4,4,4,2> for a midplane, and the high degree of connectivity. Mapping can be particularly useful for applications that have a regular Cartesian topology, and are dominated by nearest-neighbor boundary exchange, particularly for large configurations. For use with a batch job scheduler, mapping requires information about the shape of the block, not just the number of nodes, and special key words might be required to define the shape.

The **runjob** command for the Blue Gene/Q system includes two methods to specify the mapping. For example, you can add --mapping TEDCBA to request TEDCBA order, where A increments first. All permutations of ABCDET are permitted. You can also create a customized map file, and use --mapping my.map, where my.map is the name of the map file. Using a customized map file provides the most flexibility. The syntax for the map file is simple. It must contain one line for each MPI rank in the Blue Gene/Q block, with six integers on each line separated by spaces. The six integers specify the <A,B,C,D,E,T> coordinates for each MPI rank. The first line in the map file assigns MPI rank 0, the second line assigns MPI rank 1, and so on. It is important to ensure that the map file is consistent, with a unique relationship between MPI rank and <A,B,C,D,E,T> location. The “T” coordinate in the map file ranges from 0 to N - 1, where N is the number of ranks per node.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

General guidance

For applications that use a regular Cartesian topology, it is often possible to map MPI ranks onto the Blue Gene/Q torus network in a way that preserves locality for nearest-neighbor communication. For example, in a one-dimensional processor topology, where each MPI rank communicates with its rank ±1, the default ABCDET mapping can be good for blocks large enough to use torus wrap-around. The “T” coordinate increments first. When using many ranks per node, most of the communication is within the node, using shared memory. On the Blue Gene/Q system, the E-dimension is always two, and is always torus-enabled. The other dimensions become torus-enabled when their size reaches 4 or a multiple of 4. With torus wrap-around, the ABCDET order keeps one-dimensional communication local, except for one extra hop at the torus edges. The extra hop can be eliminated by using an ordering that makes a snake-like pattern through the torus by reversing direction after completing each one-dimensional sweep.

Two-dimensional (2D) logical topologies are more challenging. In many cases, the default ordering can be effective. For example, suppose that you are using a Blue Gene/Q midplane with 16 processes per node. The corresponding coordinates have sizes <4,4,4,4,2,16> for <A,B,C,D,E,T>. When two or three of these dimensions are grouped together, this ordering is naturally a good fit for logical decompositions 256 × 32 and 64 × 128, where the last dimension increments first. For a 128 × 64 decomposition, TEDCBA order, where A increments first, is a better choice. These mappings have one extra hop at torus boundaries, and a customized map file can be used to create a better mapping. For example, with 16 MPI ranks per node, the 16 processes are arranged in a logical 4 × 4 box (or 2 × 8, 8 × 2, and so on). This configuration is useful because most of the boundary exchange can be kept inside each node, using shared memory. To complete the mapping, group two or three of the torus dimensions into the logical X or Y dimension. This grouping creates a snake-like pattern that winds through the torus and reverses direction at the end of each row. This technique can ensure that shared-memory is used for much of the communication, and that the maximum number of hops on the torus network is one. This configuration minimizes link sharing and optimizes the bandwidth available for boundary exchange. You can use this strategy to construct a map file that is passed to the **runjob** command or to the batch system. Alternatively, you can keep the default mapping and use the same strategy to order the ranks in a new communicator that is an input parameter for the MPI_Cart_create() function. In that case, set the reorder parameter for the MPI_Cart_create() function to false, so that the ordering is preserved. In the first software release, the MPI_Cart_create() function is not fully aware of the topology on the Blue Gene/Q system. When the MPI_Cart_create() function is called with the reorder parameter set to true, it typically does not provide an optimal mapping. If you use a customized map file with the MPI_Cart_create() function, set the reorder parameter to false to preserve the ordering in the map file. Pseudo-code to generate a 2D map file is sketched in Example A-1 on page 118. The code to construct an ordered communicator is similar. Each process defines its logical X and Y coordinates and its rank in the 2D communicator. Then, the MPI_Comm_split() function can be used to create a reordered communicator with the rank in the 2D communicator as the key argument.

Three-dimensional (3D) Cartesian topologies occur in many simulations, and a similar strategy can be used. Again, the default mapping can be effective in a number of cases. For a midplane with 16 processes per node, the <A,B,C,D,E,T> dimensions are <4,4,4,4,2,16>. When one or more dimensions are grouped together, the default mapping can be a good match for logical 3D decompositions such as 16×32×16 and 16×16×32, where the last dimension increments first. There is one extra hop at torus boundaries, which can be avoided by constructing a customized map file, or equivalently, a specially ordered communicator. Link-sharing is a bandwidth consideration, so this optimization tends to be less important for exchange of short messages, where latency can be dominant. Consider simple permutations of ABCDET, and logical decompositions that can map naturally onto groups of those

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

coordinates, before exploring map files or reordered communicators. MPI performance tools can provide some guidance for the potential impact of mapping. If contention for link bandwidth is a major problem, performance can be optimized by controlling the layout.

Example A-1 shows pseudo-code for creating a map file for 2D Cartesian topologies.

Example A-1 Pseudo-code for creating a map file for 2D Cartesian topologies

```
for each of the five torus dimensions A,B,C,D,E {
   define 2D node coordinates (iX, iY) and sizes nX, nY
   where iX uses two of the torus dimensions and iY uses the other three
   define a local box within each node with dimensions bX*bY = ranks-per-node

   for each process within the node {
       define local coordinates : tX = 0 … bX-1,  tY = 0 … bY-1
       define 2D logical coordinates : pX = tX + bX*iX
                                       pY = tY + bY*iY
       rank in the 2D communicator : rank2D = pY + pX*(bY*nY)
       save torus information for each rank
   }
}

sort in order of increasing rank2D, saving the index array

print the map file using the ordering from the sort
```

A portion of a map file is shown in Example A-2. This map file is generated for a midplane partition (shape ABCDE = 4 × 4 × 4 × 4 × 2) with 16 ranks per node, for a total of 8192 ranks arranged in a logical 128 × 64 two-dimensional process grid. The layout in each node is a box with dimensions 4 × 4. When communication is nearest-neighbor on the two-dimensional process grid, this layout keeps most of the communication within each node, and keeps communication on the torus network local. Although custom map files can be useful, nearly optimal performance can often be obtained by simpler methods. In this example, the default mapping, ABCDET, is nearly ideal for a 64 × 128 two-dimensional process grid.

Example A-2 Map file for a 2D Cartesian layout with 128 x 4 processes

```
#A  B  C  D  E  T   # 2d : <X,Y>
 0  0  0  0  0  0   # 2d : <0,0>
 0  0  0  0  0  1   # 2d : <0,1>
 0  0  0  0  0  2   # 2d : <0,2>
 0  0  0  0  0  3   # 2d : <0,3>
 0  0  0  1  0  0   # 2d : <0,4>
 0  0  0  1  0  1   # 2d : <0,5>
 0  0  0  1  0  2   # 2d : <0,6>
 0  0  0  1  0  3   # 2d : <0,7>
 …
```

Quantum chromodynamics (QCD) applications typically use a four-dimensional (4D) process topology. This topology can often fit perfectly onto the Blue Gene/Q system. However, a map file or an ordered communicator might be required. For a midplane of a Blue Gene/Q system that uses 32 ranks per node, the <A,B,C,D,E,T> dimensions are <4,4,4,4,2,32>. If you want to use a roughly balanced 4D decomposition, such as (X,Y,Z,T) = (8,8,16,16), there is no natural grouping of ABCDET that fits. In this case, it is ideal to consider the 32 MPI ranks within each node as being arranged in a 4D box; for example: bx = 2,by = 2,bz = 4,bt = 2. Then one logical 4D mapping can be described as (bx*A,by*B,bz*C,bt*DE), with many equivalent

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

solutions, and where the DE is treated as a one-dimensional line that snakes through the DE plane to avoid the extra hop at torus boundaries. It is straightforward to use this strategy to construct either a customized map file that is passed to the **runjob** command or the batch system, or an ordered communicator that is used for boundary exchange in the application. For the same configuration, one midplane with 32 ranks per node, there are a number of other logical 4D decompositions that map naturally with the default ABCDET mapping, such as (16,16,2,32), (4,16,8,32), and permutations. Ideally, it is preferable to use the best logical decomposition and a perfect mapping to solve the problem. Again, these considerations apply mainly to large configurations, where link bandwidth, not latency, is the key factor.

For unstructured grids, the default ordering is frequently a good choice. This happens because neighboring cells tend to end up in the same part of the machine, and ABCDET order keeps many contiguous ranks in close proximity. MPI performance tools can be used to check for locality, but optimization of the layout in the general case of unstructured grids is a challenging problem.

To summarize, applications, particularly applications that use a regular Cartesian topology, might benefit from a careful mapping of processes onto processors. The default mapping is frequently a good choice. However, it might be possible to optimize performance by using a map file passed to the **runjob** command or to the batch system. It might also be possible to optimize performance by constructing a specially ordered communicator for use within the application.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

B

Blue Gene/Q personality

System calls that provide access to certain hardware or system features can be accessed by applications. This appendix illustrates how to obtain hardware-related information.

© Copyright IBM Corp. 2013. All rights reserved.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Personality of Blue Gene/Q nodes

The personality of a Blue Gene/Q node is the static data given to every compute node and I/O node at boot time by the Control System. This data contains information that is specific to the node, with respect to the block that is being booted.

The personality is a set of C language structures that contains such items as the node coordinates on the torus network. This information can be useful if the application programmer wants to determine, at run time, where the tasks of the application are running. It can also be used to tune certain aspects of the application at run time. For example, it can be used to determine which set of tasks shares the same I/O node and then optimize the network traffic from the compute nodes to that I/O node.

Examples of retrieving Blue Gene/Q personality information

Example B-1 illustrates how to invoke and print selected hardware features.

Example B-1 personality.c program

```
#include <stdio.h>
#include <mpi.h>
#include <spi/include/kernel/location.h>

int main(int argc, char * argv[])
{
   uint64_t Nflags;
   char procname[128];
   Personality_t pers;
   int rank, procid, core, hwthread, namelen;
   int Anodes, Bnodes, Cnodes, Dnodes, Enodes;
   int Acoord, Bcoord, Ccoord, Dcoord, Ecoord;
   int Atorus, Btorus, Ctorus, Dtorus, Etorus;
   MPI_Init(&argc, &argv);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   MPI_Get_processor_name(procname, &namelen);
   procid   = Kernel_ProcessorID();       // 0-63
   core     = Kernel_ProcessorCoreID();   // 0-15
   hwthread = Kernel_ProcessorThreadID(); // 0-3
   Kernel_GetPersonality(&pers, sizeof(pers));
   Anodes = pers.Network_Config.Anodes; Acoord = pers.Network_Config.Acoord;
   Bnodes = pers.Network_Config.Bnodes; Bcoord = pers.Network_Config.Bcoord;
   Cnodes = pers.Network_Config.Cnodes; Ccoord = pers.Network_Config.Ccoord;
   Dnodes = pers.Network_Config.Dnodes; Dcoord = pers.Network_Config.Dcoord;
   Enodes = pers.Network_Config.Enodes; Ecoord = pers.Network_Config.Ecoord;
   Nflags = pers.Network_Config.NetFlags;
   if (Nflags & ND_ENABLE_TORUS_DIM_A) Atorus = 1; else  Atorus = 0;
   if (Nflags & ND_ENABLE_TORUS_DIM_B) Btorus = 1; else  Btorus = 0;
   if (Nflags & ND_ENABLE_TORUS_DIM_C) Ctorus = 1; else  Ctorus = 0;
   if (Nflags & ND_ENABLE_TORUS_DIM_D) Dtorus = 1; else  Dtorus = 0;
   if (Nflags & ND_ENABLE_TORUS_DIM_E) Etorus = 1; else  Etorus = 0;
   if (rank == 0) {
      printf("block shape     : <%d,%d,%d,%d,%d>\n",
                                  Anodes,Bnodes,Cnodes,Dnodes,Enodes);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
      printf("torus links enabled : <%d,%d,%d,%d,%d>\n",
                                    Atorus,Btorus,Ctorus,Dtorus,Etorus);
   }
   printf("rank %d has processor name %s\n", rank, procname);
   printf("rank %d location <%d,%d,%d,%d,%d> core %d hwthread %d procid = %d\n",
                 rank,Acoord,Bcoord,Ccoord,Dcoord,Ecoord,core,hwthread,procid);
   MPI_Finalize();
   return 0;
}
```

The following command is an example command that is used to build a personality test program from personality.c with the GNU compiler for Blue Gene/Q:

```
/bgsys/drivers/ppcfloor/comm/gcc/bin/mpicc personality.c -o personality
```

Example B-2 shows part of the output that is generated by running the personality program with the default (ABCDET) ordering using a 128-node block and two ranks per node. See Appendix A, “Mapping” on page 115. The output is sorted by MPI rank for readability, in practice; however, output is not ordered.

Example B-2 Output with default (ABCDET) order, a 128-node block, and two ranks per node

```
$ runjob --block R02-M1-N00-128  --ranks-per-node 2 --np 256 --env_all --cwd $PWD : personality

block shape     : <2,2,4,4,2>
torus links enabled : <0,0,1,1,1>
rank 0 has processor name Task 0 of 256 (0,0,0,0,0,0)  R02-M1-N00-J00
rank 0 location <0,0,0,0,0> core 0 hwthread 0 procid = 0
rank 1 has processor name Task 1 of 256 (0,0,0,0,0,1)  R02-M1-N00-J00
rank 1 location <0,0,0,0,0> core 8 hwthread 0 procid = 32
rank 2 has processor name Task 2 of 256 (0,0,0,0,1,0)  R02-M1-N00-J07
rank 2 location <0,0,0,0,1> core 0 hwthread 0 procid = 0
rank 3 has processor name Task 3 of 256 (0,0,0,0,1,1)  R02-M1-N00-J07
rank 3 location <0,0,0,0,1> core 8 hwthread 0 procid = 32
rank 4 has processor name Task 4 of 256 (0,0,0,1,0,0)  R02-M1-N00-J12
rank 4 location <0,0,0,1,0> core 0 hwthread 0 procid = 0
...
```

Example B-3 shows output from the personality program with TEDCBA mapping. The output is sorted by MPI rank for readability.

Example B-3 Output with TEDCBA order, a 128-node block, and two ranks per node

```
$ runjob --block R02-M1-N00-128 --ranks-per-node 2 --np 256 --mapping TEDCBA --env_all --cwd
$PWD : personality

block shape     : <2,2,4,4,2>
torus links enabled : <0,0,1,1,1>
rank 0 has processor name Task 0 of 256 (0,0,0,0,0,0)  R02-M1-N00-J00
rank 0 location <0,0,0,0,0> core 0 hwthread 0 procid = 0
rank 1 has processor name Task 1 of 256 (1,0,0,0,0,0)  R02-M1-N00-J29
rank 1 location <1,0,0,0,0> core 0 hwthread 0 procid = 0
rank 2 has processor name Task 2 of 256 (0,1,0,0,0,0)  R02-M1-N00-J03
rank 2 location <0,1,0,0,0> core 0 hwthread 0 procid = 0
rank 3 has processor name Task 3 of 256 (1,1,0,0,0,0)  R02-M1-N00-J30
rank 3 location <1,1,0,0,0> core 0 hwthread 0 procid = 0
rank 4 has processor name Task 4 of 256 (0,0,1,0,0,0)  R02-M1-N00-J01
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
rank 4 location <0,0,1,0,0> core 0 hwthread 0 procid = 0
…
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

C

PAMI and MPI header files and libraries

This appendix provides information about the Parallel Active Messaging Interface (PAMI) and Message Passing Interface (MPI) header files and libraries for the Blue Gene/Q system.

© Copyright IBM Corp. 2013. All rights reserved.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Blue Gene/Q applications

Blue Gene/Q applications run on the Blue Gene/Q compute nodes and can use PAMI or MPI software to communicate between application processes on different compute nodes.

There are six different builds of the MPICH2 software. Each build contains identical header files. Table C-1 describes the header files in the following directories:

- `/bgsys/drivers/ppcfloor/comm/gcc/include`
- `/bgsys/drivers/ppcfloor/comm/gcc.legacy/include`
- `/bgsys/drivers/ppcfloor/comm/xl/include`
- `/bgsys/drivers/ppcfloor/comm/xl.legacy/include`
- `/bgsys/drivers/ppcfloor/comm/xl.ndebug/include`
- `/bgsys/drivers/ppcfloor/comm/xl.legacy.ndebug/include`

Table C-1 MPI header files

File name	Description
mpicxx.h	Message Passing Interface (MPI) C++ interface
mpif.h	MPI Fortran interface
mpi.h	MPI C interface
mpiof.h	MPI I/O Fortran interface
mpio.h	MPI I/O C interface
mpix.h	Blue Gene/Q extensions to the MPI specifications
mpi.mod,mpi_base.mod, mpi_constants.mod, mpi_sizeofs.mod	Fortran 90 bindings
opa_config.h, opa_primitives.h, opa_queue.h, opa_util.h	OpenPA headers used by MPICH2

Table C-2 describes the PAMI header files in the `/bgsys/drivers/ppcfloor/comm/sys/include/ and /bgsys/drivers/ppcfloor/comm/sys-fast/include/` directories.

Table C-2 PAMI header files in the comm/sys/include and comm/sys-fast/include directories

File name	Description
pami.h	Common Blue Gene/Q message layer interface
pami_sys.h	Common Blue Gene/Q message layer interface platform definitions

Table C-3 on page 127 describes the static and dynamic MPI libraries in the following directories:

- `/bgsys/drivers/ppcfloor/comm/gcc/lib`
- `/bgsys/drivers/ppcfloor/comm/gcc.legacy/lib`
- `/bgsys/drivers/ppcfloor/comm/xl/lib`
- `/bgsys/drivers/ppcfloor/comm/xl.legacy/lib`
- `/bgsys/drivers/ppcfloor/comm/xl.ndebug/lib`
- `/bgsys/drivers/ppcfloor/comm/xl.legacy.ndebug/lib`

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Table C-3 Static and dynamic libraries in the comm/gcc and comm/xl directories

File name	Description
libmpich.a, libmpic.so	C bindings for MPICH2
libcxxmpich.a libcxxmpich.so	C++ bindings for MPICH2
libfmpich.a libfmpich.so	Fortran bindings for MPICH2
libfmpich_.cnk.a	Fortran bindings for MPICH2 with extra underscoring; XL only
libmpich.f90.a libmpich.f90.so	Fortran 90 bindings
libopa.a	OpenPA library used by MPICH2
libtvmpich2.so	TotalView library for MPICH2 queue debugging

Table C-4 describes the static and dynamic PAMI libraries in the `/bgsys/drivers/ppcfloor/comm/sys/lib` and `/bgsys/drivers/ppcfloor/comm/sys-fast/lib` directories.

Table C-4 Static and dynamic libraries in the comm/sys/lib and comm/sys-fast/lib directories

File name	Description
libpami.a, libpami.so	Common Blue Gene/Q message library

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

D

MPI and CNK environment variables

This appendix describes the environment variables that affect the run time characteristics of programs that run on the Blue Gene/Q compute nodes. These variables configure settings for the Message Passing Interface (MPI) and the Compute Node Kernel (CNK).

Environment variables can be used to improve performance or modify functional attributes of the application.

The following topics are covered:

- Message Passing Interface environment variables
- Compute Node Kernel environment variables
- Setting environment variables

© Copyright IBM Corp. 2013. All rights reserved.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Message Passing Interface environment variables

The Blue Gene/Q Message Passing Interface (MPI) implementation provides several environment variables that affect its behavior. Setting these environment variables can allow a program to run faster, or, if set incorrectly, might cause the program not to run at all. None of these environment variables are required to be set for the Blue Gene/Q MPI implementation to work.

Table D-1 shows the MPI environment variables.

Table D-1 MPI environment variables

Environment variable	Description	Default value
COMMAGENT_ RGETPACINGMAX	The maximum number of bytes allowed to be in the network at one time as a result of paced remote gets from each node. This number must be a multiple of COMMAGENT_RGETPACINGSUBSIZE. The default value depends on the number of nodes in the block, as shown in Table D-2.	See Table D-2
COMMAGENT_ RGETPACINGSUBSIZE	The size, in bytes, of a submessage used for remote get pacing. The pacing logic breaks a large remote get into submessages of this size. Table D-3 shows the default values for the COMMAGENT_RGETPACINGSUBSIZE environment variable. These values vary depending on the size of the block where the job is being run.	See Table D-3
MUSPI_INJFIFOSIZE	The size, in bytes, of each injection first-in, first-out queue (FIFO). These FIFOs store 64-byte descriptors. Each descriptor describes a memory buffer to be sent on the torus. Making this size larger might reduce memory usage and latency when there are many outstanding messages. Reducing this size might increase that memory usage and latency. PAMI messaging optimally uses 10 injection FIFOs per context, although fewer FIFOs can be used when resources are constrained.	65536 (64 KB)
MUSPI_NUMBATIDS	The number of base address table IDs per process reserved for use by a messaging unit (MU) SPI application.	0
MUSPI_NUMCLASSROUTES	The number of collective class routes reserved for use by an MU SPI application. This value is also the number of global interrupt class routes reserved for use by an MU SPI application.	0
MUSPI_NUMINJFIFOS	The number of injection FIFOs per process reserved for use by an MU system programming interface (SPI) application.	0
MUSPI_NUMRECFIFOS	The number of reception FIFOs per process reserved for use by an MU SPI application.	0
MUSPI_RECFIFOSIZE	The size, in bytes, of each reception FIFO. Incoming torus packets are stored in this FIFO until software can process them. Making this size larger can reduce torus network congestion. Making this size smaller leaves more memory available to the application. PAMI messaging uses one reception FIFO per context.	1048576 bytes (1 MB)
PAMI_A2A_PACING_ WINDOW	The number of simultaneous send operations to start on Alltoall(v) collectives. Additional send operations cannot be started until these operations finish. This requirement reduces the resource usage for large geometries.	1024

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Environment variable	Description	Default value
PAMI_ ATOMICBARRIER_LOOPS	The number of attempts to complete the barrier in each pass.	32
PAMI_CLIENT_SHMEMSIZE	The number of bytes that are allocated from shared memory to each client. Use the “K” and “k” suffix as a 1024 multiplier or the 'M' and 'm' suffix as a 1024 × 1024 multiplier. The default value depends on the number of tasks in the node. 4M More than one task is on the node. 0 All other settings.	See description
PAMI_CLIENTS	A comma-separated ordered list of clients (no spaces). The complete syntax is [name][:repeat][/weight][,[name][:repeat][/weight]]* Each client has the form [name][:repeat][/weight], where: ► "name" is the name of the client. For example, the name of the Blue Gene/Q MPICH2 client is MPI. The default value for this option is the null string. ► ":repeat" is the repetition factor, where repeat is the number of clients having this same name. The default value for this option is 1. ► "/weight" is the relative weight assigned to the client, where weight is the weight value. The default value for this option is 1. The weight is used to determine the portion of the messaging resources that is given to the client, relative to the other clients. When middleware calls the PAMI_Client_create() function, it provides the name of the client. PAMI searches through the PAMI_CLIENTS in the order they are specified, looking for an exact name match. If there is not an exact name match with any of the PAMI_CLIENTS, PAMI searches through the PAMI_CLIENTS again, looking for a client with a null name string. The null name string is a wildcard and matches any client name. If there are exact or wildcard name matches, the first match that does not already have an active client is used, and the weight of that client determines the percentage of the available resources that are allocated to the client. If there are no available and matching clients, the PAMI client is not created. The default value of the PAMI_CLIENTS environment variable is `:1/1`, which means that all resources are assigned to the first client created, regardless of the client name, and all subsequent attempts to create a client fail due to insufficient resources. If any of the clients specified on PAMI_CLIENTS are unnamed, or more than one client has the same name, the order in which the clients are created must be the same on all processes in the job. The first client listed has exclusive use of the message unit combining collective hardware for optimizing reduction operations. The other clients use algorithms that do not use the message unit combining collective hardware.	":1/1"

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Environment variable	Description	Default value
PAMI_CLIENTS (continued)	The following examples show how the PAMI_CLIENTS variable can be used. ► PAMI_CLIENTS=MPI,CLIENTA means that up to two clients can use PAMI: one must be MPI, and the other must be CLIENTA. The MPI client is assigned the message unit combining collective hardware, and the two clients evenly split the remaining messaging resources. ► PAMI_CLIENTS=MPI:3,CLIENTA/2,CLIENTB:2/3 means that up to six clients can use PAMI. Three can be MPI, one can be CLIENTA, and two can be CLIENTB. Each MPI client has weight 1. CLIENTA has weight 2, and each CLIENTB client has weight 3. In this example, each CLIENTB client gets three times the amount of resources as each MPI client, and the first MPI client created is assigned the message unit combining collective hardware. ► PAMI_CLIENTS=MPI/3,/2 means that up to three clients can use PAMI. Two of the clients are unnamed, meaning that they can be any of the PAMI clients, and one client can only be MPI. The first MPI client created has resource weight 3 and is assigned the message unit combining collective hardware. The first non-MPI client created (or possibly the second MPI client created) has resource weight 2, and the second non-MPI client created (or possibly the second or third MPI client created) has resource weight 1. ► PAMI_CLIENTS is not specified. This setting means that there can be only one client, with any name, and it is assigned all of the resources. ► Default ":1/1" PAMI uses one reception FIFO per context and, optimally, uses 10 injection FIFOs per context, although fewer injection FIFOs can be used when resources are constrained. For more information, see the descriptions of the following variables: ► MUSPI_NUMBATIDS ► MUSPI_NUMCLASSROUTES ► MUSPI_NUMINJFIFOS ► MUSPI_NUMRECFIFOS ► MUSPI_INJFIFOSIZE ► MUSPI_RECFIFOSIZE ► PAMI_MU_RESOURCES	":1/1"
PAMI_CONTEXT_ SHMEMSIZE	Number of bytes allocated from shared memory to every context in each client. Use the “K” and “k” suffix as a 1024 multiplier or the “M” and “m” suffix as a 1024 × 1024 multiplier.	135K
PAMI_GLOBAL_SHMEMSIZE	Number of bytes allocated from shared memory for global information such as the mapcache. Use the “K” and “k” suffix as a 1024 multiplier, or the “M” and “m” suffix as a 1024 × 1024 multiplier.	4M
PAMI_M2M_ROUTING	For an all-to-all message transfer, this setting specifies the network routing that is used. "DETERMINISTIC" Use deterministic routing. "DYNAMIC" Use dynamic routing.	"DYNAMIC"

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Environment variable	Description	Default value
PAMI_M2M_ZONE	For an all-to-all message transfer that uses DYNAMIC routing, this variable specifies the routing zone that is used: ► 0 Use zone 0. ► 1 Use zone 1. ► 2 Use zone 2. ► 3 Use zone 3. The default settings depend on the size of the block: ► 1 The blocks is smaller than 512 nodes. ► 0 The block is 512 nodes or larger.	See description
PAMI_MAX_COMMTHREADS	Maximum number of commthreads to create. This setting can be used to avoid hardware thread oversubscription.	(64 / ranks per node) - 1
PAMI_MEMORY_OPTIMIZED	Determines whether PAMI is configured for a restricted memory job. If not set, PAMI is not memory optimized and uses memory as needed to increase performance.	Not set.
PAMI_MU_RESOURCES	Determines whether PAMI calculates the number of available contexts based on an “optimal” or a “minimal” allocation of MU resources to each context. Supported environment variable values are not case-sensitive and include: **Optimal** An optimal allocation of MU resources to each context limits the maximum number contexts that can be created. Each context is allocated sufficient MU resources to fully use the MU hardware and torus network. **Minimal** A minimal allocation of MU resources to each context allows the maximum number of contexts to be created regardless of MU hardware and torus network considerations.	"Optimal"
PAMI_ NUMDYNAMICROUTING	Number of simultaneous dynamically routed messages per context. If more than this many messages are being transferred, the additional messages are deterministically routed. Dynamic routing can be faster than deterministic routing. However, dynamically routed messages require more storage to track their progress, hence the reason for this option. Specify this number in increments of 64 (for example: 64, 128, 192, 256, ...).	64
PAMI_RGETINJFIFOSIZE	The size, in bytes, of each remote get FIFO. These FIFOs store 64-byte descriptors. Each descriptor describes a memory buffer to be sent on the torus, and is used to queue requests for data (remote gets). Making this size larger might reduce torus network congestion and reduce overhead. Making this size smaller might increase that congestion, memory usage, and latency. PAMI messaging uses 10 remote get FIFOs per node.	65536 (64 KB)
PAMI_RGETPACING	Specifies whether to consider messages for pacing: ► 0 No messages are paced. ► 1 Messages are considered for pacing. The default setting depends on the block size. ► 0 The block size is one rack (1024 nodes) or smaller. ► 1 The block size is larger than one rack.	See description

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Environment variable	Description	Default value
PAMI_RGETPACINGDIMS	Messages between nodes whose coordinates differ in more than this many dimensions in ABCD are considered for pacing. For example, node A has ABCD coordinates (0,0,0,0) and node B has (3,2,1,0). They differ in three dimensions (A, B, and C). Specifying 2 means that messages between these nodes are considered for pacing.	1
PAMI_RGETPACINGHOPS	Messages between nodes that are more than this many hops apart on the network are considered for pacing.	4
PAMI_RGETPACINGSIZE	Messages exceeding this size in bytes are considered for pacing.	65536 (64 KB)
PAMI_ROUTING	Specifies the PAMI network routing options to be used for point-to-point messages that are large enough to use the rendezvous protocol. That is, the messages are larger than the size specified for PAMID_EAGER. The complete syntax is PAMI_ROUTING=[size][,[small][,[low:high][,[in][,out]]]] When the source and destination nodes are on a network line (their ABCDE coordinates differ in at most one dimension), deterministic routing is always used. PAMI_ROUTING does not override this setting. When the message size is less than or equal to "size", PAMI uses the "small" network routing. When the message size is larger than "size", PAMI uses the "flexibility metric" to determine the network routing as follows: The "low:high" range is the flexibility metric range. The flexibility metric gauges the routing flexibility between a source node and destination node. The values for "low" and "high" must be floating-point numbers in the range 0.0 through 4.0. The low value must be less than or equal to the high value. The PAMI computes the flexibility metric between the source node and the destination node of a messaging transfer. The metric is the sum of the flexibility of dimensions A, B, C, and D between those nodes. The flexibility of a particular dimension is the ratio of the number of hops between the source and the destination in that dimension and the size of that dimension, and can range from 0.0 through 1.0. The "in" value is the network routing to be used for a message transfer between two nodes when their flexibility metric is between "low" and "high", and the "out" value is the network routing otherwise. The values for “in” and “out” and “small” might each be one of the following values. 0 Dynamic routing zone 0 1 Dynamic routing zone 1 2 Dynamic routing zone 2 3 Dynamic routing zone 3 4 Deterministic routing	See Table D-4 on page 141

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Environment variable	Description	Default value
PAMID_ASYNC_PROGRESS	This variable determines whether one or more communications threads are started to make asynchronous progress. This variable is required for maximum performance in message throughput cases: 0 No internal communications threads are started to assist with making communications progress 1 One or more communications threads can assist with making communications progress. Use this setting when high message throughput is required. The default value depends on the settings that are used: 0 The application is linked with the "legacy" MPICH libraries (gcc.legacy, xl.legacy, xl.legacy.ndebug), MPI_Init_thread() is called without MPI_THREAD_MULTIPLE, or the MPI_Init() function is called. 1 The application is linked with the gcc, xl, and xl.ndebug MPICH libraries and the MPI_Init_thread() function is called with MPI_THREAD_MULTIPLE. The default value cannot be changed when the application is linked with the "legacy" MPICH libraries (gcc.legacy, xl.legacy, or xl.legacy.ndebug). Attempting to set this environment variable causes an application that is linked with a "legacy" MPICH library to display a message and exit during the MPI_Init() function call.	See description
PAMID_COLLECTIVE_*name*	Turns on or off specific protocols for the MPI collective specified as *name.* Possible values for collectives are ALLTOALL, ALLTOALLV, ALLREDUCE, BARRIER, BCAST, SCATTER, SCATTERV, GATHER, GATHERV, ALLGATHER, ALLGATHERV, SCAN, and REDUCE. The MPICH option can be used to turn off all optimizations for a specific collective and use the MPICH point-to-point protocol. For many MPI_collective operations, this setting can cause poor performance on larger blocks. For information about other options and the default values, use the PAMID_VERBOSE=2 setting. For some PAMID_COLLECTIVE_* environment variables, especially PAMID_COLLECTIVE_ALLREDUCE, some optimized protocols only work with specific parameters (such as data type, operation, or message size) that are specified for the particular MPI_collective invocation. Therefore, specifying a specific protocol on the PAMID_COLLECTIVE_* environment variable might not work for a given MPI_collective invocation. In that case, the MPICH protocol is used instead. To find out which protocol was used for a specific MPI_collective invocation, invoke the MPIX_Get_last_algorithm_name() function immediately after the MPI_collective invocation. For more information, see the mpix.h file.	See description
PAMID_COLLECTIVES	Controls whether optimized collectives are used. The possible values are: 0 Optimized collectives are not used. Only MPICH point-to-point based collectives are used. 1 Optimized collectives are used.	1

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Environment variable	Description	Default value
PAMID_CONTEXT_MAX	This variable sets the maximum allowable number of contexts. Contexts are a method of dividing hardware resources among a Parallel Active Messaging Interface (PAMI) client (for example, MPI) to set how many parallel operations can occur at one time. Contexts are similar to channels in a communications system. The practical maximum is usually 64 contexts per node. The default value depends on the number of processes per node and the settings that are used: 1 The application is linked with the "legacy" MPICH libraries (gcc.legacy, xl.legacy, xl.legacy.ndebug), the MPI_Init_thread() function is called without MPI_THREAD_MULTIPLE, or the MPI_Init() function is called. ≥1 The application is linked with the gcc, xl, and xl.ndebug MPICH libraries and the MPI_Init_thread() function is called with MPI_THREAD_MULTIPLE. The default value cannot be changed when the application is linked with the "legacy" MPICH libraries (gcc.legacy, xl.legacy, or xl.legacy.ndebug). Attempting to set this environment variable causes an application that is linked with a "legacy" MPICH library to display a message and exit during the MPI_Init() function call.	See description
PAMID_CONTEXT_POST	This variable must be enabled to allow parallelism of multiple contexts. It might increase latency. Enabling this variable is the only method to allow parallelism between contexts: 0 Only one parallel communications context can be used. Each operation runs in the application thread. 1 Multiple parallel communications contexts can be used. An operation is posted to one of the contexts, and communications for that context are driven by communications threads. The default value depends on the settings that are used: 0 The application is linked with the "legacy" MPICH libraries (gcc.legacy, xl.legacy, xl.legacy.ndebug), the MPI_Init_thread() function is called without MPI_THREAD_MULTIPLE, or the MPI_Init() function is called. 1 The application is linked with the gcc, xl, and xl.ndebug MPICH libraries and MPI_Init_thread() is called with MPI_THREAD_MULTIPLE. The default value cannot be changed when the application is linked with the "legacy" MPICH libraries (gcc.legacy, xl.legacy, or xl.legacy.ndebug). Attempting to set this environment variable causes an application that is linked with a "legacy" MPICH library to display a message and exit during the MPI_Init() function call.	See description

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Environment variable	Description	Default value
PAMID_DISABLE_INTERNAL _EAGER_TASK_LIMIT	Overrides the default job size at which point the eager protocols are disabled for internal MPI operations. This override has the same effect as specifying the environment variable: `PAMID_PT2PT_LIMITS=::::0:0:0:0` This environment variable is processed before the PAMID_EAGER or PAMID_RZV, PAMID_EAGER_LOCAL or PAMID_RZV_LOCAL, PAMID_SHORT, and PAMID_PT2PT_LIMITS environment variables.	512k
PAMID_EAGER_LOCAL, PAMID_RZV_LOCAL	Sets the cutoff value for the switch to the rendezvous protocol when the destination rank is local. The two options are identical. This variable takes an argument, in bytes, to switch from the eager protocol to the rendezvous protocol for point-to-point messaging. The default value effectively disables the eager protocol for local transfers because the default value for PAMID_EAGER_LOCAL is less than the default value for PAMID_SHORT. The 'K' and 'M' multipliers can be used in the value. For example, "16K" or "1M" can be used.	4097 bytes
PAMID_EAGER, PAMID_RZV	Sets the cutoff for the switch to the rendezvous protocol. These options are identical. This variable takes an argument, in bytes, to switch from the eager protocol to the rendezvous protocol for point-to-point messaging. Increasing the limit might help for larger blocks and if most of the communication is with the nearest neighbor. The 'K' and 'M' multipliers can be used in the value. For example, "16K" or "1M" can be used.	4097 bytes

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Environment variable	Description	Default value

PAMID_PT2PT_LIMITS	Specify all point-to-point limit overrides. This environment variable is processed after the PAMID_EAGER or PAMID_RZV, PAMID_EAGER_LOCAL or PAMID_RZV_LOCAL, and PAMID_SHORT environment variables.
	The entire point-to-point limit set is determined by three Boolean configuration values:
	► 'is non-local limit' versus 'is local limit'
	► 'is eager limit' versus 'is immediate limit'
	► 'is application limit' versus 'is internal limit'
	The point-to-point configuration limit values are specified in order and are delimited by ':' characters. If a value is not specified for a given configuration, the limit is not changed. There is no requirement to specify all eight configuration values. However, to set the last (eighth) configuration value, the previous seven configurations must be listed. The 'k', 'K', 'm', and 'M' multipliers can be specified. For example:
	
	`PAMID_PT2PT_LIMITS="::::::10k"`
	
	The configuration entries can be described as:
	0 remote eager application limit
	1 local eager application limit
	2 remote immediate application limit
	3 local immediate application limit
	4 remote eager internal limit
	5 local eager internal limit
	6 remote immediate internal limit
	7 local immediate internal limit
	
	The following example show how the PAMID_PT2PT_LIMITS variable can be used.
	► "10K" sets the application internode eager (the "normal" eager limit)
	► "10240::64" sets the application internode eager and immediate limits
	► "::::0:0:0:0" disables 'eager' and 'immediate' for all internal point-to-point limits
	This environment variable does not override any point-to-point limits by default.
	
	If no other point-to-point limit environment variables are used, and if the job size is less than PAMID_DISABLE_INTERNAL_EAGER_TASK_LIMIT, the effective default value is:
	
	`4097:4097:113:113:2049:64:113:113`
	
	If no other point-to-point limit environment variables are used, and if the job size is not less than PAMID_DISABLE_INTERNAL_EAGER_TASK_LIMIT, the effective default value is:
	
	`4097:4097:113:113:0:0:0:0`
		See description

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Environment variable	Description	Default value
PAMID_RMA_PENDING	Maximum outstanding Remote Memory Access (RMA) requests. Limits the number of PAMI_Request objects allocated by MPI one-sided (also known as RMA) operations. The 'K' and 'M' multipliers can be used in the value. For example, "16K" or "1M" can be used.	1000
PAMID_SHMEM_PT2PT	Determines whether intranode point-to-point communication uses the optimized shared memory protocols: 0 Optimized shared memory protocols are not used. 1 Optimized shared memory protocols are used.	1
PAMID_SHORT	Sets the cutoff for the switch to the eager protocol. This variable takes an argument, in bytes, to switch from the short protocol to the eager protocol for point-to-point messaging. If a value greater than 113 bytes is specified, 113 bytes are used. The 'K' and 'M' multipliers can be used in the value. For example, "16K" or "1M" can be used.	113 bytes
PAMID_STATISTICS	Turns on the printing of statistics for the message layer such as the maximum receive queue depth. Possible values: 0 No statistics are printed. 1 Statistics are printed.	0
PAMID_THREAD_ MULTIPLE	Specifies the messaging execution environment. It specifically selects whether there can be multiple independent communications occurring in parallel, driven by internal communications threads: 0 The application threads drive the communications. No additional internal communications threads are used. This setting is equivalent to specifying PAMID_ASYNC_PROGRESS=0, PAMID_CONTEXT_POST=0, PAMID_CONTEXT_MAX=1. 1 There can be multiple independent communications occurring in parallel, driven by internal communications threads. This setting is equivalent to specifying PAMID_ASYNC_PROGRESS=1, PAMID_CONTEXT_POST=1, PAMID_CONTEXT_MAX≥1. The default value depends on the settings that are used: 0 The application is linked with the "legacy" MPICH libraries (gcc.legacy, xl.legacy, xl.legacy.ndebug), or the MPI_Init_thread() function is called without MPI_THREAD_MULTIPLE, or the MPI_Init() function is called. 1 The application is linked with the gcc, xl, and xl.ndebug MPICH libraries and the MPI_Init_thread() function is called with MPI_THREAD_MULTIPLE. The default value cannot be changed when the application is linked with the "legacy" MPICH libraries (gcc.legacy, xl.legacy, or xl.legacy.ndebug). Attempting to set this environment variable causes an application that is linked with a "legacy" MPICH library to display a message and exit during the MPI_Init() function call.	See description

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Environment variable	Description	Default value
PAMID_VERBOSE	Provides debugging information during MPI_Abort() and during various MPI function calls. Some settings affect performance. To simplify debugging, set this variable to 1 for all applications: 0 No additional information is provided. 1 Print summary information during the MPI_Init() function call on rank 0. Use this setting to simplify debugging. It only impacts performance by using the MPI_Init() function to print the information. A small amount of text is printed, including the PAMID_, PAMI_, MUSPI_, COMMAGENT_, and BG_ environment variables and other variables that the user specifies. The MPI_Init() function does not verify that variable names are specified correctly. 2 Print summary information for collective operations and print additional information for point-to-point. This information can be useful when debugging which collective is being used on a communicator. Approximately one line of output per rank per communicator is created, and one line of output per rank of point-to-point send statistics are provided on finalize. This setting can affect the performance of routines that are typically not timed (for example, MPI_Comm_create, MPI_Finalize, and so on). 3 Print detailed information. This setting generates extensive information when used with large numbers of ranks.	0

Table D-2 shows the default values for the COMMAGENT_RGETPACINGMAX environment variable.

Table D-2 Default values for the COMMAGENT_RGETPACINGMAX variable

Block size (racks)	COMMAGENT_RGETPACINGMAX value
Racks < 2	65536
2 ≤ racks < 4	65536
4 ≤ racks < 8	32768
8 ≤ racks < 16	24576
16 ≤ racks < 32	24576
32 ≤ racks < 48	24576
48 ≤ racks < 64	24576
64 ≤ racks < 80	24576
80 ≤ racks < 96	24576
96 ≤ racks	24576

Table D-3 shows the default values for the COMMAGENT_RGETPACINGSUBSIZE environment variable. These values vary depending on the size of the block where the job is being run.

Table D-3 Default settings for the COMMAGENT_RGETPACINGSUBSIZE variable

Block size (racks)	COMMAGENT_RGETPACINGSUBSIZE value
Racks < 2	16384

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Block size (racks)	COMMAGENT_RGETPACINGSUBSIZE value
2 ≤ racks < 4	16384
4 ≤ racks < 8	8192
8 ≤ racks < 16	8192
16 ≤ racks < 32	8192
32 ≤ racks < 48	8192
48 ≤ racks < 64	8192
64 ≤ racks < 80	8192
80 ≤ racks < 96	8192
96 ≤ racks	8192

Table D-4 shows the default values for the PAMI_ROUTING environment variable. These values vary depending on the size of the block where the job is being run.

Table D-4 Default values for the PAMI_ROUTING environment variable

Block size	Size	Small	Flexibility metric range, low - high	Routing when in range	Routing when out of range
32 ≤ nodes < 64	65536	4	1.5 - 3.5	3	3
64 ≤ nodes < 128	65536	4	1.5 - 3.5	3	3
128 ≤ nodes < 256	65536	4	1.5 - 3.5	3	3
256 ≤ nodes < 512	65536	4	1.5 - 3.5	3	3
512 ≤ nodes < 1024	65536	4	1.5 - 3.5	3	3
1 ≤ racks < 2	65536	4	1.5 - 3.5	2	0
2 ≤ racks < 4	65536	4	1.5 - 3.5	3	0
4 ≤ racks < 8	65536	4	1.5 - 3.5	3	0
8 ≤ racks < 16	65536	4	1.5 - 2.5	3	0
16 ≤ racks < 32	65536	4	1.3 - 3.0	3	0
32 ≤ racks < 48	65536	4	1.3 - 3.0	3	0
48 ≤ racks < 64	65536	4	1.3 - 3.0	3	0
64 ≤ racks < 80	65536	4	.75 - 3.0	3	0
80 ≤ racks < 96	65536	4	.75 - 3.0	3	0
96 ≤ racks	65536	4	.75 - 3.0	3	0

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Compute Node Kernel environment variables

Several environment variables affect the runtime characteristics of the Compute Node Kernel (CNK). If these variables are set incorrectly, programs might not run. None of these environment variables are required to be set for the CNK to work.

Table D-5 lists the CNK environment variables.

Table D-5 MPI environment variables

Environment variable	Description	Default value
BG_AGENTHEAPSIZE	The heap size in MB that is allocated to the application agent process. The default value is 16 if an application agent is defined in the BG_APPAGENT environment variable. Otherwise, it is 0.	16 or 0. See description.
BG_AGENTCOMMHEAPSIZE	The heap size in MB that is allocated to the application agent that is reserved for use by the messaging software. The default value is 16 if an application agent for messaging is not disabled by BG_APPAGENTCOMM=DISABLE. Otherwise, it is 0.	16 or 0. See description.
BG_APPAGENT	The path to an application agent program. The default is no application agent.	See description
BG_APPAGENTCOMM	The path to an application agent that is reserved for use by the messaging software. To disable the default PAMI application agent, specify DISABLE. The default value is `/bgsys/drivers/ppcfloor/agents/bin/comm.elf`	See description
BG_COREDUMPBINARY	Specifies the MPI ranks for which a binary core file is generated rather than a lightweight core file. This type of core file can be used with the GNU Project Debugger (GDB) but not the Blue Gene/Q Coreprocessor utility. If this variable is not set, all ranks generate a lightweight core file. To generate a binary core file, set the variable to a comma-separated list of the ranks. To have all ranks generate a binary core file, set the variable to "*" (an asterisk).	See description
BG_COREDUMPDISABLED	Boolean that specifies whether core files are created: 0 Enable creation of core files. 1 Disable creation of core files.	0
BG_COREDUMPFILEPREFIX	Sets the file name prefix of the core files. The MPI task number is appended to this prefix to form the file name.	"core."
BG_COREDUMPFPR	Boolean that controls whether register information is included in the core files. BG_COREDUMP_FPR controls output of floating-point registers (FPRs): 0 Disable this setting. 1 Enable this setting.	1
BG_COREDUMPGPR	Boolean that controls whether register information is included in the core files. BG_CORE_DUMPGPR controls integer general-purpose registers (GPRs): 0 Disable this setting. 1 Enable this setting.	1

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Environment variable	Description	Default value
BG_COREDUMPINTCOUNT	Boolean that controls whether the number of interrupts handled by the node is included in the core file: 0 Disable this setting. 1 Enable this setting.	1
BG_COREDUMPMAXNODES	Specifies the maximum number of nodes that generate core files for abnormally terminating processes. This variable can be used to limit the number of core files generated in cases when most of the processes in a very large block abnormally terminate.	2048
BG_COREDUMPONERROR	Boolean that controls the creation of core files when the application exits with a nonzero exit status. This variable is useful when the application performed an exit(1) operation and the cause and location of the exit(1) is not known: 0 Disable this setting. 1 Enable this setting.	0
BG_COREDUMPONEXIT	Boolean that controls the creation of core files when the application exits. This variable is useful when the application performed an exit() operation and the cause and location of the exit() operation is not known. To enable this setting, the value must be set to 1: 0 Disable this setting. 1 Enable this setting.	0
BG_COREDUMPPATH	Sets the directory for the core files. The default value is the current working directory.	See description.
BG_COREDUMPPERS	Boolean that controls whether the node personality information (XYZ dimension location, memory size, and so on) is included in the core files: 0 Disable this setting. 1 Enable this setting.	1
BG_COREDUMPRANKS	Specifies a comma-separated list of ranks to generate a core file when the job ends. The ranks specified in this list are not prevented from being generated by any other BG_COREDUMP environment variable.	See description.
BG_COREDUMPREGS	Boolean that controls whether register information is included in the core files. BG_COREDUMPREGS is the master switch: 0 Disable this setting. 1 Enable this setting.	1
BG_COREDUMPSPR	Boolean that controls whether register information is included in the core files. BG_COREDUMP_SPR controls the output of special-purpose registers (SPRs): 0 Disable this setting. 1 Enable this setting.	1

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Environment variable	Description	Default value
BG_COREDUMPSTACK	Boolean that controls whether the application stack addresses are to be included in the core file: 0 Disable this setting. 1 Enable this setting.	0
BG_COREDUMPTLBS	Boolean that controls whether the TLB layout at the time of the core is to be included in the core file: 0 Disable this setting. 1 Enable this setting.	1
BG_MAPCOMMONHEAP	This option obtains a uniform heap allocation between the processes; however, the trade off is that memory protection between processes is not as stringent. In particular, when using the option, it is possible to write into another process' heap. Normally this would cause a segmentation violation, but with this option set, the protection mechanism is disabled to provide a balanced heap allocation. The processes have independent heaps and system calls return EFAULT if an address is passed in that is out-of-bounds.	0
BG_MAPNOALIASES	This option disables long-running alias mode. This feature is used for some TM or SE configurations.	0
BG_MAPALIGN16	This option changes the memory alignment restrictions for ppn = 16, ppn = 32, and ppn = 64. With the option enabled, the physical memory for each process begins on a 16 MB boundary, as opposed to a power-of-2 size. This has the potential for better memory mappings for ppn $\geq$ 16. By default, BG_MAPALIGN16 is enabled.	1
BG_MAXALIGNEXP	The maximum number of floating-point alignment exceptions that the CNK can handle. If the maximum is exceeded, the application core dumps: 0 No alignment exceptions are processed. -1 All alignment exceptions are processed. <n> n alignment exceptions are processed	1000
BG_PERSISTMEMRESET	Boolean that indicates that the persistent memory region must be cleared before the job starts: 0 Disable this setting. 1 Enable this setting.	0
BG_PERSISTMEMSIZE	Size, in MB, of the persistent memory region.	0
BG_POWERMGMTDUR	The number of microseconds spent in one proactive power management idle loop.	0
BG_POWERMGMTPERIOD	The number of microseconds between proactive power management idle loops. When 0, power management is disabled.	0

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Environment variable	Description	Default value
BG_SHAREDMEMSIZE	Size, in MB, of the shared memory region. To increase the default value by a specific number of MB, specify a '+' prefix with the value. To replace the default value with a new value, omit the '+'. The default shared memory size is chosen by the CNK based on the known requirements of the current configuration: If ranks-per-node = 1, the shared memory size defaults to 32 MB. If ranks-per-node > 2, the shared memory size defaults to 64 MB.	See description.
BG_STACKGUARDENABLE	Boolean that indicates whether the CNK creates guard pages. If the variable is specified, a value must be set to either 0 or 1: 0 Do not create guard pages. 1 Create guard pages.	0
BG_STACKGUARDSIZE	The size, in bytes, of the main() function stack guard area. If the specified value is greater than zero but less than 512, 512 bytes are used.	4096
BG_SYSIODPOSIXMODE	Run I/O operations with POSIX rules: 0 I/O operation that is initiated from a compute node can cause multiple I/O operations on the I/O node. 1 Each I/O operation that is initiated from a compute node completes atomically.	0
BG_THREADLAYOUT	Specifies the algorithm that the CNK uses to select a hardware thread during software thread creation: 1 Assign software threads across the cores within the process before assigning software threads to additional hardware threads within a core. 2 Assign software threads to all hardware threads within a core before assigning software threads on other cores.	1
BG_THREADMODEL	Activates a specific thread model: 0 Operate in the native Blue Gene/Q thread model, allowing multiple pthreads per hardware thread. 1 Allow only one application pthread per hardware thread. 2 For V1R2M0 and later releases, enable the extended thread affinity control.	0

Setting environment variables

The simplest method to set environment variables is to specify them on the command line when running the **runjob** command. For example, to set environment variable “XYZ” to value “ABC,” call the **runjob** command as the following example shows:

```
$ runjob --envs XYZ=ABC myprogram.rts
```

To send multiple environment variables, separate them with a space, for example:

```
$ runjob --envs XYZ=ABC DEF=123 myprogram.rts
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

E

Using GNU profiling

This appendix describes the GNU profiling function that is provided in the GNU toolchain for Blue Gene/Q.

For basic documentation about the GNU toolchain profiling tools, see the **gprof** documentation on the following web site:

`http://sourceware.org/binutils/docs-2.21/gprof/index.html`

The path for the Blue Gene/Q gprof utility is `/bgsys/drivers/ppcfloor/gnu-linux/bin/`powerpc64-bgq-linux-gprof.

© Copyright IBM Corp. 2013. All rights reserved.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Using the Blue Gene/Q gmon tool

The basic **gmon** support is described in the man pages for the GNU toolchain:

http://gcc.gnu.org/

Specifying which ranks generate gmon.out files

Blue Gene/Q applications typically run simultaneously on multiple ranks. The execution path through the code, and therefore the profiling data collected, might vary depending on the rank. A separate gmon.out file is generated for each rank in the program. The gmon.out files are named $gmon.out.N$ where N is the rank where the program was run.

On large blocks, many gmon.out files might be written. It is normally not necessary to write a file for each rank because many of the files contain similar or identical information. The default setting is to generate gmon.out files only for profiling data collected on ranks 0 - 31. This setting reduces I/O usage. To generate gmon.out files for a different set of ranks, use the BG_GMON_RANK_SUBSET environment variable to specify which ranks have gmon.out files generated. The variable can be set as shown in Example E-1.

Example E-1 Setting the BG_GMON_RANK_SUBSET variable

```
BG_GMON_RANK_SUBSET=N /* Only generate the gmon.out file for rank N. */

BG_GMON_RANK_SUBSET=N:M /* Generate gmon.out files for all ranks from N to M. */

BG_GMON_RANK_SUBSET=N:M:S /* Generate gmon.out files for all ranks from N to M.
Skip S; 0:16:8 generates gmon.out.0, gmon.out.8, gmon.out.16  */
```

Functions to disable gmon.out files for some nodes

The Blue Gene/Q toolchain includes the mondisable() function. This function can be called from the user program to prevent writing out a gmon.out file on the node from which it is called. This function can be used to reduce the number of gmon.out files that are generated on a large block. It can be especially useful if it is known that many nodes generate the same or similar profiling data.

Profiling for threads

The base GNU toolchain does not provide support for profiling on threads. The patches that are provided for the Blue Gene/Q toolchain include support for profiling on threads. These patches include various methods to enable and disable profiling on a per-thread basis.

Profiling with the GNU toolchain

Profiling tools provide information about potential bottlenecks in the program. They help identify functions or sections of the code that might become good candidates to optimize. When using **gmon** profiling, two levels of profiling information can be generated: machine instruction level or full level. Select options based on the required level of detail and the acceptable amount of overhead. As with any type of performance data collection, monitoring and saving of performance information uses system resource and affects the resulting performance data. The amount of additional resource, or profiling overhead, is greater with

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

some options than with others. When compiling with the GNU compilers or the IBM XL compilers, enable profiling by adding -pg to the compile flags.

Using timer tick (machine instruction level) profiling

This level of profiling provides timer tick profiling information at the machine instruction level. Profiling data collection is based on the SIGPROF timer. The timer is enabled in the program. When the timer expires, the program counter for the executing instruction is updated. As the program runs, the data collection provides a sample of instruction addresses that are executed by the program. To enable this type of profiling and no other performance data collection, add the -pg option on the link command but do not include it on the compile commands. This level of profiling adds the least amount of performance collection overhead. It provides profile information based on instruction addresses but does not provide call graph or call count information.

In the base GNU toolchain, threads are not profiled by default. However, the Blue Gene/Q system includes support for thread profiling. Thread profiling is not enabled by default. To enable thread profiling, link the program with the -pg option and perform one of the following steps:

- Set the BG_GMON_START_THREAD_TIMERS environment variable on the **`runjob`** command.

  Set this environment variable to “all” to enable the SIGPROF timer on all threads created with the pthread_create() function.

  When profiling an MPI application, additional threads called comm threads might be created to assist with the MPI function. Set this environment variable to “nocomm” to enable the SIGPROF timer on all threads except the extra threads that are created to support MPI.
- Add a call to the gmon_start_all_thread_timers() function to the program so that it is called from the main thread. This setting configures the SIGPROF timer on all threads that are created with pthread_create after the point where this call is made. Threads that are created before the call to the gmon_start_all_thread_timers() function are not profiled.
- Add a call to the gmon_thread_timer(int start) function from the thread to be profiled. To start the thread time, call this function with 1 as an argument. To stop the thread timer, call this function with the value 0.

The prototypes for these **`gmon`** functions are in `sys/gmon.h` in the Blue Gene/Q toolchain.

Collecting call count information

In addition to instruction profiling, call count information can also be collected. To collect this type of data, all files must be compiled and linked with the -pg option. This option provides profiling information based on the SIGPROF timer as described in “Using timer tick (machine instruction level) profiling” on page 149 and call graph information and procedure call counts. This level of performance data collection introduces the most overhead. The call count information is collected on all threads that execute code that was compiled with the -pg option. It is not affected or controlled by the thread profiling switches that are described in this appendix. When higher levels of compiler optimization are used, the statement mappings and procedure calls might not appear as expected due to inlining, code movement, scheduling, and other optimizations that are performed by the compiler. Programs built this way collect call count information for all threads.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

To also collect profiling information for threads, thread-level profiling must be enabled with one of the methods that are described in “Using timer tick (machine instruction level) profiling” on page 149.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

F

Hardware performance counters

The Blue Gene/Q system has multiple hardware performance events and counters. To tune performance and monitor hardware performance events, use the Blue Gene/Q Hardware Performance Monitoring (BGPM) API or another tool that uses it. This section describes the BGPM API and the industry-standard Performance Application Programming Interface (PAPI-C).

© Copyright IBM Corp. 2013. All rights reserved.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Blue Gene Hardware Performance Monitoring API

The native BGPM API implements a C programming interface for the Blue Gene/Q universal performance counter hardware. You can use BGPM API functions to program, control, and access counters and events from the four integrated hardware units and the CNK software counters.

The BGPM API documentation is in HTML and is stored in the driver directory at `/bgsys/drivers/ppcfloor/bgpm/docs/html/index.html`. It is also available in the Blue Gene Navigator Knowledge Center.

The documentation includes:

- An overview and quick start example.
- Extended overview pages about units and limitations.
- A reference section with the API calls, event tables, instruction operation codes and instruction groups, tips, and some examples.

The BGPM API operation is fairly low level. The API can be used to abstract the hardware collections into manageable events and units. It does not include constructs such as formatted reports or direct profiling. However, it does provide functions on which these constructs can be built. These functions include event sets, event information queries, overflow detection, handler routines, and coordinated multiplexing. The BGPM API includes the following features:

- There are five types of units:
  - The Punit counters, which control events and counters that are related to the hardware-threaded CPU
  - The L2 unit
  - The I/O unit
  - The Network unit
  - The CNK unit counters and events
- All counters are 64 bits.
- There are three major modes of operation:
  - Software distributed mode, where each software thread configures and controls its own Punit counters
  - Hardware distributed mode, where a single software thread can configure and simultaneously control all Punit counters for all cores
  - Low latency mode. This mode provides faster start and stop access to the Punit counters. However, the Punit counters have only a 14-bit capacity.
- The software distributed mode additional support for:
  - Interrupt on overflow. The Punit, L2, and I/O units support interrupt on overflow. This support can be used to register and set thresholds for the counter value and to call an overflow handler when the threshold value is reached.
  - Punit event multiplexing. Multiplexing allows more events to be configured than can be simultaneously collected. Event collection is time-sliced based on a CPU cycle threshold or manually by the user instrumentation.
  - Abstract hardware thread counters. These counters are used to present a software thread context (Punit counters are stopped and restarted when the thread is swapped in or out) in cooperation with the CNK.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

- The Network unit provides counters for each of 11 separate network links per node and events for filtering based on types of transfers, including events to allow calculation of use and receive queue length.
- The separate units are generally programmed and controlled separately, but use a common set of BGPM API functions.
- There are 405 base events. Many of these events can be varied with user-selectable attributes.
- All cores and hardware threads can be gathered simultaneously. The CPU core events can be programmed to use 6, 12, or 24 counters for a thread. The number of counters depends on the number of active hardware threads per core.

Performance Application Programming Interface

The PAPI-C library is an industry-standard API that is designed to provide a consistent interface and methodology for using the performance counter hardware. It can be used to interface with the BGPM API to control and access the performance counters.

For more information about the PAPI-C interface, including the documentation and toolkit, see the PAPI website:

`http://icl.cs.utk.edu/papi`

The PAPI-C library must be downloaded, built, and installed separately from the Blue Gene/Q drivers. For information about installing the library, see the PAPI release installation notes in the installation directory. The path is papi/INSTALL.txt.

The PAPI-C features that can be used for the Blue Gene/Q system include:

- A standard instrumentation API that can be used by other tools.
- A collection of standard preset events, including some events that are derived from a collection of events. The BGPM API native events can also be used through the PAPI-C interfaces.
- Support for both a C and a Fortran instrumentation interface.
- Support for separate components for each of the BGPM API unit types. The Punit counter is the default PAPI-C component. The L2, I/O, Network, and CNK units require separate component instances in the PAPI-C interface.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

G

Requirements for C++ programming in a failover environment

In a failover configuration, special consideration must be taken for C++ applications that use the Standard Input/Output facilities of the C++ Standard Library. During a failover event, the standard output and standard error streams are closed by the system software. The error state flags must be cleared by the application afterwards to resume using these streams. Example G-1 shows one method to clear the error state flags.

Example G-1

```
if ( !(cout << "Hello World" << endl) ) {
    cout.clear();
}
```

If the error state is left in the set state, standard output or standard error information is lost because the error state flags for the stream are on.

© Copyright IBM Corp. 2013. All rights reserved.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Abbreviations and acronyms

ABI	Application binary interface
ADI	Abstract device interface
ALU	Arithmetic logic unit
API	Application programming interface
ASIC	Application-specific integrated circuit
BIOS	Basic input/output system
CDTI	Code Development and Tools Interface
CIOD	Control and I/O daemon
DDR	Double data RAM
DMA	Direct memory access
ELF	Executable and linking format
EPL	Eclipse Public License
ESSL	Engineering and Scientific Subroutine Library
FIFO	First-in, first-out queue
FPR	Floating-point register
FPU	Floating-point unit
GCC	GNU Compiler Collection
GDB	GNU Project Debugger
GLIBC	GNU C Library
GPFS	General Parallel File System
GPR	General-purpose register
HPC	High-performance computing
HTC	High-throughput computing
INF	Infinity
IP	Internet protocol
L1p	L1 prefetcher
LLCS	Low Level Control System
MASS	Mathematical Acceleration Subsystem
MMCS	Midplane Management Control System
MPI	Message Passing Interface
MU	Messaging unit
MUSPI	Message unit system programming interface
NAN	Not a number
NFS	Network File System
NPTL	Native POSIX Thread Library
OSS	Open Source Software
OpenMP	Open multi-processing
PAMI	Parallel Active Messaging Interface
PID	Process identifier
POSIX	Portable Operating System Interface
QCD	Quantum chromodynamics
QPX	Quad-processing extensions
RAM	Random access memory
RAS	Reliability, availability, and serviceability
ROM	Read-only memory
RPM	RPM package manager
SIMD	Single instruction, multiple data
SMP	Symmetrical multiprocessing
SPI	System programming interface
SPR	Special-purpose registers
TCP	Transmission Control Protocol
TGID	Thread group identifier
TID	Thread identifier
UDP	User Datagram Protocol

© Copyright IBM Corp. 2013. All rights reserved.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Related publications

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

IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks” on page 160. Note that some of the documents referenced here might be available in softcopy only:

- *IBM System Blue Gene Solution: Blue Gene/Q Code Development and Tools Interface*, REDP-4659
- *IBM System Blue Gene Solution: Blue Gene/Q Hardware Overview and Installation Planning*, SG24-7872-00
- *IBM System Blue Gene Solution: Blue Gene/Q Hardware Installation and Maintenance Guide*, SG24-7974-00
- *IBM System Blue Gene Solution: Blue Gene/Q Safety Considerations*, REDP-4656
- *IBM System Blue Gene Solution: Blue Gene/Q System Administration*, Manual, SG24-7869-00

Other publications

These publications are also relevant as further information sources:

- *General Parallel File System HOWTO for the IBM System Blue Gene/Q Solution*, SC23-6939-00
- Gropp, W. and Lusk, E. "Dynamic Process Management in an MPI Setting." *7th IEEE Symposium on Parallel and Distributed Processing*. p. 530, 1995:

  http://www.cs.uiuc.edu/homes/wgropp/bib/papers/1995/sanantonio.pdf
- Gropp, William; Huss-Lederman, Steven; Lumsdaine, Andrew; Lusk, Ewing; Nitzberg, Bill; Saphir, William; Snir, Marc. *MPI: The Complete Reference, Volume 2 - The MPI-2 Extensions*. MIT Press, Cambridge, Massachusetts, 1998. ISBN 0-262-69216-3.
- Heyman, J. "Recommendations for Porting Open Source Software (OSS) to Blue Gene/P," white paper WP101152.

  http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP101152
- Quinn, Michael J. *Parallel Programming in C with MPI and OpenMP*. McGraw-Hill, New York, 2004. ISBN 0-072-82256-2.
- Snir, Marc; Otto, Steve; Huss-Lederman, Steven; Walker, David; Dongarra, Jack. *MPI: The Complete Reference, 2nd Edition, Volume 1*. MIT Press, Cambridge, Massachusetts, 1998. ISBN 0-262-69215-5.

© Copyright IBM Corp. 2013. All rights reserved.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Online resources

These websites are also relevant as further information sources:

- Compiler-related topics:
  - IBM XL C and C++ compilers

    http://www.ibm.com/software/awdtools/xlcpp/
  - XL Fortran Advanced Edition for Blue Gene

    http://www.ibm.com/software/awdtools/fortran/xlfortran/features/bg/
  - XL Fortran Compilers

    http://www.ibm.com/software/awdtools/fortran/
- Debugger-related topics:
  - GDB: The GNU Project Debugger

    http://www.gnu.org/software/gdb/gdb.html
  - GDB documentation:

    http://www.gnu.org/software/gdb/documentation/
- Engineering and Scientific Subroutine Library (ESSL) and Parallel ESSL

  http://www.ibm.com/systems/software/essl/index.html
- GCC, the GNU Compiler Collection

  http://gcc.gnu.org/
- General Parallel File System

  http://www.ibm.com/systems/software/gpfs/
- Intel MPI Benchmarks is formerly known as "Pallas MPI Benchmarks."

  http://www.intel.com/cd/software/products/asmo-na/eng/219848.htm
- Mathematical Acceleration Subsystem

  http://www.ibm.com/software/awdtools/mass/index.html
- Message Passing Interface Forum

  http://www.mpi-forum.org/
- MPI Performance Topics

  http://www.llnl.gov/computing/tutorials/mpi_performance/
- The OpenMP API Specification:

  http://www.openmp.org
- Danier, CJ, "What is Direct Memory Access (DMA)?"

  http://cnx.org/content/m11867/latest/

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 website:

http://www.redbooks.ibm.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Help from IBM

IBM Support and downloads

http://www.ibm.com/support

IBM Global Services

http://www.ibm.com/services

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

References

1. The MPI Forum. The MPI message-passing interface standard. May 1995:

   http://www.mcs.anl.gov/mpi/standard.html

2. OpenMP application programming interface (API):

   http://www.openmp.org

3. GCC, the GNU Compiler Collection:

   http://gcc.gnu.org/

4. Engineering and Scientific Subroutine Library (ESSL):

   http://www.ibm.com/systems/software/essl/index.html

5. Snir, Marc, et. al. *MPI: The Complete Reference, 2nd Edition, Volume 1*. MIT Press, Cambridge, Massachusetts, 1998. ISBN 0-262-69215-5.

6. Gropp, William, et. al. *MPI: The Complete Reference, Volume 2 - The MPI-2 Extensions*. MIT Press, Cambridge, Massachusetts, 1998. ISBN 0-262-69216-3.

7. See note 1.

8. See note 2.

9. See note 3.

10. See note 4.

11. Ganier, C J. ,"What is Direct Memory Access (DMA)?"

    http://cnx.org/content/m11867/latest/

12. See note 1.

13. See note 2.

14. Quinn, Michael J. *Parallel Programming in C with MPI and OpenMP*. McGraw-Hill, New York, 2004. ISBN 0-072-82256-2.

15. Snir, Marc, et. al. *MPI: The Complete Reference, 2nd Edition, Volume 1*. MIT Press, Cambridge, Massachusetts, 1998. ISBN 0-262-69215-5.

16. Gropp, William, et. al. *MPI: The Complete Reference, Volume 2 - The MPI-2 Extensions*. MIT Press, Cambridge, Massachusetts, 1998. ISBN 0-262-69216-3.

© Copyright IBM Corp. 2013. All rights reserved.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Index

A
A2 3, 15, 26–28, 53, 79, 83, 93
abstract device interface 67
addr2line utility 109–113
application binary interface 16
application-specific integrated circuit 10, 26
arithmetic logic unit 67
atomic operation 53

B
basic input/output system 11

C
Code Development and Tools Interface 105, 159
compute node 4–5, 7, 9–13, 16–17, 25, 54–56, 63, 73, 88, 90, 93, 95–96, 105–106, 122, 129, 142, 145
Compute Node Kernel 4, 9, 13, 25, 93, 129, 142
core file 112

D
debug server 105
direct memory access 11, 68, 160
double data RAM 27–28

E
end node 2, 6, 8, 54, 79, 87, 89–90, 92–96, 105–106, 111
Engineering and Scientific Subroutine Library 4, 11, 92, 160
environment variable 19, 29–30, 68, 84, 88, 90, 93, 95, 130–131, 133, 135–136, 139–142, 145, 148–149
executable and linking format 54, 88, 142

F
first-in, first-out queue 22, 130, 132–133
floating-point register 142
floating-point unit 6, 83, 97

G
gdbserver 105
gdbtool 107–108
General Parallel File System 4–5, 95, 159–160
general-purpose register 53
GNU C Library 4, 11, 17, 19, 59, 81, 83–84, 112
GNU Compiler Collection 4, 11, 16, 74–77, 80–82, 84–87, 93–95, 123, 126–127, 135–136, 139, 148, 160
GNU Project Debugger 4, 7, 81, 105–108, 142, 160

H
high-performance computing 56, 65
high-throughput computing 65
hwthread 122–123

I
I/O node 5, 8–10, 12, 14, 54, 56, 80, 87–88, 92–93, 95–96, 105–108, 122, 145
infinity 83
internet protocol 5, 105–108
ITSO xi

L
L1 prefetcher 26–28, 34, 63

M
Mathematical Acceleration Subsystem 92, 160
memory access 11, 139, 160
Message Passing Interface 4–6, 11, 15, 65–77, 87, 93, 97, 112, 116–119, 122–123, 125–126, 129–132, 135–136, 139–140, 142, 149, 155, 159–160
message unit system programming interface 66, 74, 130, 132, 140
messaging unit 63, 69, 130, 132–133
midplane 116
mmap 7, 29–30, 62, 64

N
Native POSIX Thread Library 4, 17
Network File System 5
not a number 83

O
Open multi-processing 4, 11, 15, 65–66, 73, 77, 83–84, 159–160
Open Source Software 159

P
Parallel Active Messaging Interface 66, 74, 88, 125–127, 130–134, 136, 139–142
Portable Operating System Interface 4, 17, 56, 62, 145
printf 111
process count 14
process identifier 16–17, 61–62, 64, 91, 108
procid 122–124
Python interpreter 96

Q
quad-processing extensions 72, 79, 81–83, 97–99
quantum chromodynamics 118

R
random access memory 11, 26

© Copyright IBM Corp. 2013. All rights reserved.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Reliability, availability, and serviceability 4
reliability, availability, and serviceability 4
RPM package manager 6, 92
runjob 14–15, 30, 90–91, 97, 107, 123, 145

S

shared libraries 5, 83, 85, 87–88, 90, 96
single instruction, multiple data 6, 83, 97–98
sockets 5, 11, 56, 59–61
special-purpose registers 143
speculative thread 83
static libraries 85
symmetrical multiprocessing 26
system call 56
system programming interface 20, 34, 53, 56, 62–63, 66, 76, 122, 130

T

thread group identifier 17
thread identifier 16–17, 20, 64
torus 73
torus network 67
transactional memory 83
Transmission Control Protocol 5, 105

U

User Datagram Protocol 5

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IBM Redbooks

IBM System Blue Gene Solution: Blue Gene/Q Application Development

(0.2”spine)
0.17”<->0.473”
90<->249 pages

IBM System Blue Gene Solution
Blue Gene/Q
Application Development



Understand the Blue Gene/Q programming environment

See available parallel programming paradigms

Learn how to run and debug programs

This IBM Redbooks publication is one in a series of IBM books written specifically for the IBM System Blue Gene supercomputer, Blue Gene/Q, which is the third generation of massively parallel supercomputers from IBM in the Blue Gene series. This document provides an overview of the application development environment for the Blue Gene/Q system. It describes the requirements to develop applications on this high-performance supercomputer.

This book explains the unique Blue Gene/Q programming environment. This book does not provide detailed descriptions of the technologies that are commonly used in the supercomputing industry, such as Message Passing Interface (MPI) and Open Multi-Processing (OpenMP). References to more detailed information about programming and technology are provided.

This document assumes that readers have a strong background in high-performance computing (HPC) programming. The high-level programming languages that are used throughout this book are C/C++ and Fortran95. For more information about the Blue Gene/Q system, see “IBM Redbooks” on page 159.

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

SG24-7948-01 ISBN 0738438235

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP